QUESTÕES ALGORÍTMICAS DE NATUREZA COMBINATÓRIA

Cristina Gomes Fernandes

TRABALHO SISTEMATIZADO
APRESENTADO
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA O CONCURSO DE
LIVRE-DOCÊNCIA
JUNTO AO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

São Paulo, março de 2004

QUESTÕES ALGORÍTMICAS DE NATUREZA COMBINATÓRIA

CRISTINA GOMES FERNANDES

RESUMO. São expostas as nossas contribuições na resolução de algumas questões algorítmicas nas áreas de combinatória, otimização combinatória e teoria dos grafos.

SUMÁRIO

1. INTRODUÇÃO

O estudo de problemas de combinatória, otimização combinatória e teoria dos grafos do ponto de vista algorítmico tem diversas facetas. Dentre elas estão por exemplo a determinação da complexidade computacional do problema em questão, a busca por algoritmos eficientes, exatos ou não, dependendo do problema, e a busca de implementações tão eficientes quanto possível para os algoritmos existentes.

Essas três facetas estão interligadas, já que, ao considerar-se um problema, de decisão ou de otimização, é natural tentar determinar a sua complexidade. Essa informação indica o tipo de resultado que se pode esperar para o problema em questão. Se o problema for NP-difícil, pode-se partir em busca de classes de instâncias onde ele se torna tratável, ou, no caso de problemas de otimização, pode-se, por exemplo, buscar bons algoritmos de aproximação.

Já de posse de um algoritmo para um determinado problema, há ainda a fase de, digamos, "acabamento", que visa responder à questão de qual é a melhor maneira de se implementar tal algoritmo. A busca de implementações eficientes bem como a condução de estudos experimentais comparativos entre algoritmos competitivos é uma parte importante da área de projeto e análise de algoritmos.

Este texto aborda problemas e resultados que se enquadram nessas três linhas. Os resultados de complexidade envolvem tanto técnicas usuais de redução de problemas quanto técnicas mais recentes de reduções que preservam aproximabilidade. A parte de projeto de algoritmos discutida neste texto concentra-se especialmente no projeto de algoritmos de aproximação para problemas de otimização NP-difíceis, mas inclui também refinamentos de análises de algoritmos já existentes, bem como propostas de algoritmos exatos para problemas combinatórios tanto de decisão quanto de otimização. O texto aborda também alguns estudos experimentais desenvolvidos ou em andamento. Esses estudos envolvem geralmente a implementação de algoritmos relativamente sofisticados, teste, análise e refinamento de tais algoritmos. Muitas vezes, especialmente na parte de refinamento das implementações, surgem questões algorítmicas simples mas inquietantes, que envolvem, por exemplo, a escolha de estruturas de dados adequadas, o uso de algoritmos conhecidos para resolver subproblemas, etc.

O texto está organizado da seguinte maneira. A próxima seção fala de algoritmos de aproximação. Cada uma das demais seções trata de um problema ou de uma classe de problemas correlatos em que trabalhamos.

2. Algoritmos de aproximação

O desenvolvimento de algoritmos de aproximação surgiu em resposta à dificuldade computacional de muitos dos problemas de otimização combinatória: em termos técnicos, muitos são NP-difíceis. Nessa situação, é razoável sacrificar a otimalidade em troca de uma aproximação de boa qualidade que possa ser eficientemente calculada. Esse compromisso entre perda de otimalidade e ganho em eficiência é o paradigma dos algoritmos de aproximação. Convém observar que um algoritmo de aproximação não é simplesmente uma heurística: ele garante encontrar, eficientemente, um elemento do domínio cujo valor guarda uma relação pré-estabelecida com o valor ótimo.

No início da década de 70, Garey, Graham e Ullman (1972), bem como Johnson (1974), formalizaram o conceito de algoritmo de aproximação. O conceito já estava implícito em um trabalho de Graham (1966) sobre um problema de escalonamento em máquinas paralelas e em um trabalho de Erdős (1967) sobre grafos bipartidos. Na década de 90, o estudo de algoritmos de aproximação passou a receber um tratamento mais sistemático, com a formalização e o uso de técnicas e ferramentas aplicáveis a toda uma gama de problemas.

É importante mencionar também o aparecimento de certos resultados negativos de aproximabilidade: para alguns problemas, aproximar é tão difícil quanto resolver. Em termos mais técnicos, alguns problemas não admitem algoritmos de aproximação com razão melhor que um certo limiar, a menos que P = NP. As teorias nessa direção foram impulsionadas na década de 90 pelas descobertas de Arora *et al.* (1992), que provaram resultados desse tipo para vários problemas usando caracterizações probabilísticas da classe NP.

O desenvolvimento de algoritmos de aproximação e de provas de inaproximabilidade é uma das linhas de pesquisa que mais cresceu ultimamente na área de otimização combinatória e teoria da computação. Esta observação encontra respaldo na grande quantidade de artigos de pesquisa que surgiram nos últimos anos. Vários livros sobre o assunto também foram publicados recentemente: Ausiello *et al.* (1999), Hochbaum (1997), Mayr *et al.* (1998) e Vazirani (2001). Outro indício da efervescência da área é a grande quantidade de teses de doutorado concluídas na década de 90, algumas introduzindo teorias revolucionárias.

São várias as nossas contribuições nessa linha de pesquisa. Além de uma série de resultados que, por tratarem de algum problema específico, serão discutidos nas seções a seguir, há um livro sobre o assunto (em português, escrito por Carvalho *et al.* (2001)). O livro discute desde algoritmos de aproximação clássicos, como o de Graham (1966) e o de Christofides (1976), até técnicas sofisticadas, desenvolvidas na última década, como o método primal-dual para aproximação e o uso de programação semidefinida no projeto de algoritmos de aproximação.

Um algoritmo de aproximação é uma *α-aproximação* para um problema de otimização combinatória que busca minimizar (maximizar) uma função se ele é polinomial e, para toda instância do problema, a razão (o inverso da razão) entre o valor da solução viável produzida pelo algoritmo para essa instância e o valor de uma solução ótima não ultrapassa (não fica abaixo de) α. Aqui, α tanto pode ser uma constante quanto uma função e é uma *razão de aproximação* do algoritmo. Quanto mais próxima de 1 a razão, melhor.

3. Planaridade em grafos

Questões relacionadas à planaridade são fundamentais em teoria dos grafos e, por terem aplicações reais diversas, são também bem estudadas do ponto de vista algorítmico. Nesta seção, são apresentados os problemas e resultados de [1, 2, 3, 5, 6, 7].

3.1. **Subgrafo planar máximo.**

Existem vários problemas de otimização combinatória relacionados a planaridade. Um deles é o problema GT31 do livro de Ausiello *et al.* (1999) — o *problema do subgrafo planar máximo* (PSPM): dado um grafo, encontrar um subgrafo planar com o número máximo de arestas. O PSPM é NP-difícil, o que motiva a busca por algoritmos de aproximação. Cimikowski (1995) analisou vários algoritmos conhecidos para o PSPM e mostrou que nenhum deles tem razão de aproximação maior que 1/3. (A razão de 1/3 é obtida trivialmente por qualquer algoritmo que produz um subgrafo planar maximal.) Uma de nossas contribuições é o primeiro algoritmo de aproximação para o PSPM com razão de aproximação acima de 1/3 [5]. Provamos primeiramente uma razão de 2/5 para este algoritmo [5] e mais tarde conseguimos melhorar esse resultado, demonstrando por meio de uma análise justa e bastante técnica que o algoritmo é uma 4/9-aproximação [3, 6]. Esta é até hoje a melhor aproximação para o PSPM. Trata-se de um algoritmo simples, mas que utiliza como subrotina um sofisticado algoritmo para o problema da paridade em matróides gráficos. A complexidade resultante é de $O(m^{3/2} n \log^6 n)$, onde n e m são, respectivamente, o número de vértices e o número de arestas do grafo de entrada, que não é muito boa. Numa tentativa de obter um algoritmo mais eficiente para o problema, obtivemos uma aproximação um pouco pior, de razão 7/18, que consome tempo linear em grafos de grau limitado [5]. No mesmo artigo, mostramos ainda que o PSPM é Max SNP-difícil, o que implica, segundo o resultado de Arora *et al.* (1992), que existe um $\epsilon > 0$ para o qual não existe uma $(1-\epsilon)$-aproximação para o PSPM, a menos que P = NP. Faria *et al.* (1999) melhoram esse resultado, mostrando que o PSPM é Max SNP-difícil mesmo para grafos cúbicos.

3.2. **Subgrafo planar de peso máximo.** Uma extensão natural do PSPM é a sua versão com pesos. As nossas aproximações acima não se aplicam nesse caso. Mais difícil que a sem pesos, a variante com pesos do PSPM se mostrou especialmente interessante por ter possibilitado o inesperado uso de técnicas bem-conhecidas de aproximação para o problema de Steiner em grafos. Projetamos a primeira (e, do que sabemos, única até agora) aproximação para o problema com razão maior que 1/3 [7]. (A razão de 1/3 é obtida trivialmente por um algoritmo que, para um grafo conexo com pesos nas arestas, produz uma árvore geradora de peso máximo.) Nosso algoritmo é uma $(1/3 + 1/72)$-aproximação. Infelizmente, o aumento na razão de aproximação não é muito grande e a análise é longa e bastante técnica.

3.3. **Subgrafo máximo de um dado genus.** Um resultado menor que obtivemos enquanto trabalhávamos com o PSPM refere-se ao *problema do subgrafo máximo de genus d*: dado um grafo conexo, encontrar um subgrafo de genus no máximo d com número máximo de arestas. Trata-se de um algoritmo simples cuja razão de aproximação é pelo menos 1/4 [3].

3.4. **Algoritmos para teste de planaridade.** Mais recentemente iniciamos um estudo de vários algoritmos existentes para o *teste de planaridade*: dado um grafo, decidir se ele é ou não planar. É comum considerar também a sua variante que, além da resposta SIM ou NÃO, produz um certificado à resposta: um desenho plano do grafo (mais precisamente, uma representação combinatória do mesmo), caso ele seja planar, ou uma subdivisão do $K_{3,3}$ ou do K_5 no grafo, caso ele não seja planar.

O primeiro algoritmo linear para o teste de planaridade deve-se a Hopcroft e Tarjan (1974). O algoritmo deles é uma engenhosa implementação do método de Auslander e Parter (1961) e Goldstein (1963). (Algumas notas sobre o algoritmo foram feitas por Deo (1976) e mais detalhes foram apresentados por Williamson (1980/1985) e Reingold, Nievergelt, e Deo (1977).)

O segundo método de teste de planaridade comprovadamente linear deve-se a Lempel, Even e Cederbaum (LEC) (1967). A implementação linear desse método utiliza o algoritmo linear projetado por Even e Tarjan (1976) para construir uma st-numeração num grafo e a estrutura de dados conhecida como PQ-árvore proposta por Booth e Lueker (BL) (1976). (Chiba *et al.* (1985) aumentaram as operações em PQ-árvores de maneira que um desenho plano pudesse ser construído também em tempo linear.)

Todos estes algoritmos de planaridade são reconhecidamente complexos. Recentemente, dois algoritmos lineares mais simples foram propostos na literatura, um por Shih e Hsu (SH) (1993/1999), e outro por Boyer e Myrvold (BM) (1999/2004). Ambos apresentam idéias similares e muito interessantes, e podem ser vistos como implementações do método de LEC.

Baseados na descrição do algoritmo de SH feita por Thomas (1997), produzimos uma descrição do método de LEC que, na nossa opinião, aumenta o entendimento dos algoritmos de BL, de SH e de BM (todos implementações do método de LEC) [2]. Junto com essa descrição, disponibilizamos e descrevemos em detalhes uma implementação do algoritmo de SH desenvolvida por Noma (2003) sob nossa orientação (em conjunto com Pina). Esta é, até onde sabemos, a primeira e única implementação existente deste algoritmo. Num outro trabalho, complementamos e corrigimos a apresentação original do algoritmo de SH [1].

4. Problema de Steiner em grafos

O *problema de Steiner* (problema ND8 do livro de Ausiello *et al.* (1999)) é central na área de algoritmos de aproximação. A sua versão geométrica foi introduzida por Gauss em uma carta a Schumacher. (Uma cópia desta carta aparece, por exemplo, na capa do livro de Vazirani (2001).) Em termos de grafos, ele pode ser descrito da seguinte forma: dados um grafo G, um conjunto S de vértices de G e uma função não-negativa c definida nas arestas de G, que atribui a cada aresta um número que representa o "comprimento" da aresta, encontrar uma árvore em G de comprimento total mínimo que contenha todos os vértices de S. Este problema tem um amplo espectro de aplicações, que vão desde projeto de circuitos VLSI até a construção de árvores filogenéticas.

Uma árvore em G que contém os vértices de S é chamada de *árvore de Steiner*. Os vértices de S são os chamados *terminais*, enquanto que os vértices que não estão em S são chamados de *vértices de Steiner*.

Nosso algoritmo de aproximação para a versão com pesos do PSPM [7] busca por um *cacto triangular* (grafo em que todo bloco tem no máximo 3 vértices) no grafo dado de peso o maior possível. Esse problema pode ser reescrito como uma versão particular do problema de encontrar uma árvore de Steiner 3-restrita mínima. Uma árvore de Steiner T é *3-restrita* quando todo vértice de Steiner em T é adjacente em T a exatamente 3 vértices, todos eles terminais. Essa variante do problema de Steiner é bem estudada até por ter sido a base da primeira aproximação com razão menor que uma constante menor que 2 para o problema de Steiner: o algoritmo das árvores 3-restritas de Zelikovsky (1993). É sabido que a análise original do algoritmo de Zelikovsky tinha alguns problemas. Durante vários anos, não havia na literatura uma análise satisfatória desse algoritmo. O nosso trabalho na versão com pesos do PSPM [7] colaborou um pouco no sentido de prover alguns esclarecimentos sobre a análise do algoritmo das árvores 3-restritas. Ele é por exemplo citado no *survey* recente de Gröpl *et al.* (2001) sobre algoritmos para o problema de Steiner.

Existem dezenas de variantes do problema de Steiner, várias delas mencionadas nos comentários de Ausiello *et al.* (1999) sobre o problema de Steiner (ND8). A seguir, algumas destas variantes são apresentadas e são comentadas as nossas contribuições [8, 11] ou o nosso interesse em cada uma delas.

4.1. Problema de Steiner com coleta de prêmios.

O *problema de Steiner com coleta de prêmios* (*prize-collecting Steiner tree*) (PCST), apesar do que o nome indica, é definido usualmente da seguinte maneira. Seja G um grafo, c uma função não-negativa definida nas arestas de G e π uma função não-negativa definida nos vértices de G. O PCST consiste no seguinte: dados G, c e π, encontrar uma árvore T em G que minimize

$$\textstyle\sum_{e\in E(T)} c(e) + \sum_{v\notin V(T)} \pi(v).$$

Algoritmos de aproximação para o PCST foram usados por Chudak, Roughgarden e Williamson (2001), e também por Garg (1996), no estudo de outros problemas de otimização.

As melhores aproximações conhecidas para o PCST baseiam-se no método primal-dual de aproximação. Este método tem sido usado no projeto de algoritmos exatos e de aproximação para vários problemas. Diferentes formulações como problemas de programação inteira podem levar a diferentes algoritmos. Uma de nossas contribuições aqui foi uma formulação diferente, e um novo algoritmo para o PCST baseado nela [11].

A versão *com raiz* do PCST pede que a árvore T contenha um dado vértice r, chamado de raiz. Goemans e Williamson (GW) (1995) usaram o método primal-dual para derivar uma $(2 - 1/(n-1))$-aproximação para a versão com raiz do PCST, onde n é o número de vértices do grafo de entrada. Aplicando este algoritmo uma vez com cada um dos vértices como raiz, eles obtiveram uma $(2-1/(n-1))$-aproximação para o PCST. Tal algoritmo consome tempo $O(n^3 \log n)$. Johnson, Minkoff e Phillips (JMP) (2000) propuseram uma modificação do algoritmo de GW que permite executar o método primal-dual uma única vez, resultando em um algoritmo $O(n^2 \log n)$

para o PCST. Eles afirmam que essa modificação atinge a mesma razão que o algoritmo de GW, porém isso não é verdade sempre. Cole *et al.* (2001) referem-se à razão correta do algoritmo de JMP, mas não apresentam uma demonstração completa dessa razão.

O nosso algoritmo para o PCST é na verdade muito semelhante ao algoritmo de JMP. Ao estudarmos este último, deparamo-nos com um problema em sua análise que invalida a razão de aproximação anunciada pelos seus autores. A degradação é pequena: o algoritmo é uma 2-aproximação. Porém a análise que nos permitiu concluir a razão de 2 para o algoritmo de JMP é delicada e justa [11]. O exemplo apresentado como prova de que a razão originalmente anunciada não vale [11] serve também para mostrar que a nossa análise é justa.

Foi durante a tentativa de escrever uma análise correta para o algoritmo de JMP que derivamos as idéias para o nosso algoritmo, que é uma $(2-2/n)$-aproximação para o PCST que também executa o método primal-dual uma única vez, ou seja, consome tempo $O(n^2 \log n)$ [11]. Uma outra curiosidade é que, diferentemente do algoritmo de JMP, não é de todo óbvio que o nosso algoritmo é polinomial.

4.2. **Problema de Steiner com qualidade de serviço.** Diversas variantes do problema de Steiner aparecem no contexto de distribuição de mídia, que é geralmente feita através de uma árvore. Abaixo, uma dessas variantes é descrita.

Considere um grafo G, um vértice s, chamado de raiz, e um inteiro não-negativo r_v para cada vértice v. O número r_v é o *requisito de velocidade* (*rate requirement*) do vértice v. Uma *árvore multicast* é uma árvore em G que contém s e todos os vértices de G com requisito de velocidade positivo.

Para um conjunto não-vazio X de vértices, seja $r(X)$ o maior requisito de velocidade entre os vértices de X. Para uma aresta e na árvore multicast T, denotamos por X_e o conjunto dos vértices da componente de $T-e$ que não contém s. A *largura de banda* de uma aresta e em T é o número $r(X_e)$.

O *problema de Steiner com qualidade de serviço* (*quality of service Steiner tree*) (QoS) consiste em: dados um grafo G, uma função não-negativa c definida nas arestas de G, um vértice raiz s e um requisito de velocidade r_v para cada vértice v de G, encontrar uma árvore multicast T que minimiza a soma do custo de cada uma de suas arestas vezes a largura de banda da aresta. Ou seja, minimiza

$$\textstyle\sum_{e \in E(T)} c_e t_e,$$

onde t_e é a largura de banda da aresta e em T.

Charikar, Naor e Schieber (CNS) (2000) propuseram o primeiro algoritmo com razão constante para o QoS. Num primeiro passo, todos os requisitos de velocidade são arredondados para cima, para a potência de 2 mais próxima. Claramente isso no máximo dobra o valor de uma solução para o problema. Num segundo passo, árvores de Steiner são produzidas para cada requisito de velocidade separadamente, com uma certa aproximação, digamos, α. A melhor razão de aproximação conhecida atualmente para o problema de Steiner é 1,55, portanto pode-se usar $\alpha \approx 1{,}55$. A união destas árvores é a solução produzida pelo algoritmo de CNS, que é uma 4α-aproximação para o QoS. Usando ainda uma técnica probabilística, Charikar, Naor e Schieber (2000) reduzem a razão de aproximação para $e\alpha \approx 4{,}21$, onde $e = 2{,}71828\ldots$. Karpinski *et al.* (2003) recentemente propuseram uma aproximação ainda melhor, de razão 3,802.

Apresentamos uma 4,311-aproximação primal-dual probabilística para QoS, que foi derivada de uma formulação do problema como um programa linear inteiro diferente das formulações usadas previamente [8]. O resultado infelizmente não ultrapassa a melhor razão conhecida para o problema, porém o algoritmo de Karpinski *et al.* (2003), que atinge a melhor razão, utiliza árvores k-restritas, como os conhecidos algoritmos para o problema de Steiner. Como o uso de árvores k-restritas implica em algoritmos com alta complexidade computacional (especialmente se o valor escolhido de k for alto), torna-se interessante ter algoritmos alternativos, mais rápidos, com razões de aproximação semelhantes.

4.3. **Problema de Steiner com grupos.** O *problema de Steiner para grupos* (*group Steiner tree*) (GST) consiste no seguinte: dados um grafo G, uma função não-negativa c definida nas arestas de G, um inteiro k e conjuntos $g_1, \ldots, g_k$ de vértices de G, encontrar uma árvore em G que contenha pelo menos um vértice de cada g_i e tenha comprimento total mínimo. Um conjunto g_i arbitrário é chamado de *grupo*. Há uma redução do problema da cobertura mínima por conjuntos para o GST que preserva aproximação. Assim, não só o GST é NP-difícil como, por um resultado de Feige (1998) para o problema da cobertura por conjuntos, a existência de uma $(1-\delta)\ln k$-aproximação para o GST implicaria que NP $\subset$ DTIME$(n^{O(\log\log n)})$, onde n é o número de vértices de G. O melhor algoritmo de aproximação conhecido para o GST deve-se a Garg, Konjevod e Ravi (2000) e tem uma razão de aproximação de $O(\log n \log\log n \log k)$. Trabalhamos (com Nierhoff) nesse problema em busca de uma $O(\log k)$-aproximação. Alguns resultados parciais foram obtidos porém ainda há muito a fazer para obter-se um resultado significativo.

4.4. **Estudos experimentais.** Atualmente estamos trabalhando em dois estudos experimentais relacionados ao problema de Steiner. Um deles envolve diversas implementações propostas na literatura para o algoritmo de Goemans e Williamson (1995) para o *problema da floresta de Steiner*: a implementação original sugerida por Goemans e Williamson, a implementação de Cole *et al.* (2001), a implementação de Klein (1994) e a implementação de Gabow, Goemans e Williamson (1993).

O segundo estudo experimental que estamos conduzindo envolve os diversos algoritmos de aproximação para o problema de Steiner que utilizam árvores k-restritas. (Uma árvore de Steiner T é *k-restrita* se toda subárvore maximal de T cujos vértices internos são de Steiner tem no máximo k vértices terminais.) São eles o algoritmo das árvores 3-restritas (1993), o algoritmo de Zelikovsky (1996), o algoritmo de Karpinski e Zelikovsky (1997) e o algoritmo de Robins e Zelikovsky (2000).

5. Problemas de conexidade em grafos

O estudo de conexidade em teoria dos grafo tem importantes aplicações em áreas de projeto e confiabilidade de redes. Diversos algoritmos de aproximação têm sido desenvolvidos para o problema de encontrar subgrafos satisfazendo certos requisitos de conexidade. Nesta seção, são apresentados os problemas e resultados de [4, 9, 10, 12].

5.1. **Subgrafo gerador k-conexo mínimo.** Trabalhamos no problema ND28 do livro de Ausiello *et al.* (1999) — o *problema do subgrafo gerador k-aresta-conexo de tamanho mínimo* (k-SGAC): dados um inteiro positivo k e um grafo G k-aresta-conexo, encontrar um subgrafo gerador k-aresta-conexo de G com número mínimo de arestas.

Esse problema é NP-completo, mesmo para $k = 2$: se o grafo G é hamiltoniano, um subgrafo gerador 2-aresta-conexo de G com número mínimo de arestas é um circuito hamiltoniano. Por muito tempo, 2 era a melhor razão de aproximação conhecida para o k-SGAC. Tal razão vem do fato que todo grafo k-aresta-conexo tem grau mínimo pelo menos k e, conforme Mader (1971/1972) demonstrou, todo subgrafo gerador k-aresta-conexo minimal tem no máximo kn arestas.

Karger (1999) apresentou uma $(1 + O(\sqrt{(\log n)/k}))$-aproximação para o k-SGAC. A razão desse algoritmo é menor que 2 apenas quando $k \gg \log n$. Além desse, existem outros algoritmos na literatura com razão menor que 2 para valores particulares de k. Khuller e Vishkin (1994) por exemplo projetaram uma 1,5-aproximação para o 2-SGAC. Como observado por Khuller e Raghavachari (1996), combinando esta aproximação com umas idéias de Cheriyan, Kao e Thurimella (1993), pode-se facilmente obter uma $(2 - 1/k)$-aproximação para o k-SGAC.

Khuller e Raghavachari (1996) apresentaram o primeiro algoritmo que atinge uma razão menor que uma constante menor que 2 para o k-SGAC. Mais precisamente, eles mostraram que o algoritmo deles é uma 1,85-aproximação. Uma de nossas contribuições foi uma análise mais precisa do algoritmo de Khuller e Raghavachari, que mostrou que ele é uma 1,75-aproximação para o k-SGAC [12].

A análise de Khuller e Raghavachari do algoritmo deles é na verdade melhor que 1,85 para valores pequenos de k: é 1,5 para $k = 2$, 1,666... para $k = 3$, 1,75 para $k = 4$, 1,733... para $k = 5$, etc. Estas delimitações são justas para $k = 2$ e 3. Nossa análise melhorou as delimitações de Khuller e Raghavachari para todo $k \geq 4$. Em particular, para $k = 4$, obtivemos a razão de 1,65 e para $k = 5$, obtivemos 1,68 [12].

Khuller e Vishkin (1994) introduziram o seguinte conceito: um *esboço de árvore* (*tree-carving*) num grafo $G = (V, E)$ é uma partição de V em subconjuntos $V_1, \ldots, V_a$ com as seguintes propriedades. Cada subconjunto consiste em um nó em uma árvore Γ. Para cada vértice v em V_j, todos os vizinhos de v em G pertencem a V_j ou a um V_i adjacente a V_j na árvore Γ. Eles usaram este conceito para provar a cota de 1,5 na razão do algoritmo deles para $k = 2$. Apresentamos uma generalização do conceito de esboço de árvore e usamos essa generalização para mostrar que o algoritmo de Khuller e Raghavachari é uma 1,7-aproximação para o k-SGAC para k suficientemente grande [12].

O melhor algoritmo conhecido para o problema atualmente é uma $(1 + 2/(k + 1))$-aproximação projetada por Cheriyan e Thurimella (2000) na mesma época em que melhoramos a análise do algoritmo de Khuller e Raghavachari. Numa primeira fase, o algoritmo de Cheriyan e Thurimella encontra um conjunto M de arestas de cardinalidade mínima tal que todo vértice do grafo é incidente a pelo menos k arestas em M. Numa segunda fase, o algoritmo encontra um conjunto minimal F de arestas, disjunto de M, tal que o grafo induzido por $M \cup F$ é k-aresta-conexo. Este grafo é a saída do algoritmo, que consome tempo $O(k^3n^2 + m^{1.5}(\log n)^2)$, onde n e m são, respectivamente, o número de vértices e arestas do grafo. O algoritmo de Khuller e Raghavachari por sua vez é mais rápido, consumindo tempo $O(k^2n^2)$.

Finalmente mostramos que o 2-SGAC é Max SNP-difícil, ou seja, existe um $\epsilon > 0$ para o qual não existe uma $(1 + \epsilon)$-aproximação a menos que P = NP [12].

5.2. **Problemas de multicorte.** Uma outra categoria de problemas ligada a conexidade são os problemas de cortes e multicortes. O *problema do multicorte mínimo* (k-MC) consiste no seguinte: dados um grafo G e uma coleção de k pares $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ de vértices distintos de G, encontrar um conjunto de arestas de tamanho mínimo cuja remoção desconecta cada s_i do t_i correspondente. (Este é o problema ND28 do livro de Ausiello *et al.* (1999).)

A versão com pesos do k-MC é denotada por k-WMC. O caso particular dela em que $k = 1$ é caracterizado pelo famoso *Teorema do Fluxo Máximo Corte Mínimo* de Ford e Fulkerson (1956), e é solúvel em tempo polinomial. Para $k = 2$, uma variante do Teorema do Fluxo Máximo Corte Mínimo vale (veja Hu (1963) e Itai (1978)) e implica que o 2-WMC é solúvel em tempo polinomial também. Dahlhaus *et al.* (1994) mostraram que, para $k \geq 3$, o k-WMC é NP-difícil.

A melhor razão de aproximação para o k-WMC é $O(\log k)$ e deve-se a Garg, Vazirani e Yannakakis (1997). Muita pesquisa tem sido feita em busca de algoritmos melhores para classes especiais de grafos. Para grafos planares, Tardos e Vazirani (1993) apresentam um teorema aproximado de Fluxo Máximo Corte Mínimo e um algoritmo com uma razão de aproximação constante.

Garg, Vazirani e Yannakakis (1996) consideraram o caso em que o grafo de entrada é uma árvore. Eles observaram que o k-MC restrito a estrelas (árvores de altura 1) é equivalente, inclusive quanto à aproximação, ao *problema da cobertura mínima por vértices*. Segue daí que o k-MC em estrelas é NP-difícil e Max SNP-difícil. Na verdade, obter uma razão de aproximação melhor que 2 parece bem difícil, já que é uma questão em aberto famosa a existência ou não de um algoritmo de aproximação para cobertura mínima por vértices com razão melhor que 2. Garg, Vazirani e Yannakakis (1996) propuseram uma 2-aproximação para o k-MC em árvores.

Em nossos trabalhos [4, 9, 10], duas variantes do k-MC se mostraram úteis. A primeira trata do *problema do multicorte mínimo de vértices* (*vertex multicut*) (k-VMC), que consiste no seguinte: dados um grafo G e uma coleção de k pares $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ de vértices de G não-adjacentes distintos, chamados *terminais*, encontrar um conjunto mínimo de vértices não-terminais cuja remoção desconecta cada s_i do t_i correspondente. A segunda trata do *problema do multicorte irrestrito mínimo de vértices* (*unrestricted vertex multicut*) (k-UVMC), que consiste em: dados um grafo G e uma coleção de k pares $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ de G, chamados *terminais*, encontrar um conjunto mínimo de vértices de G cuja remoção desconecta cada s_i do t_i correspondente. (Nessa variante, terminais podem ser removidos.) O k-VMC é pelo menos tão difícil quanto o k-UVMC, inclusive quanto à aproximação [9]. De fato, de uma instância do k-UVMC, podemos obter uma instância do k-VMC adicionando, para cada s_i, um novo vértice s'_i, adjacente apenas a s_i, e para cada t_i, um vértice novo t'_i, adjacente apenas a t_i. Cada par $\{s_i, t_i\}$ é substituído pelo novo par $\{s'_i, t'_i\}$. Resolver esta instância do k-VMC é equivalente a resolver a instância original do k-UVMC.

Garg, Vazirani e Yannakakis (1997) consideraram a versão com pesos do k-VMC, denotada aqui por k-WVMC. Eles apresentaram uma $O(\log k)$-aproximação para esta variante em grafos arbitrários.

Primeiramente, mostramos que o k-VMC é NP-difícil em árvores de grau limitado, enquanto que sua variante irrestrita é mais fácil: é polinomial em árvores, mas torna-se NP-difícil em grafos série-paralelos de grau limitado [9].

A largura arbórea de um grafo G pode ser definida utilizando-se a noção de decomposição arbórea de G. O estudo de decomposições arbóreas começou nos anos 80 e teve diversas conseqüências em duas áreas: teoria dos grafos e projeto de algoritmos. Na primeira, decomposições arbóreas foram fundamentais na prova de Seymour e Robertson da famosa conjectura de Wagner sobre menores de grafos: toda família de grafos fechada sobre as operações de remoção e contração de arestas pode ser caracterizada excluindo-se um conjunto finito de grafos como menores. Na segunda, este conceito foi explorado com sucesso no projeto de algoritmos polinomiais para diversos problemas NP-difíceis em grafos com largura arbórea limitada.

Na verdade, o conceito de largura arbórea pode também ser usado para se obter bons algoritmos de aproximação para alguns problemas de otimização que continuam NP-difíceis mesmo quando restritos a grafos com largura arbórea limitada. No nosso caso, o k-UVMC é NP-difícil para grafos com largura arbórea no máximo 2 [9], pois esta classe coincide exatamente com os grafos série-paralelos. Um PTAS (*polynomial-time approximation scheme*) para um problema de minimização (maximização) consiste numa família $\{A_\epsilon : \epsilon > 0\}$ de algoritmos tal que A_ϵ é uma $(1+\epsilon)$-aproximação ($(1-\epsilon)$-aproximação) para o problema em questão, para todo $\epsilon > 0$. Projetamos um PTAS bastante simples para o k-UVMC em grafos com largura arbórea limitada [9].

Prosseguindo, derivamos uma redução que preserva aproximação, do k-MC para o k-UVMC [9]. Se a instância do k-MC tem grau e largura arbórea limitada, então a instância correspondente do k-UVMC obtida pela redução tem largura arbórea limitada. Combinando a redução com o PTAS, obtemos um PTAS para o k-MC em grafos com grau e largura arbórea limitada [9]. Uma implementação linear do PTAS para árvores de grau limitado também é descrita [9]. De acordo com o teorema 6.8 do livro de Garey e Johnson (1979), um FPTAS, que seria ainda melhor que um PTAS, neste caso não pode existir, a menos que P = NP. (Um FPTAS consiste numa família $\{A_\epsilon : \epsilon > 0\}$ de algoritmos tal que A_ϵ é uma $(1 \pm \epsilon)$-aproximação para o problema que consome tempo polinomial também em $1/\epsilon$.)

Posteriormente consideramos [4] a seguinte versão orientada do problema (*directed multicut*), introduzida por Klein *et al.* (1997) e denotada aqui por k-DMC: dados um grafo orientado D e k pares $(s_1, t_1), \ldots, (s_k, t_k)$ de vértices distintos de D, encontrar um conjunto mínimo de arcos cuja remoção garante que nenhuma das componentes fortemente conexas do grafo resultante contém um dos pares. Em outras palavras, ou todos os caminhos de s_i a t_i ou todos os caminhos de t_i a s_i foram quebrados.

Klein *et al.* (1997) mostraram uma $O(\log^2 k)$-aproximação para a versão com pesos do k-DMC em grafos orientados arbitrários. Conseguimos estender os nossos resultados acima para grafos orientados, exibindo um PTAS para o k-DMC em grafos orientados com largura arbórea *orientada* limitada [4]. O conceito de largura arbórea *orientada*, proposto por Johnson, Robertson, Seymour, e Thomas (2001), difere da versão usual de largura arbórea. Por exemplo, a classe dos grafos orientados com largura arbórea orientada zero consiste nos grafos orientados acíclicos.

Do ponto de vista de complexidade, mostramos que o k-MC é NP-difícil mesmo em árvores binárias (árvores com grau máximo três) [9]. Assim, na classe dos grafos de grau e largura arbórea limitados, que contém as árvores binárias, o k-MC é mais fácil (há um PTAS) que em grafos em geral, mas ainda é NP-difícil. Estes resultados indicam que não podemos eliminar nenhuma das três restrições no grafo de entrada — sem pesos, grau limitado e largura arbórea limitada — e ainda obter um PTAS [9]. Isso porque sabe-se que, para um problema Max SNP-difícil, existe um PTAS apenas se P = NP, e, lembre-se, o k-MC é Max SNP-difícil em estrelas. Ou seja, deixar o grau ilimitado no grafo de entrada torna o problema mais difícil. Mostramos que a versão com pesos do multicorte mínimo de arestas, o k-WMC, é Max SNP-difícil em árvores binárias [9], portanto admitir pesos também torna o problema mais difícil. Finalmente, mostramos que o k-MC é Max SNP-difícil numa classe de grafos (*walls*) que têm grau máximo três e largura arbórea ilimitada [9]. Ou seja, permitir grafos com largura arbórea ilimitada também torna o problema mais difícil. Para o caso orientado, mostramos resultados semelhantes [4]. Por exemplo, mostramos que o k-DMC é NP-difícil para grafos orientados com largura arbórea no máximo um e grau de entrada e saída no máximo três [4, 10].

6. Problemas envolvendo circuitos e caminhos em grafos

Serão descritos aqui quatro problemas e nossas contribuições em cada um deles [14, 15, 17]. O primeiro é uma variante do famoso TSP. Os dois seguintes são versões para grafos mistos de dois problemas também bem-conhecidos: o problema do carteiro chinês e o problema da cobertura mínima por circuitos. O último problema discutido é uma generalização do problema do caminho mais curto entre dois vértices num grafo orientado.

6.1. **Problema do UPS.**

Para apresentar o primeiro problema, é necessário introduzir um pouco de notação. Seja G o grafo completo sobre um conjunto V de vértices. Uma função l que atribui, a cada aresta uv de G, um valor não-negativo l_{uv}, representando o comprimento da aresta uv, é chamada de *função distância* em G. Diz-se que l *satisfaz a desigualdade triangular* se $l_{uv} \leq l_{uw} + l_{wv}$ para todo u, v e w em V.

Seja C um circuito hamiltoniano em G e S um subconjunto de V. O *comprimento* de C é a soma dos comprimentos l_{uv} para toda aresta uv em C. O circuito C' induzido por V em S, denotado por $\mathrm{sc}_S(C)$, é definido como o circuito que passa apenas pelos vértices de S e passa por eles na ordem induzida por C. (Informalmente, C' é obtido de C fazendo-se um "*short-cut*", como no algoritmo de Christofides (1976), visitando apenas os vértices de S.)

Um conjunto $\mathbf{p} = \{p_v : v \in V\}$ em que p_v é um número em $(0, 1]$ é chamado de *vetor de probabilidades* sobre V. Diz-se que um conjunto S de vértices de V *foi obtido aleatoriamente de acordo com* $\mathbf{p}$ se ele resultou do seguinte experimento aleatório: para cada vértice v, independentemente, inclua v em S com probabilidade p_v (e não inclua com probabilidade $1 - p_v$).

O *problema do United Parcel Service* (PUPS) consiste no seguinte: dado um conjunto V de vértices, uma função distância l no grafo completo G sobre V, satisfazendo a desigualdade triangular, e um vetor de probabilidades $\mathbf{p}$ sobre V, encontrar um circuito hamiltoniano C em G que minimize a esperança do comprimento de $\mathrm{sc}_S(C)$, onde S é um conjunto obtido aleatoriamente de acordo com $\mathbf{p}$.

Esse problema, proposto por Savelsbergh (2001), aparece no seguinte contexto. Motoristas de companhias de entrega visitam clientes diariamente para fazer entregas. Para a companhia, quanto menor a distância percorrida, melhor. Para o motorista, rotas que mudam drasticamente de um dia para o outro são inconvenientes: é mais fácil cortar caminho em uma rota fixa. O PUPS, cujo objetivo captura estes dois pontos de vista, é pelo menos tão difícil de aproximar quanto o TSP métrico. É trivial obter uma instância do TSP métrico a partir de uma instância do PUPS: basta ignorar o vetor de probabilidades. Considere uma instância $(G, l, \mathbf{p})$ do PUPS em que $p_v = 1$ para pelo menos um vértice. Mostramos que a razão entre o comprimento de uma solução ótima do TSP métrico em (G, l) e o comprimento de uma solução ótima do PUPS em $(G, l, \mathbf{p})$ é no máximo $1/p_{\min}$, onde $p_{\min} := \min_{v \in V} p_v$ [15]. Exibimos também um exemplo que mostra que esse resultado é justo. Como conseqüência, o algoritmo de Christofides para o TSP métrico é uma $3/(2p_{\min})$-aproximação para o PUPS [15].

6.2. Carteiro chinês e cobertura por circuitos em grafos mistos.

Os próximos dois problemas estão definidos para grafos mistos. Grafos mistos generalizam a noção de grafos orientados no sentido que podem conter tanto arestas quanto arcos. Um *grafo misto* é uma tripla $M = (V, E, A)$ onde V é o conjunto de vértices, E é o conjunto de arestas e A o conjunto de arcos de M. Quando E é vazio, diz-se que M é um grafo orientado e, quando A é vazio, diz-se que M é um grafo. (Note que apenas grafos mistos simples estão sendo considerados, ou seja, laços e arestas/arcos paralelos não são permitidos.)

Uma *cobertura por circuitos* de um grafo misto $M = (V, E, A)$ é uma família $\mathcal{C} = \{C_1, \ldots, C_k\}$ de circuitos de M tal que cada aresta ou arco de M pertence a pelo menos um circuito em $\mathcal{C}$. O *comprimento* de $\mathcal{C}$ é a soma do número de arestas e arcos dos circuitos em $\mathcal{C}$.

O *problema da cobertura mínima por circuitos* (PCMC) consiste no seguinte: dado um grafo misto M fortemente conexo sem pontes, encontrar uma cobertura por circuitos de M de comprimento mínimo. Lee e Wakabayashi (2002) mostraram que esse problema é NP-difícil se M é um grafo misto planar arbitrário. Ele é bem estudado quando M é um grafo: Thomassen (1997) mostrou que este caso é NP-difícil. Além disso, este caso está relacionado à bem conhecida conjectura da cobertura dupla por circuitos e ao problema do carteiro chinês.

Seja M um grafo misto fortemente conexo. Um *passeio de carteiro* em M é um passeio fechado que contém todas as arestas e arcos de M. Qualquer cobertura por circuitos de comprimento k pode ser convertida em um passeio de carteiro de comprimento k, mas o inverso não é verdade.

O *problema do carteiro chinês* (PCC) em grafos mistos é uma variante do PCMC: dado um grafo misto M fortemente conexo, encontrar um passeio de carteiro em M de comprimento mínimo.

Edmonds (1965) mostrou que o PCC pode ser resolvido em tempo polinomial em grafos e Edmonds e Johnson (1973) resolveram o problema para grafos orientados. Por outro lado, Papadimitriou (1976) provou que o PCC é NP-difícil em grafos mistos planares.

Estudamos a complexidade do PCMC e do PCC em grafos mistos com largura arbórea limitada por uma constante. (O conceito de largura arbórea usado aqui é a versão usual aplicada ao grafo não-orientado subjacente ao grafo misto em questão.) A nossa contribuição é um algoritmo polinomial para o PCMC e um para o PCC, ambos para grafos mistos com largura arbórea limitada [14].

6.3. **Caminhos mais curtos.** O bem-conhecido *problema do caminho mais curto* consiste no seguinte: dado um grafo orientado D, uma função não-negativa l nos arcos de D e dois vértices s e t, encontrar um caminho P de s a t que minimize $l(P)$, onde $l(P)$ denota a soma de $l(e)$ sobre todos os arcos e em P. Esse problema pode ser resolvido em tempo polinomial. O quarto problema é a seguinte generalização do problema do caminho mais curto, denotada por k-SPs: dados um grafo orientado $D = (V, A)$, funções não-negativas $l_1, \ldots, l_k$ definidas nos arcos de D e vértices s e t, encontrar k caminhos $P_1, \ldots, P_k$ de s a t internamente disjuntos nos vértices tais que $l_1(P_1) + \cdots + l_k(P_k)$ é o menor possível. O caso em que $l_1 = \ldots = l_k$ reduz-se ao problema do fluxo máximo de custo mínimo e portanto pode ser resolvido em tempo polinomial.

Primeiramente consideramos o k-SPs em grafos orientados acíclicos [17]. Algoritmos para encontrar caminhos disjuntos nos arcos em grafos orientados acíclicos têm aplicações em problemas de escalonamentos e em problemas de atribuição de aeronaves. Reformulamos o k-SPs em grafos orientados acíclicos em termos de encontrar um caminho mais curto em um grafo orientado acíclico auxiliar, que tem tamanho exponencial em k. Essa é uma conhecida reformulação que se deve a Perl e Shiloach (1978) para encontrar dois caminhos disjuntos nos vértices sob certas condições em grafos orientados acíclicos. Mais tarde, isso foi estendido por Fortune, Hopcroft e Wyllie (1980) no contexto de derivar um algoritmo polinomial para o problema de encontrar k caminhos disjuntos nos vértices sob certas condições em grafos orientados acíclicos (veja também Schrijver (1993)). Dessa reformulação, conseguimos, para todo k fixo, um algoritmo polinomial para o k-SPs restrito a grafos orientados acíclicos [17]. Também mostramos que o problema se torna muito mais difícil em grafos orientados arbitrários, mesmo para $k = 2$. Mais exatamente, mostramos que, para qualquer constante c, não existe n^c-aproximação para o 2-SPs a menos que P = NP, onde n é o número de vértices do grafo [17]. Por outro lado, mesmo em grafos orientados acíclicos, se k não estiver fixo, mostramos que o problema também é difícil: para qualquer constante c, não existe n^c-aproximação para o k-SPs restrito a grafos orientados acíclicos a menos que P = NP [17]. Esses dois resultados de inaproximabilidade mostram que se removermos qualquer uma das duas restrições sobre a entrada do algoritmo apresentado, o problema se torna intratável (a menos que P = NP), assim o resultado é num certo sentido o melhor possível.

Nesta mesma ocasião, consideramos ainda uma variante do problema para grafos (não-orientados), com múltiplos pares de terminais. Para esta variante, apresentamos um algoritmo polinomial para o caso em que todas as funções comprimento coincidem, o grafo dado é planar e todos os terminais encontram-se numa ordem adequada na mesma face de um desenho plano do grafo [17].

7. Problemas combinatórios relacionados a bases de Gröbner

Quando se escreve um polinômio em muitas indeterminadas, é costume fixar uma ordem nas variáveis e expressar cada monômio como um produto ordenado de potências das variáveis. Um tal monômio pode ser olhado como uma palavra no monóide livre nas mesmas indeterminadas. Isso determina uma função do monóide livre comutativo no monóide livre. Uma de nossas contribuições foi uma análise do comportamento de ideais sobre essa função [13]. Para explicar isso melhor, é preciso introduzir um pouco de notação.

Seja X um alfabeto finito (de *letras*). Denota-se por $[X]$ o monóide livre comutativo sobre X e por X^* o monóide livre sobre X. Os elementos de $[X]$ são chamados de *monômios* e os de X^* de *palavras*. Quando for conveniente, supõe-se que $X = \{x_1, x_2, \ldots, x_n\}$, de maneira que um monômio possa ser escrito em notação multiplicativa como $x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$. Denota-se por $\langle M \rangle$ o ideal gerado por um conjunto M, seja em $[X]$ ou em X^*. Assim, $\langle M \rangle = M[X]$ se $M \subseteq [X]$, e $\langle M \rangle = X^* M X^*$ se $M \subseteq X^*$. O epimorfismo canônico de monóides $X^* \to [X]$ é denotado por π.

Sabe-se que todo ideal num monóide livre comutativo é finitamente gerado (Lema de Dickson). O mesmo não vale para o monóide livre. Mostramos uma caracterização boa de quando $\pi^{-1}(\langle M \rangle)$ é finitamente gerado para um conjunto finito $M \subseteq [X]$ [13].

As próximas questões foram motivadas pelo estudo de apresentações não-comutativas de álgebras afins e suas bases de Gröbner. Elas podem ser apresentadas dentro do contexto de monóides como segue.

Para uma ordenação $\prec$ das letras, diz-se que uma palavra é *ordenada* se suas letras ocorrem nela em ordem crescente; se $x_1 \prec x_2 \prec \cdots \prec x_n$, uma tal palavra pode ser escrita de forma única como $x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$. Seja $\sigma_\prec : [X] \to X^*$ a função que leva cada monômio m à única palavra ordenada em $\pi^{-1}(m)$. Assim $\pi\sigma_\prec$ é a função identidade em $[X]$.

Considere ideais da forma $\mathcal{I}_\prec(M) = \langle \sigma_\prec(\langle M \rangle) \rangle$, com $M \subseteq [X]$, isto é, o ideal gerado por palavras ordenadas correspondendo a um ideal comutativo, especificado por seus geradores. O primeiro problema que consideramos, denotado aqui por P1, foi o seguinte: dado um conjunto finito $M \subseteq [X]$ e uma ordenação $\prec$ de X, decidir se $\mathcal{I}_\prec(M)$ é ou não finitamente gerado. O segundo, denotado por P2, busca uma ordenação: dado um conjunto finito $M \subseteq [X]$, decidir se existe uma ordenação $\prec$ de X tal que $\mathcal{I}_{\prec}(M)$ é finitamente gerado.

Para dar respostas algorítmicas a estas questões, é preciso definir como a entrada desses problemas é dada. Primeiramente, consideramos que os monômios de M são dados explicitamente, como vetores de expoentes. Neste caso, o tempo do algoritmo é medido em função da soma do número de bits nos expoentes. Para P1, exibimos uma caracterização que implica em um algoritmo polinomial que o resolve [13]. Esta caracterização implica ainda que o P2 está em NP. Mostramos também que, quando M é livre de quadrados, o P2 pode ser decidido em tempo polinomial, porém ele se torna NP-completo mesmo quando M consiste apenas de monômios quadráticos [13].

Consideramos ainda uma segunda maneira de se apresentar a entrada do problema. Olhando para o ideal $[X]$ como um subconjunto de $\mathbb{N}^n$, ele é caracterizado pela propriedade "$u \geq v \in I$ implica que $u \in I$", onde os vetores são comparados componente a componente. Conjuntos com tal propriedade podem ser apresentados como as soluções inteiras de um sistema de inequações lineares, $Av \geq b, Bv = 0$, onde A e B são matrizes inteiras, A é não-negativa, e b é um vetor inteiro; de fato, qualquer ideal de $[X]$ é uma união finita de conjuntos dessa forma. Esse tipo de descrição pode ser muito mais compacto que a lista de geradores de um ideal; ou seja, um conjunto minimal de tais geradores pode ter tamanho super-polinomial no tamanho total do sistema linear. Com a entrada apresentada nesse formato, mostramos que os dois problemas acima bem como o problema de decidir se $\pi^{-1}(\langle M \rangle)$ é finitamente gerado para um conjunto finito $M \subseteq [X]$ tornam-se difíceis: este último e o P1 são coNP-completos enquanto o P2 é NP-difícil [13].

8. Colorações de arestas em multigrafos

Nesta seção, os grafos podem ter arestas paralelas, mas não laços. Seja k um inteiro não-negativo. Um grafo G é *k-aresta-colorível* se existe uma função $\kappa : E(G) \to \{1, \ldots, k\}$, chamada de *$k$-aresta coloração*, tal que $\kappa(e) \neq \kappa(f)$ para quaisquer duas arestas distintas e e f de G que têm pelo menos um extremo em comum. O *índice cromático* $\chi'(G)$ de G é o menor k não-negativo tal que G é k-aresta-colorível. É claro que $\chi'(G) \geq \Delta(G)$, onde $\Delta(G)$ é o grau máximo em G, mas existe uma outra delimitação inferior. Seja

$$\Gamma(G) = \max\left\{ \frac{2|E(G[U])|}{|U| - 1} : U \subseteq V(G), |U| \geq 3 \text{ e } |U| \text{ é ímpar} \right\}.$$

Se U é como acima, então todo emparelhamento no subgrafo de G induzido por U tem tamanho no máximo $\lfloor \frac{1}{2}|U| \rfloor$. Conseqüentemente, $\chi'(G) \geq \Gamma(G)$. Se G é o grafo de Petersen, ou o grafo de Petersen com um vértice a menos, então $\chi'(G) > \max\{\Delta(G), \lceil \Gamma(G) \rceil\}$.

Seymour conjecturou que $\chi'(G) = \max\{\Delta(G), \lceil \Gamma(G) \rceil\}$ se G é um grafo planar. Tal conjectura provavelmente não tem uma prova fácil, pois implica o Teorema das Quatro Cores. No entanto, Seymour (1990) provou que sua conjectura vale para grafos série-paralelos. Esse resultado é fácil para grafos simples. A dificuldade está na presença de arestas paralelas. A prova de Seymour é elegante e interessante, porém o passo da indução requer a verificação de um grande número de desigualdades.

Nossa contribuição aqui é uma prova mais simples [16], baseada num lema estrutural sobre grafos série-paralelos, que por sua vez é uma conseqüência direta do bem-conhecido fato que todo grafo série-paralelo simples tem um vértice de grau no máximo dois. Nosso trabalho foi motivado pela conjectura de coloração de arestas com listas (*list edge-coloring*) (veja Bollobás e Harris (1985) e também o problema 12.20 no livro de Jensen e Toft (1995)). Estávamos tentando adquirir intuição sobre esta conjectura para grafos série-paralelos. Ela foi provada para grafos série-paralelos simples por Juvan, Mohar e Thomas (1999), mas está em aberto para grafos série-paralelos com arestas paralelas. Nossos esforços resultaram apenas numa prova mais simples do resultado de Seymour e em um algoritmo linear para decidir se um grafo série-paralelo pode ou não ser colorido com um certo número de cores [16].

9. Lista dos Trabalhos Anexos

Segue a lista de trabalhos que anexamos a este texto sistematizado. Nas referências abaixo, indicamos a seção em que os respectivos trabalhos são mencionados.

[1] J. Boyer, C.G. Fernandes, W.L. Hsu, A. Noma, and J.C. Pina, *Correcting and implementing the PC-tree planarity algorithm*, submetido, 2003. (*§3.4*)

[2] J. Boyer, C.G. Fernandes, A. Noma, and J.C. Pina, *Lempel, Even, and Cederbaum planarity method*, Proceedings of the III Workshop on Efficient and Experimental Algorithms, 2004, a aparecer, 14pp. (*§3.4*)

[3] G. Călinescu and C.G. Fernandes, *Finding large planar subgraphs and large subgraphs of a given genus*, Proc. 2nd Intern. Computing and Combinatorics Conference, vol. 1, 1996, pp. 152–161. (*§3.1*) (*§3.3*)

[4] G. Călinescu and C.G. Fernandes, *Multicuts in unweighted digraphs with bounded degree and bounded tree-width*, Electronic Notes in Discrete Mathematics **7** (2001), 4pp, Proceedings of the Brazilian Symposium on Graphs, Algorithms and Combinatorics (GRACO). (*§5.2*)

[5] G. Călinescu, C.G. Fernandes, U. Finkler, and H. Karloff, *A better approximation algorithm for finding planar subgraphs*, Proceedings of the Eleventh ACM-SIAM Symposium on Discrete Algorithms (SODA), 1996, pp. 16–25. (*§3.1*)

[6] G. Călinescu, C.G. Fernandes, U. Finkler, and H. Karloff, *A better approximation algorithm for finding planar subgraphs*, Journal of Algorithms **27** (1998), no. 2, 269–302. (*§3.1*)

[7] G. Călinescu, C.G. Fernandes, H. Karloff, and A. Zelikovsky, *A new approximation algorithm for finding heavy planar subgraphs*, Algorithmica **36** (2003), 179–205. (§*3.2*) (§*4*)

[8] G. Călinescu, C.G. Fernandes, I. Mandoiu, A. Olshevsky, K. Yang, and A. Zelikovsky, *Primal-dual algorithms for QoS multimedia multicast*, Proceedings of the GLOBECOM, 2003, 6pp. (§*4.2*)

[9] G. Călinescu, C.G. Fernandes, and B. Reed, *Multicuts in unweighted graphs with bounded degree and bounded tree-width*, Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization (IPCO) (R.E. Bixby, E.A. Boyd, and R.Z. Ríos-Mercado, eds.), Lecture Notes in Computer Science, vol. 1412, Springer, 1998, pp. 137–152. (§*5.2*)

[10] G. Călinescu, C. G. Fernandes, and B. Reed, *Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width*, Journal of Algorithms **48** (2003), no. 2, 333–359. (§*5.2*)

[11] P. Feofiloff, C.G. Fernandes, C.E. Ferreira, and J.C. Pina, *Approximation algorithms for the prize-collecting Steiner tree problem*, submetido, 2003. (§*4.1*)

[12] C.G. Fernandes, *A better approximation ratio for the minimum k-edge-connected spanning subgraph problem*, Journal of Algorithms **28** (1998), no. 1, 105–124. (§*5.1*)

[13] C.G. Fernandes, E. Green, and A. Mandel, *From monomials to words to graphs*, Journal of Combinatorial Theory, Series A **105** (2004), no. 2, 185–206. (§*7*)

[14] C.G. Fernandes, O. Lee, and Y. Wakabayashi, *The minimum cycle cover and the Chinese postman problems on mixed graphs with bounded tree width*, submetido, 2003. (§*6.2*)

[15] C.G. Fernandes and T. Nierhoff, *The UPS problem*, Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS) (Berlin), Lecture Notes in Computer Science, Springer, 2001, pp. 238–246. (§*6.1*)

[16] C.G. Fernandes and R. Thomas, *Coloring series-parallel multigraphs*, manuscrito, 2000. Disponível em `http://www.math.gatech.edu/~thomas/`. (§*8*)

[17] C.G. Fernandes, H. van der Holst, and J.C. Pina, *Multilength single pair shortest disjoint paths*, Tech. report, Universidade de São Paulo, 2004, RT–MAC 2004-02. (§*6.3*)

Referências

1. S. Arora, *Reductions, codes, PCPs, and inapproximability*, Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS), 1995, pp. 404–413.
2. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and intractability of approximation problems*, Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS), 1992, pp. 24–27.
3. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and approximation: Combinatorial optimization problems and their approximability properties*, Springer, 1999.
4. L. Auslander and S.V. Parter, *On imbedding graphs in the plane*, Journal of Mathematics and Mechanics **10** (1961), 517–523.
5. B. Bollobás and A.J. Harris, *List colorings of graphs*, Graphs and Combinatorics (1985), no. 1, 115–127.

6. K.S. Booth and G.S. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ–tree algorithms*, Journal of Computer and Systems Sciences **13** (1976), 335–379.
7. J.M. Boyer and W. Myrvold, *On the cutting edge: Simplified $O(n)$ planarity by edge addition*, Preprint, 29pp.
8. ______, *Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm*, Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (1999), 140–146.
9. M.H. Carvalho, M.R. Cerioli, R. Dahab, P. Feofiloff, C.G. Fernandes, C.E. Ferreira, K.S. Guimarães, F.K. Miyazawa, J.C. Pina, J. Soares, and Y. Wakabayashi, *Uma introdução sucinta a algoritmos de aproximação*, Publicações Matemáticas do IMPA, 2001.
10. M. Charikar, J. Naor, and B. Schieber, *Resource optimization in* QoS *multicast routing of real-time multimedia*, Proceedings of the 19th Annual IEEE INFOCOM, 2000, pp. 1518–1527.
11. J. Cheriyan, M. Kao, and R. Thurimella, *Algorithms for parallel k-vertex connectivity and sparse certificates*, SIAM Journal on Computing **22** (1993), no. 1, 157–174.
12. J. Cheriyan and R. Thurimella, *Approximating minimum-size k-connected spanning subgraphs via matching*, SIAM Journal on Computing **30** (2000), no. 2, 528–560 (electronic).
13. N. Chiba, T. Nishizeki, A. Abe, and T. Ozawa, *A linear algorithm for embedding planar graphs using PQ–trees*, Journal of Computer and Systems Sciences **30** (1985), 54–76.
14. N. Christofides, *Worst-case analysis of a new heuristic for the traveling salesman problem*, Technical Report 388, Carnegie Mellon University, 1976.
15. F. Chudak, T. Roughgarden, and D.P. Williamson, *Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation*, Proc. 8th Conference on Integer Programming and Combinatorial Optimization Conference (IPCO), Lecture Notes in Computer Science, vol. 2081, Springer, 2001, p. 60 ff.
16. R. Cimikowski, *An analysis of some heuristics for the maximum planar subgraph problem*, Proceedings of the Sixth ACM-SIAM Symposium on Discrete Algorithms (SODA), 1995, pp. 322–331.
17. R. Cole, R. Hariharan, M. Lewenstein, and E. Porat, *A faster implementation of the Goemans-Williamson clustering algorithm*, Symposium on Discrete Algorithms, 2001, pp. 17–25.
18. E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis, *The complexity of multiterminal cuts*, SIAM Journal on Computing **23** (1994), no. 4, 864–894.
19. C.M.H. de Figueiredo, L. Faria, and C.F.X. de Mendonça Neto, *Optimal node-degree bounds for the complexity of nonplanarity parameters*, Proceedings of the Thirteenth ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999, pp. 887–888.
20. ______, *On the complexity of the approximation of nonplanarity parameters for cubic graphs*, Discrete Applied Mathematics (2004).
21. N. Deo, *Note on Hopcroft and Tarjan planarity algorithm*, Journal of the Association for Computing Machinery **23** (1976), 74–75.
22. J. Edmonds, *Maximum matching and a polyhedron with* $0,1$*-vertices*, Journal of Research of the National Bureau of Standards B **69** (1965), 125–130.
23. J. Edmonds and E.L. Johnson, *Matching, Euler tours and the Chinese postman problem*, Mathematical Programming **5** (1973), 88–124.
24. P. Erdős, *Gráfok páros körüljárású részgráfjairól* (On bipartite subgraphs of graphs, em húngaro), Matematikai Lapok **18** (1967), 283–288.
25. S. Even and R.E. Tarjan, *Computing an st-numbering*, Theoretical Computer Science **2** (1976), 339–344.

26. U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, Journal of the ACM **45** (1998), no. 4, 634–652.
27. C.G. Fernandes, *A better approximation ratio for the minimum k-edge-connected spanning subgraph problem*, Proceedings of the Eleventh ACM-SIAM Symposium on Discrete Algorithms (SODA), 1997, pp. 629-638.
28. L.R. Ford and D.R. Fulkerson, *Maximal flow through a network*, Canadian Journal of Mathematics **8** (1956), 399–404.
29. S. Fortune, J. Hopcroft, and J. Wyllie, *The direct subgraph homeomorphism problem*, Theoretical Computer Science **10** (1980), 111–121.
30. H.N. Gabow, M.X. Goemans, and D.P. Williamson, *An efficient approximation algorithm for the survivable network design problem*, Proceedings of the 3rd Conference on Integer Programming and Combinatorial Optimization (IPCO) (G. Rinaldi and L.A. Wolsey, eds.), CIACO, 1993, pp. 57–74.
31. M.R. Garey, R.L. Graham, and J.D. Ullman, *Worst-case analysis of memory allocation algorithms*, Proceedings of the 4th Annual ACM Symposium on the Theory of Computing (STOC), 1972, pp. 143–150.
32. M.R. Garey and D.S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, 1979.
33. N. Garg, *A 3-approximation for the minimum tree spanning k vertices*, Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS), 1996, pp. 320–309.
34. N. Garg, G. Konjevod, and R. Ravi, *A polylogarithmic approximation algorithm for the group Steiner tree problem*, Journal of Algorithms **37** (2000), no. 1, 66–84.
35. N. Garg, V.V. Vazirani, and M. Yannakakis, *Approximate max-flow and min-(multi)cut theorems and their applications*, SIAM Journal on Computing **25** (1996), 235–251.
36. ———, *Primal-dual approximation algorithms for integral flow and multicut in trees*, Algorithmica **18** (1997), 3–20.
37. M.X. Goemans and D.P. Williamson, *A general approximation technique for constrained forest problems*, SIAM Journal on Computing **24** (1995), no. 2, 296–317.
38. A.J. Goldstein, *An efficient and constructive algorithm for testing whether a graph can be embedded in a plane*, Graph and Combinatorics Conf., Contract No. NONR 1858-(21), Office of Naval Research Logistics Proj., Dep. of Math., Princeton U., 2 pp., 1963.
39. R.L. Graham, *Bounds for certain multiprocessor anomalies*, Bell System Technical Journal **45** (1966), 1563–1581.
40. C. Gröpl, S. Hougardy, T. Nierhoff, and H.J. Prömel, *Approximation algorithms for the Steiner tree problem in graphs*, Steiner trees in industry, Comb. Optim., vol. 11, Kluwer Acad. Publ., Dordrecht, 2001, pp. 235–279.
41. D.S. Hochbaum (ed.), *Approximation algorithms for NP-hard problems*, PWS Publishing, 1997.
42. J. Hopcroft and R. Tarjan, *Efficient planarity testing*, Journal of the Association for Computing Machinery **21** (1974), no. 4, 549–568.
43. W.L. Hsu, *An efficient implementation of the PC-tree algorithm of Shih & Hsu's planarity test*, Tech. report, Inst. of Information Science, Academia Sinica, 2003.
44. T.C. Hu, *Multicommodity network flows*, Operations Research **9** (1963), 898–900.
45. A. Itai, *Two-commodity flow*, Journal of the ACM **25** (1978), no. 4, 596–611.
46. T.R. Jensen and B. Toft, *Graph coloring problems*, Wiley, New York, 1995.
47. D.S. Johnson, *Approximation algorithms for combinatorial problems*, Journal of Computer and System Sciences **9** (1974), 256–278.
48. D.S. Johnson, M. Minkoff, and S. Phillips, *The prize collecting Steiner tree problem: theory and practice*, Symposium on Discrete Algorithms, 2000, pp. 760–769.

49. M. Jünger, S. Leipert, and P. Mutzel, *Pitfalls of using PQ-trees in automatic graph drawing*, Proc. 5th International Symposium on Graph Drawing '97 (G. Di Battista, ed.), Lecture Notes in Computer Science, vol. 1353, Springer Verlag, Sept. 1997, pp. 193–204.
50. M. Juvan, B. Mohar, and R. Thomas, *List edge-colorings of series-parallel graphs*, Electronic Journal of Combinatorics **6** (1999), no. 1, Research Paper 42.
51. D.R. Karger, *Random sampling in cut, flow, and network design problems*, Math. Oper. Res. **24** (1999), no. 2, 383–413.
52. M. Karpinski, I. Mandoiu, A. Olshevsky, and A. Zelikovsky, *Improved approximation algorithms for some generalization of the Steiner tree problem*, Proc. Int. Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, vol. 2748, Springer, 2003, pp. 401–411.
53. M. Karpinski and A. Zelikovsky, *New approximation algorithms for the Steiner tree problems*, Journal of Combinatorial Optimization **1** (1997), no. 1, 47–65.
54. S. Khuller and B. Raghavachari, *Improved approximation algorithms for uniform connectivity problems*, Journal of Algorithms **21** (1996), no. 2, 434–450.
55. S. Khuller and U. Vishkin, *Biconnectivity approximations and graph carvings*, Journal of the ACM **41** (1994), no. 2, 214–235.
56. P. Klein, *A data structure for bicategories, with application to speeding up an approximation algorithm*, Information Processing Letters **52** (1994), no. 6, 303–307.
57. P.N. Klein, S.A. Plotkin, S. Rao, and E. Tardos, *Approximation algorithms for Steiner and directed multicuts*, Journal of Algorithms **22** (1997), no. 2, 241–269.
58. O. Lee and Y. Wakabayashi, *On the circuit cover problem for mixed graphs*, Combinatorics, Probability and Computing **11** (2002), 43–59.
59. A. Lempel, S. Even, and I. Cederbaum, *An algorithm for planarity testing of graphs*, Theory of Graphs (New York) (P. Rosenstiehl, ed.), Gordon and Breach, 1967, pp. 215–232.
60. W. Mader, *Minimale n-fach zusammenhängende Graphen mit maximaler Kantenzahl*, J. Reine Angew. Math. **249** (1971), 201–207.
61. ———, *Ecken vom Grad n in minimalen n-fach zusammenhängenden Graphen*, Arch. Math. (Basel) **23** (1972), 219–224.
62. N. Maxemchuk, *Video distribution on multicast networks*, IEEE Journal on Selected Issues in Communications **15** (1997), 357–372.
63. E. Mayr, H.J. Prömel, and A. Steger (eds.), *Lectures on proof verification and approximation algorithms*, Lecture Notes in Computer Science, vol. 1367, Springer, 1998.
64. A. Noma, *Análise experimental de algoritmos de planaridade*, Master's thesis, Universidade de São Paulo, 2003, `http://www.ime.usp.br/dcc/posgrad/teses/noma/dissertation.ps.gz`.
65. C.H. Papadimitriou, *On the complexity of edge traversing*, Journal of ACM **23** (1976), 544–554.
66. Y. Perl and Y. Shiloach, *Finding two disjoint paths between two pairs of vertices in a graph*, Journal of the Association for Computing Machinery **25** (1978), 1–9.
67. E.M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial algorithms: Theory and practice*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.
68. G. Robins and A. Zelikovsky, *Improved Steiner tree approximation in graphs*, Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 2000) (New York), ACM, 2000, pp. 770–779.
69. M.W.P. Savelsbergh, comunicação pessoal, 2001.
70. A. Schrijver, *A group-theoretical approach to disjoint paths in directed graphs*, CWI Quarterly **6** (1993), 257–266.
71. P.D. Seymour, *Colouring series-parallel graphs*, Combinatorica **10** (1990), no. 4, 379–392.

72. W.K. Shih and W.L. Hsu, *A simple test for planar graphs*, Proceedings of the International Workshop on Discrete Math. and Algorithms, University of Hong Kong, 1993, pp. 110–122.
73. ———, *A new planarity test*, Theoretical Computer Science **223** (1999), 179–191.
74. H. Takahashi and A. Matsuyama, *An approximate solution for the Steiner problem in graphs*, Mathematica Japonica **24** (1979/80), no. 6, 573–577.
75. E. Tardos and V.V. Vazirani, *Improved bounds for the max-flow min-multicut ratio for planar and $K_{r,r}$-free graphs*, Inform. Process. Lett. **47** (1993), no. 2, 77–80.
76. R. Thomas, *Planarity in linear time*, `http://www.math.gatech.edu/~thomas/`, 1997.
77. C. Thomassen, *On the complexity of finding a minimum cycle cover of a graph*, SIAM Journal on Computing **26** (1997), 675–677.
78. V.V. Vazirani, *Approximation algorithms*, Springer, 2001.
79. S.G. Williamson, *Embedding graphs in the plane - algorithmic aspects*, Ann. Disc. Math. **6** (1980), 349–384.
80. ———, *Combinatorics for computer science*, Computer Science Press, Maryland, 1985.
81. A.Z. Zelikovsky, *An 11/6-approximation algorithm for the network Steiner problem*, Algorithmica **9** (1993), no. 5, 463–470.
82. ———, *Better approximation bounds for the network and Euclidean Steiner tree problems*, Tech. Report CS-96-06, University of Virginia, 1996.

Trabalhos Anexos

Correcting and Implementing the PC-tree Planarity Algorithm

John M. Boyer
PureEdge Solutions Inc.
jboyer@PureEdge.com; jboyer@acm.org

Cristina G. Fernandes[*], Alexandre Noma[†], José Coelho de Pina Jr.[*]
University of São Paulo, Brazil
{cris,noma,coelho}@ime.usp.br

Wen-Lian Hsu
Institute of Information Science, Academia Sinica
Taipei, Taiwan, R. O. C.
hsu@iis.sinica.edu.tw

August 2003

Abstract

A graph is *planar* if it can be drawn on the plane with vertices at unique locations and no edge intersections except at the vertex endpoints. Recent research efforts have produced new algorithms for solving planarity-related problems. Shih and Hsu proposed a linear-time algorithm based on a data structure they named PC-tree, which is similar to but much simpler than a PQ-tree. The paper does not explain in detail how to implement the routines that manipulate a PC-tree, and there are some nontrivial correctness and run-time issues that were not addressed. So it is far from trivial to derive a proper linear-time implementation from their description. This paper presents additions to the theoretical framework of the PC-tree algorithm that are necessary to achieve correctness and linear running time. A linear-time implementation that addresses the issues raised in this paper was developed in the LEDA platform and is available.

[*] Research partially supported by PRONEX/CNPQ 664107/1997-4 (Brazil).
[†] Supported by FAPESP, a Brazilian funding agency.

1 Introduction

The first linear-time planarity testing algorithm is due to Hopcroft and Tarjan [9]. The method first embeds a cycle of the graph, then it breaks the remainder of the graph into a sequence of paths that can be added either to the inside or outside of the starting cycle. Some corrections appear in [7], and significant additional details are presented by Williamson [21, 23] as well as the text by Reingold, Nievergelt and Deo [18].

The second method of planarity testing proven to achieve linear time began with a quadratic algorithm due to Lempel, Even, and Cederbaum [16] (the LEC algorithm). The algorithm begins by creating an st-numbering for a biconnected input graph. One property of an st-numbering is that there is a path of higher numbered vertices leading from every vertex to the vertex t, which has the highest number. Thus, if the input graph is planar, there must exist an embedding $\tilde{G}_k$ of the first k vertices such that the remaining vertices ($k+1$ to t) can be embedded in a single face of $\tilde{G}_k$. This planarity testing algorithm was optimized to linear time by a pair of contributions. Even and Tarjan [8] optimized st-numbering to linear time, while Booth and Lueker [1] developed the PQ-tree data structure, which allows the planarity test to efficiently maintain information about the portions of the graph that can be permuted or flipped before and after embedding each vertex. Chiba, Nishizeki, Abe, and Ozawa [5] augmented the PQ-tree operations so that a planar embedding is computed as the operations are performed, all in linear time.

These algorithms are widely regarded as being quite complex [5, 14, 19]. Recent research efforts have resulted in two simpler linear-time algorithms, proposed independently, one by Boyer and Myrvold [3, 2] and the other by Shih and Hsu [19]. Both algorithms present a number of similar and very interesting ideas. One of the common ideas consists of processing the vertices in a post-order traversal of the depth first search (DFS) tree of the graph, or simply the reversal of the DFS number order (instead of an st-numbering). This has the property that there is a path of unprocessed vertices from every vertex to the root of the DFS-tree. While processing vertex v, the edges from v to the already processed vertices are embedded (if possible).

The Boyer-Myrvold method uses a graph data structure to maintain the collection of planar biconnected components that are formed as edges are added. The cut vertices separating the biconnected components are represented by 'virtual' vertices. For each vertex v in reverse of the depth first search order, a preliminary bottom-up method is performed to identify the 'active' portion of the DFS subtree rooted at v based on which of its subtrees contain a proper descendant that, in the input graph, is adjacent to v by a back edge. Then, a method called 'Walkdown' traverses the active DFS subtree in a top-down fashion, embedding back edges from v to its descendants and merging biconnected components as necessary while preserving planarity. The Walkdown traversal method obeys a few simple rules that guarantee that it will be able to embed all edges from v to its descendants except when a $K_{3,3}$ or K_5 minor can be identified.

The method of Shih and Hsu [19] also processes the vertices of the input graph from descendants to ancestors, and it also adds the back edges from i to its descendants unless a nonplanarity conditions is detected. To effect this strategy, Shih and Hsu created a data structure called a PC-tree, which is a simplified form of the Booth-Lueker PQ-tree. For each vertex i, the algorithm first searches for a number of defined nonplanarity patterns in the PC-tree, and if none are found, then a planarity reduction is applied to embed the edges from i to its descendants. However, Shih and Hsu's formulation lacks a description of how exactly to implement the routines that manipulate the PC-tree to solve the planarity problem. This description is essential for one to derive a linear-time implementation of their algorithm, and there is a series of nontrivial details involved. Moreover, there are some flaws in the proof that the nonplanarity patterns and the planarity reduction patterns together form an unavoidable set. Some of the problems were first reported in Boyer's dissertation [2]. The problems we solve are mainly additional non-planarity cases that occur when a PC-tree contains C-nodes, and the more recent publications have not solved these problems, but instead have focused on equating PQ-tree reductions on planar graphs with PC-tree reductions [10] and the application of PC-trees to the consecutive ones problem [11, 13]. A book chapter under development by McConnell and Hsu [12] presents an alternate proof that accounts for the cases we independently discovered. The proof often uses new definitions, graph theoretic arguments and an unrooted view of the PC-tree in lieu of PC-tree constructs. Our presentation states the results in terms of specific non-planarity conditions that arise within a PC-tree as formulated in [19].

Section 2 first presents an overview of the PC-tree data structure and algorithm as presented in [19]. Then, Section 3 presents corrections to the SH algorithm, and Section 4 presents a proof that the corrected SH algorithm does indeed distinguish between planar and nonplanar graphs. As for performance, Section 5 describes two issues that arise when one tries to create a linear-time implementation of Shih and Hsu's ideas. The solutions for these two issues were inspired by Boyer and Myrvold [3]. A linear-time implementation that accounts for the correctness and speed issues described in this paper can be found at `http://www.ime.usp.br/~coelho/sh`. Section 6 presents an empirical comparison of this implementation to linear-time implementations of other well-known planarity algorithms.

2 Overview of Planarity by PC-trees

The Shih-Hsu algorithm begins by embedding the depth first search tree (a trivial task). The main processing model is therefore concerned with embedding the back edges for each vertex. The vertices are processed in a post-order traversal of the depth first search tree. For a vertex i, the back edges from i to its descendants are added. The back edges from i to its ancestors are embedded when those ancestors are processed.

If a graph G is planar, then it is always possible to produce a planar embedding $\tilde{G}_i$ of the subgraph induced by the subtree rooted by i such that all descendants of i with back edge connections to ancestors of i are on the boundary of a single face of $\tilde{G}_i$. The rationale is the same as that given above for the LEC algorithm. Hence, when the SH algorithm is processing a planar graph, it creates successively larger partial embeddings of the form $\tilde{G}_0$, $\tilde{G}_1$, $\tilde{G}_2$, ..., $\tilde{G}_i$, ..., $\tilde{G}_n$, where the last result is an embedding of G. Naturally, the SH algorithm must also account for nonplanar graphs. Nonetheless, the processing model for vertex i remains quite simple: **search the partial embedding $\tilde{G}_{i-1}$ for nonplanarity conditions established by several lemmata, and if none are found, then apply a planarity reduction to produce $\tilde{G}_i$.**

The SH algorithm represents the partial embedding with a data structure called a PC-tree. The starting PC-tree represents $\tilde{G}_0$, which is the depth first search tree only, with no back edges. Each node of the tree is a P-node that represents a single vertex of the input graph G. The PC-tree remains a tree at all times even though it conceptually represents a subgraph that contains cycles as back edges from G are embedded. As back edges are added, they biconnect portions of the embedding that were previously separable. The separable components are represented by multiple nodes of the PC-tree, and these are consolidated into a single C-node representing the new biconnected component.

In general, the P-nodes of a PC-tree represent cut vertices in the partial embedding, and the C-nodes represent biconnected components. Before the back edges from a vertex i to its descendants can be embedded, the partial embedding $\tilde{G}_{i-1}$ must be rearranged so that vertices with back edge connections to proper ancestors of i are in a single face. This rearrangement must follow certain rules. Specifically, the children of a P-node can be arbitrarily permuted, and the children of a C-node can only be flipped (reversed). The nonplanarity conditions detect when the required rearrangement is not possible. If the rearrangement is possible, then the planarity reductions perform the rearrangement, and they consolidate portions of the PC-tree into single C-nodes as necessary to effect the embedding of the new back edges and produce $\tilde{G}_i$.

Each C-node in a PC-tree has only P-node neighbors that represent vertices along the external face bounding cycle of the biconnected component represented by the C-node. For this reason, the P-node neighbors of a C-node are called its *representative bounding cycle* (RBC). Given a C-node c, the neighbor that is closer to i than c is the parent of c, and the other neighbors of c are its children. However, the children of a C-node cannot indicate the C-node as the parent (see Section 5.2), so in order to traverse from a child w to the parent p of c (or vice versa), one of two paths around the RBC is taken.

In general, T_v denotes the PC-subtree rooted by v, which represents the partial embedding of a subgraph of G induced by the vertices of the DFS subtree rooted by vertex v. For each DFS child r of i, the algorithm considers separately the embedding of back edges between i and vertices in T_r. This is permissible since, given a subgraph H containing the DFS tree of G plus all back edges between vertices in T_i, vertex i still separates any two of its DFS children r_1 and r_2 in H. Therefore, a Kuratowski subgraph cannot span the subgraphs induced by T_{r_1} and T_{r_2} because there are not enough paths connecting them.

Within T_r, a subtree T_s is an *i^*-subtree* if it has unembedded back edge connections only to proper DFS ancestors of i. Similarly, an *i-subtree* is a subtree T_s of T_r that has unembedded back edges only to i. The absence of the nonplanarity patterns is supposed to guarantee that one of the planarity reductions is applicable. The planarity reductions have the property that the children of P-nodes can be permuted and the children of C-nodes can be flipped (reversed) such that all i-subtrees are near i and all i^*-subtrees can be avoided while visiting the i-subtrees to embed the back edges to i. See Figure 1.

A *terminal node* is a node t in the PC-tree with the following properties: 1) t has a child i-subtree or is adjacent to i by a back edge; 2) t has a child i^*-subtree or is adjacent to a proper ancestor of i by a back edge; 3) t has no proper descendants in the PC-tree with the same properties. Terminal nodes are so named because they are the endpoints

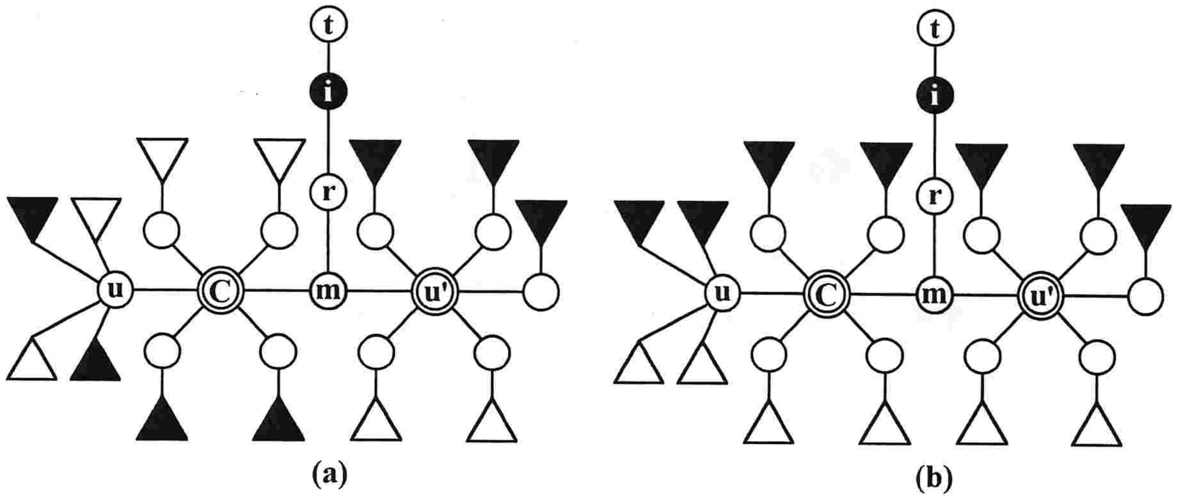


Figure 1: Permute P-nodes and flip C-nodes to visit i-subtrees and to avoid i^* subtrees. a) Darkened triangles are i-subtrees, whitened triangles are i^*-subtrees, double circles are C-nodes and single circles are P-nodes. b) The children of u are permuted, node c is flipped on the path axis (u, ..., m), and u' is not changed since it has the desired configuration.

of *critical paths* to r that must be searched for nonplanarity conditions. As an example, Figure 2 illustrates a nonplanarity condition that can arise if there are three or more terminal nodes. Then, Lemma 1 (Lemma 2.5 in [19]) describes a necessary condition for planarity, the absence of which yields the $K_{3,3}$ minor appearing in Figure 3.

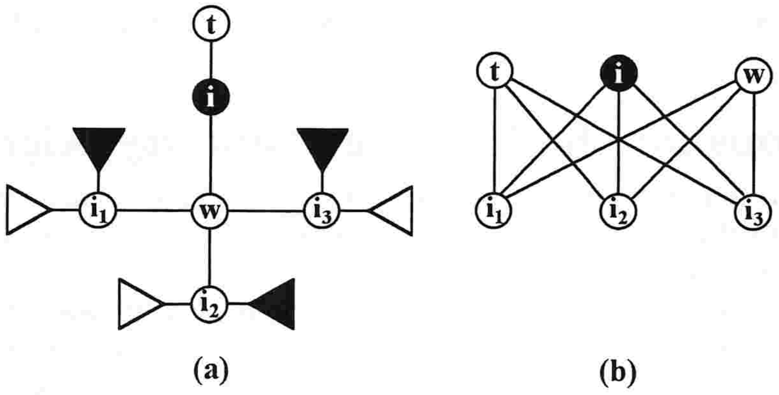


Figure 2: (a) A PC-tree with three terminal nodes. (b) The corresponding $K_{3,3}$ Minor. Note that there are many possible variations in connections of the critical paths and the i^*-subtree connections to proper ancestors of i, but edge contraction is used to eliminate unnecessary complexities.

Lemma 1 (Shih and Hsu) *Suppose there are two terminal nodes u and u' in T_r. Let P be the unique path in T from u to u'. Let m be the least common ancestor of u and u'. Let P' be the unique path from m to r. Let $S = \{v||v$ is a child of a node in P, but v is not in $P\}$. Let $S' = \{v||v$ is a child of a node in P' - $\{m\}$, but v is not in $P'\}$ (note that when m=r, S' is empty). Then, for each node v in S, T_v is either an i-subtree or an i^*-subtree, and for each node v in S', T_v is an i-subtree.*

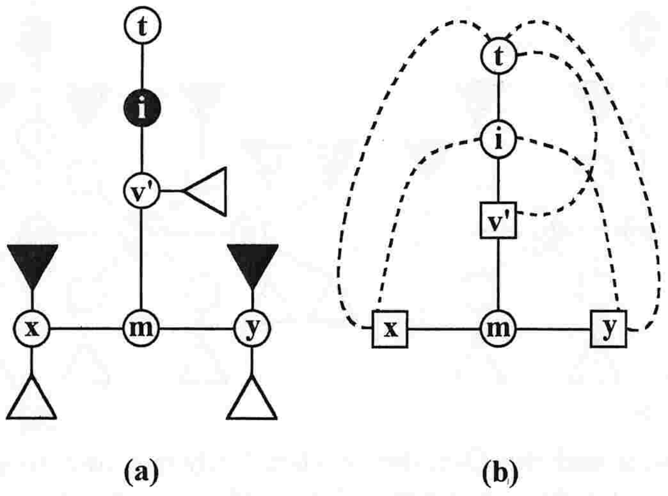


Figure 3: (a) An i^*-subtree attached to a proper ancestor v' of m, the closest common ancestor of two terminal nodes u and u'. (b) The resulting $K_{3,3}$ from [19]

Proof. Since nonessential nodes are removed, if v in S or S' is not the root of an i-subtree or i^*-subtree, then T_v must contain another terminal node, contradicting the assumption of two terminal nodes. That $v \in S'$ cannot be the root of an i^*-subtree is proven by Figure 3, which depicts the resulting $K_{3,3}$ minor. $\square$

Remark: The proof of Lemma 1 (Lemma 2.5 in [19]) is specific to PC-trees that contain only P-nodes. Section 3 discusses difficulties with its extension to general PC-trees that contain C-nodes.

3 Corrections for the PC-tree Planarity Algorithm

The PC-tree method in [19] requires some fixes to yield a correct planarity test. Aside from the three terminal node case, Shih and Hsu present four necessary conditions for maintaining planarity: "In Lemma 2.5, Corollary 2.6, [and] Lemmas 3.1 and 3.2 we made the assumption that graph G is planar in deriving at those conclusions. We shall show that if these conclusions hold at each iteration by showing that these conditions imply a feasible internal embedding for each 2-connected component." [19, p. 188]. The authors then proceed to demonstrate how to perform planarity reductions for the one and two terminal node cases, but the proof does not show that the presence of the four necessary planarity conditions yields only PC-trees that are reducible by the methods shown.

3.1 Patterns of child i-subtrees and i^*-subtrees around a terminal C-node

Perhaps the most critical problem for PC-tree planarity correctness pertains to Lemma 3.2 in [19]. The lemma seeks to characterize the allowable pattern of child i-subtrees and i^*-subtrees around a terminal C-node. Put simply, the lemma states that for the root j of any child i-subtree of a terminal C-node, one of the two RBC paths from j to the parent of the C-node must contain only i-subtrees.

While the lemma statement is certainly necessary to maintaining planarity, it is only sufficient in the one terminal node case when the terminal node has no proper ancestor with a child i^*-subtree. In the two terminal node case and the one terminal node case where the terminal node has a proper ancestor with a child i^*-subtree, it is possible to be compliant with the statement of the lemma yet still have a nonplanarity condition. Lemma 2 characterizes the additional restriction required on terminal C-nodes. Figure 4 depicts example PC-trees for the additional restriction, along with the resulting $K_{3,3}$ minor.

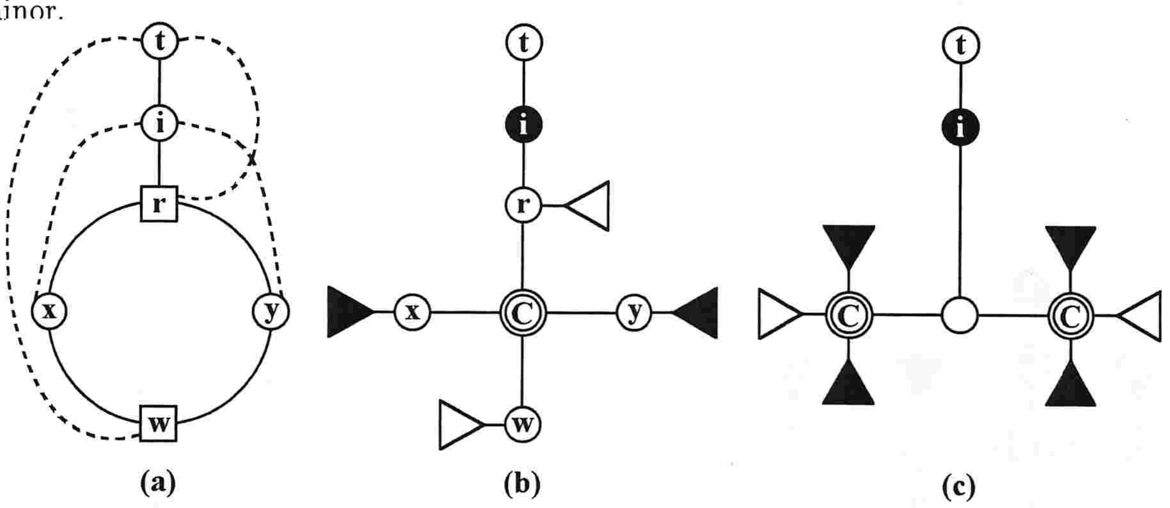


Figure 4: (a) A $K_{3,3}$ nonplanarity minor from [3], (b) A corresponding PC-tree with one terminal C-node having the forbidden child i-i^* subtree pattern, (c) Another example with two terminal C-nodes that have the forbidden child i-i^* subtree pattern. Only one of the terminal nodes must be a C-node with the forbidden subtree pattern.

Lemma 2 *If a terminal C-node c has a proper ancestor a with either a direct back edge to a proper ancestor of i or a child v not an ancestor of c such that T_v contains an i^*-subtree, then c must have a child w for which an RBC path from w to the parent p of c contains all child i-subtrees of c.*

Proof. The children of the terminal C-node in Figure 4(b) depict the minimal configuration of forbidden subtrees to which all forbidden subtree patterns can be reduced. The result is the $K_{3,3}$ minor in Figure 4(a). Figure 4(c) shows that with two terminal nodes, a terminal C-node also must not have the forbidden subtree configuration because the subtree containing the other terminal node attached to the least common ancestor m is analogous to a child i^*-subtree, so again the $K_{3,3}$ minor in Figure 4(a) results. □

3.2 Patterns of child i-subtrees and i^*-subtrees around an intermediate C-node

Lemma 3.1 of [19] places a necessary condition on the intermediate C-nodes of the path P between two terminal nodes. Given an intermediate C-node c with neighbors v and v' in P, one of the two RBC paths strictly between v and v' must contain only i-subtrees and the opposing RBC path strictly between v and v' must contain only child i^*-subtrees.

There are three problems with this lemma in [19]. First of all, as a proof by contradiction, the proof must account for the negation of the condition in the theorem. The proof in [19] presents the case of having both a child i-subtree and i^*-subtree along a single RBC path. However, it is possible to avoid this case yet still have a nonplanarity condition according to the lemma statement if both RBC paths contain only a child i^*-subtree. Secondly, the stated condition is not quite strong enough if the intermediate C-node is m, the closest common ancestor of the terminal nodes. Lemma 3 provides the required modifications to the statement and proof of Lemma 3.1 in [19]. The third problem is simply that Lemma 3.1 of [19] applies only to the two terminal node case, but Corollary 4 demonstrates the need to extend the necessary condition of the lemma to the analogous scenario in the one terminal node case.

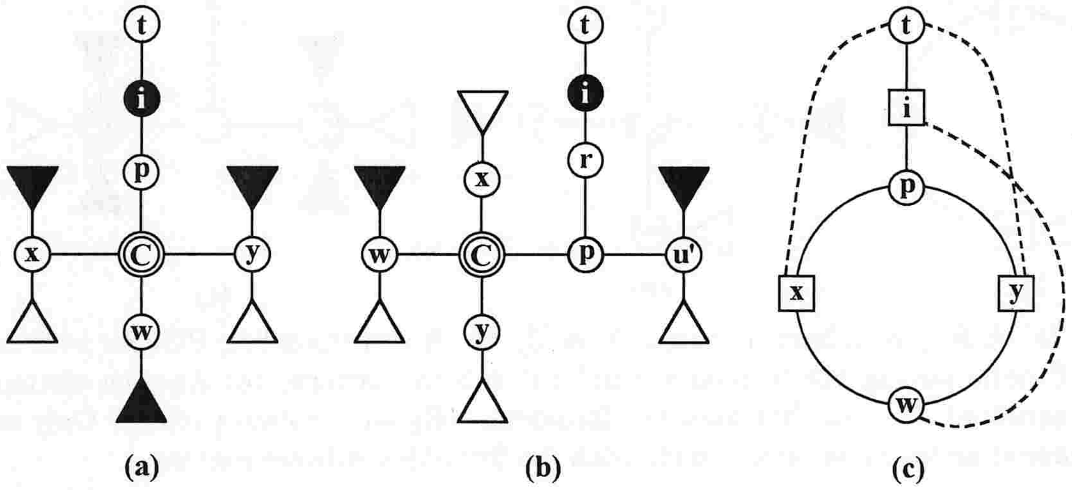


Figure 5: (a) A PC-tree in which m is a C-node with a child i-subtree below path P between terminal nodes x and y. (b) A PC-tree with an intermediate C-node that has child i^*-subtrees along both RBC paths from parent p to the next node w in path P. (c) The corresponding $K_{3,3}$ minor from [3]. Note: This minor also appears in the new three terminal node case of Figure 7 as well as the forbidden child i-i^* subtree pattern of Lemma 3.2 in [19].

Lemma 3 *Given the PC-tree path P between two terminal nodes u and u' in T_r, consider an intermediate C-node c in $P - \{u, u'\}$ with neighbors v and v' in P. Let m denote the closest common ancestor of u and u' in the PC-tree. Then, the children of c along one RBC path of c strictly between v and v' must be only child i-subtrees, and the opposing RBC path strictly between v and v' must contain only child i^*-subtrees. Further, if $c = m$, then the RBC path containing the child i-subtrees must also contain the parent p of c.*

Proof. When $c \neq m$, the proof of Lemma 3.1 in [19] demonstrates the nonplanarity condition that results if one RBC path contains both a child i-subtree and i^*-subtree. The nonplanarity condition also covers the case in which $c = m$ and both a child i^*-subtree and i-subtree appear in the RBC path that excludes the parent p of c. The remaining points below were omitted from the proof of Lemma 3.1 in [19].

When $c = m$, the RBC path strictly between v and v' that excludes the parent p can still generate a nonplanarity condition even if it contains no child i^*-subtree as required by the partial proof of Lemma 3.1. If that RBC path contains a child i-subtree, then the PC-tree has the form depicted in Figure 5(a), which results in the $K_{3,3}$ depicted in Figure 5(c).

When $c = m$, the RBC path strictly between v and v' that includes the parent p of c cannot contain a child i^*-subtree. Given the root j of such a child j i^*-subtree, one of the nonplanarity minors depicted for Lemma 7 can be obtained by edge contracting the RBC path to merge j with the parent p of c.

When $c \neq m$, then both RBC paths around c strictly between v and v' cannot contain a child i^*-subtree. If both RBC paths contain child i^*-subtrees as shown in Figure 5(b), then a $K_{3,3}$ can be found according to the nonplanarity minor in Figure 5(c). □

Remark: The nonplanarity condition of Figure 5(a) is similar to the one in Lemma 3.2 of [19], which requires the C-node to be a terminal node and x and y to be child i^*-subtrees that obstruct both RBC paths from w to the parent of the C-node.

Corollary 4 *Given one terminal node u, let P denote the path from u to the farthest ancestor u' with a child i^*-subtree. Let c be an intermediate C-node in path $P - \{u\}$. For $c \neq u'$, let v and v' denote the neighbors of c in P. For $c = u'$, let v denote the neighbor of c in P and let v' denote the closest child i^*-subtree along either RBC path from the parent p of c. The following conditions must hold:*

- *The children of c in one RBC path strictly between v and v' must contain only child i-subtrees.*
- *The opposing RBC path strictly between v and v' must contain only child i^*-subtrees.*
- *If $c = u'$, then the RBC path containing the child i-subtrees must also contain p.*

3.3 Direct back edges as degenerate i-subtrees and i^*-subtrees

There is an omission from the presentation of several results in [19], including Theorem 2.4, Lemma 2.5 and Corollary 2.6 of [19]. Lemma 5 demonstrates that the nonplanarity condition for Lemma 2.5 in [19] (Lemma 1 above) can still occur despite the absence of the condition stated by the lemma. Since a number of other results in [19] have the same problem, Corollary 6 makes a statement that fixes the underlying problem.

Lemma 5 *Given the same assumptions as Lemma 1, nonplanarity can result if S' is empty or devoid of vertices that root child i^*-subtrees.*

Proof. A node in $P' - \{m\}$ can have a direct back edge to a proper ancestor of i. □

Corollary 6 *A back edge (v, i) can be considered equivalent to a child i-subtree of v, and a back edge (v, t) where t is a proper ancestor of i can be considered equivalent to a child i^*-subtree of v.*

Proof. Solely for the purpose of simplifying proof statements, such direct back edges can be considered to be subdivided by an implicit degree two vertex w, which would be an implicit child of v. □

3.4 Additional cases of surrounding an i^*-subtree

Consider the extension of Lemma 2.5 in [19] (presented in Lemma 1 above) to the case of a PC-tree that contains C-nodes. Specifically, suppose that the closest common ancestor m of the two terminal nodes is in fact a C-node whose parent has the only child i^*-subtree along the path P'. Figure 6 depicts an example PC-tree and the corresponding K_5 minor pattern from [3]. This case is of critical importance because some graphs that it represents do not even contain a $K_{3,3}$, which demonstrates that the proof of Lemma 2.5 does not "go through for the case of general trees without any changes provided that the paths through a C-node are interpreted correctly" [19, p. 185]. Lemma 7 properly extends Lemma 2.5 of [19], including a proof that no other nonplanarity patterns result from its necessary condition.

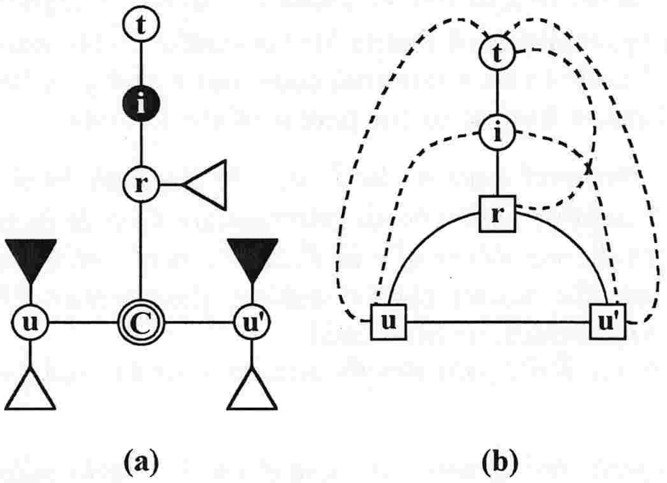


Figure 6: (a) A PC-tree in which the closest common ancestor of terminal nodes u and u' is a C-node with a proper ancestor that has a child i^*-subtree. (b) The corresponding K_5 minor from [3].

Lemma 7 *Suppose there are two terminal nodes u and u' in T_r, and let m be their closest common ancestor. Let P' be the unique path from m to r. If m has a proper ancestor in T_r with a child i^*-subtree, then the input graph is not planar.*

Proof. If m is a C-node, then the PC-tree has the form shown in Figure 6(a) and the input graph can be edge contracted to the K_5 in Figure 6(b) as follows. First, since r and its ancestors are P-nodes, edge contract the proper ancestor of i into one vertex t and do nothing to i and r. For each C-node c in $P' - m$, edge contract its RBC so that only the parent and child of c in P' remain. Then, edge contract $P' - m$ into r. Similarly, edge contract the RBCs of C-nodes in $P - m$ into a single edge per C-node. Then, edge contract the proper descendants of m leading to u into either u if u is a P-node or a child of u if u is a C-node. Likewise, edge contract the proper descendants of m leading to u' into either u' if it is a P-node or a child of u' if u' is a C-node.

On the other hand, if m is a P-node, then all C-nodes in T_r can be edge contracted as described above. Then, the $K_{3,3}$ given for Lemma 2.5 in [19] is applicable (see Figure 3). □

The proof of Lemma 7 is also important because it demonstrates the actual method by which K_5 homeomorphs are found by the PC-tree algorithm, which also contradicts [19]: "we could have three terminal nodes being neighbors of a C-node, in which case we would get a subgraph homeomorphic to K_5 as illustrated in Fig. 6." The K_5 in Figure 6 of [19] is equivalent to Figure 6(b). It does not result in three terminal nodes, but is instead discovered by the condition in Lemma 7. Moreover, the case of three terminal node neighbors of a C-node depicted in Figure 7 should be part of an extension of Theorem 2.4 in [19] to PC-trees containing C-nodes, but again the proof does not extend to general PC-trees because the $K_{3,3}$ identified in the proof cannot always be obtained. Lemma 8 provides the proper extension of the three terminal node case to general PC-trees that contain C-nodes.

Lemma 8 *If T_r contains three terminal nodes, then the input graph is not planar.*

Proof. The proof of Theorem 2.4 in [19] provides the proper $K_{3,3}$ in PC-trees containing only P-nodes (see Figure 2). For general PC-trees containing C-nodes, only proper descendants of r in T_r need to be considered since r and its ancestors are P-nodes.

For each of the three terminal nodes, denoted i_1, i_2 and i_3, let P_1, P_2 and P_3 denote the critical paths from each terminal node to r. Without loss of generality, label the terminal nodes so that the join point j_1 of P_1 and P_2 is equal or descendant to the join point j_2 of the first two paths with P_3. The endpoints of these critical paths are r and each of the terminal nodes. The endpoint r is a P-node. For each terminal C-node, edge contract the children of the RBC into a single vertex so that only the parent and one child of the C-node remain, then use the child of the C-node as an image vertex of a $K_{3,3}$, either from Figure 2 or Figure 7 depending on the conditions described below.

For each internal C-node of each critical path except j_1 and j_2 (if either is indeed a C-node), edge contract the RBC to a single edge containing the parent and child in the critical path. Since the endpoints of the critical paths have already been discussed, this leaves only j_1 and j_2 to consider.

If both j_1 and j_2 are P-nodes, then clearly j_1 can be used as the image vertex w in Figure 2, and the $K_{3,3}$ identified in [19] for the three terminal node case can still be obtained. Thus, suppose one or both of j_1 and j_2 are C-nodes.

Suppose $j_1 \neq j_2$. If j_1 is a P-node, then j_2 must be a C-node. Let c_3 denote the child of j_2 that leads to i_3, and let $c_{1,2}$ denote the child of j_2 that leads to j_1. In this case j_1 can again be used as the image vertex w. The path from w to r leads up to $c_{1,2}$. Then it follows the RBC path from $c_{1,2}$ through c_3 to the parent of j_2 then up to r. On the other hand, if j_1 is a C-node with parent p and children c_1 and c_2 leading to i_1 and i_2, then the RBC paths from p to each of c_1 and c_2 can be contracted to a single edge. If j_2 is a P-node, then p is the desired vertex w and we are done. If j_2 is a C-node, then j_2 must be a proper ancestor of p. Again, we let p be the desired vertex w since the path from w to r can be obtained by going around the RBC of j_2 as described above.

Finally, suppose $j_1 = j_2$ is a C-node. Let c_1, c_2 and c_3 denote the children of the C-node in RBC order that lead to each of the respective terminal nodes, and let p denote the parent of the C-node. Unless the biconnected component represented by the C-node happens to have an internally embedded path connecting c_2 and p, the desired vertex w in the $K_{3,3}$ of Figure 2 cannot be obtained. Figure 7(a) depicts the PC-tree for this case, which reduces to the $K_{3,3}$ in Figure 7(b). □

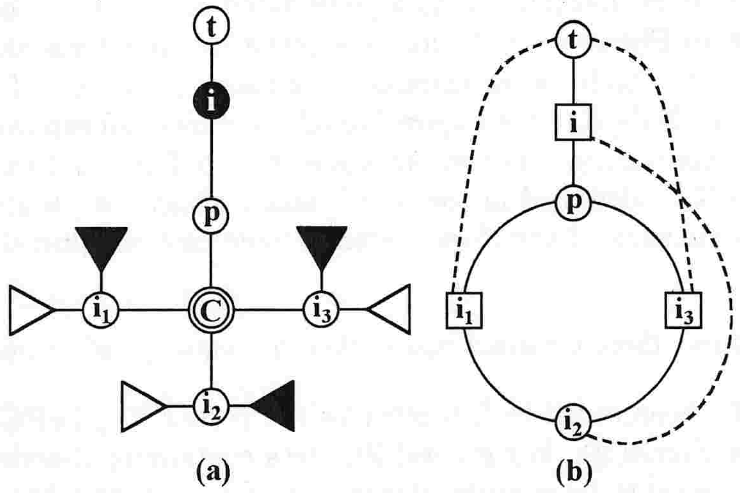


Figure 7: (a) A PC-tree with three terminal node proper descendants of a C-node. Note that the paths from the terminal nodes to the RBC vertices of the C-node have been contracted. (b) The corresponding $K_{3,3}$ minor from [3].

3.5 The case of zero terminal nodes

Lemma 9 presents an additional planarity reduction for zero terminal nodes, which occurs in the final step of embedding every graph. This case is easy to resolve, but it is worth mentioning since it is essentially a missing planarity reduction.

Lemma 9 *If, during the embedding of a biconnected graph G, there is a step i for which zero terminal nodes are identified, then the PC-tree can be reduced to a single C-node plus P-node neighbors for the RBC of the C-node.*

Proof. If there are no terminal nodes, then there are no i^*-subtrees within T_r. Having no i^*-subtrees prior to the last step contradicts the biconnectedness of G. Since G is biconnected, by definition its final embedding can be represented as described. □

4 Proof of Correctness for Modified PC-tree Algorithm

This section presents a proof of correctness for the PC-tree algorithm as modified by the lemmas and corollaries of Section 3. First, the planarity reduction patterns are clearly characterized with property statements below. Then, violations of the properties are mapped to the lemmas and corollaries so that it is clear that the planarity reduction patterns are the only ones that do not result in a nonplanarity condition. Since it is clear how to maintain planarity for each of the planarity reductions, the correctness of the algorithm follows.

For the two terminal node case, let u and u' denote the two terminal nodes. Let m denote the closest common ancestor of u and u', and let P denote the PC-tree path (u, ..., m, ..., u'). Let P' denote the PC-tree path (r, ..., m). For the one terminal node case, let u denote the terminal node and let u' denote the ancestor of u closest to the root of T_r that has a child i^*-subtree. Let m be a second label for u'. Let P denote the path (u, ..., u'), and let P' denote the path (r, ..., u'). To simplify the statement of properties, consider path P to be arranged horizontally in the plane, and consider P' as extending vertically upward from P. Let L denote an infinite horizontal line that contains P.

Property 1 *Nodes in $P' - \{m\}$ have no child i^*-subtrees.*

Property 2 *The children of nonterminal nodes in P are arranged so that all child i-subtrees are above L and all child i^*-subtrees are below L.*

Property 3 *Except for the case of one terminal C-node and $u = u'$, the children of terminal nodes in P are arranged so that all child i-subtrees are above L and all child i^*-subtrees are below L.*

Property 4 *For the case of one terminal C-node and $u = u'$, let p be the parent of u and let w and w' be the first child i^*-subtrees in each of the two RBC paths extending from p. The children of u on the RBC path strictly between w and w' that contains p must be the roots of all child i-subtrees of u.*

The proof of correctness of the modified PC-tree planarity algorithm in Theorem 10 will show that violations of the four properties above result in a nonplanarity condition. That the above properties characterize the planarity reductions in [19] and that the planarity reductions embed all back edges from i to descendants of r while maintaining planarity are taken to be straightforward. Moreover, the fact that maintaining planarity through all steps implies the planarity of the graph and that finding a nonplanarity condition in a step implies the nonplanarity of the graph are also taken to be evident.

Theorem 10 *The modified PC-tree planarity algorithm applies a planarity reduction to T_r if and only if there are no terminal nodes or if there are at most two terminal nodes and Properties 1, 2, 3, and 4 hold.*

Proof. Case *no terminal nodes*: The planarity reduction described in Lemma 9 is applied.

Case *one terminal node*: Property 1 holds by definition. Property 2 holds if $u = u'$ because there are no nonterminal nodes in P. Property 2 holds if $u \neq u'$ except for nonplanarity conditions due to Corollary 4. If $u \neq u'$, then Property 3 holds except for nonplanarity conditions due to Lemma 2 and Property 4 holds degenerately (is not applicable). On the other hand, if $u = u'$, then Property 3 holds degenerately, and Property 4 holds except for nonplanarity conditions due to Lemma 3.2 of [19].

Case *two terminal nodes*: Property 1 holds except for nonplanarity conditions due to Lemma 7. Property 2 holds except for nonplanarity conditions due to Lemma 3.

Property 3 holds except for nonplanarity conditions due to Lemma 2 and Property 4 holds degenerately.

Case *more than two terminal nodes*: If there are at least three terminal nodes in T_r, then the input graph is not planar according to Lemma 8. Hence no planarity reduction is applied. □

5 Issues Concerning Linear-Time Performance

Shih and Hsu [19] present the ideas necessary to achieve linear total work for the identification of terminal nodes, i-subtrees and i^*-subtrees in all steps of the PC-tree algorithm. However, there are two impediments to achieving linear time by the methods stated in [19]; both are complexities that arise when planarity reductions are applied to a PC-tree that contains C-nodes.

5.1 Maintaining the RBC when flipping C-nodes

The claim that the "RBC will be stored as a circular doubly linked list" [19, p. 184] cannot be supported. When the representative bounding cycles of C-nodes must be joined together, the direction of traversal of two successive C-nodes may be reversed at the intervening P-node depending on which path contains the child i-subtrees in each C-node. Joining the RBCs of two such C-nodes into a circular doubly linked list would require the inversion of links in the RBC nodes of one of the two C-nodes. It is easy to create planar graphs in which $\Theta(n^2)$ link inversions occur in total. To solve this problem, one can represent the RBC with a *discordant list* (defined in [3]). When merging the RBCs of two C-nodes separated by a P-node, only the neighbors of the P-node in the two RBCs are linked together. If the merge must be done such that one C-node is flipped relative to the other, then the resulting RBC pointers after the merge will be in discord. However, traversal of the RBC is still possible with only a little extra effort. When a traversal arrives at a node v from a predecessor p along a discordant RBC, the successor of v is indicated by one of its two RBC pointers. The pointer to use is the one that does not indicate p. Figure 8 illustrates this concept.

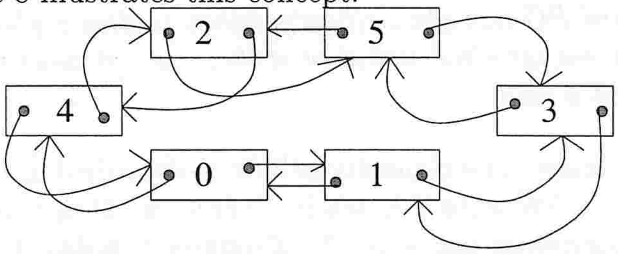


Figure 8: A discordant list of size 3 or more can be traversed by taking whichever pointer does not lead back to the preceding node (from [3]).

5.2 On the infeasibility of C-nodes as parents

In the conceptual ideal, every C-node and P-node indicates its parent according to the PC-tree definition. However, the child P-nodes in the RBC of a C-node cannot indicate the C-node as their parent. Consider a C-node with the following properties: 1) The RBC of the C-node has a subset S of children that root i^*-subtrees which all have back edges only to the last vertex to be processed, and 2) the RBC of the C-node also has an $O(n)$ sized succession of children that root i^*-subtrees which connect to proper ancestors of i such that successive steps of the algorithm merge the RBC of the original C-node into other ancestor C-nodes. At each merge, the members of S must be reparented to point to the new C-node that becomes their parent after the planarity reduction. This reparenting must be performed on a number of i^*-subtree roots that is a constant fraction of n, and the set of reparenting operations must be performed each time the parent C-node of the members of S must be changed to some other ancestor C-node with which the parent is merged. This results in $\Theta(n^2)$ performance in the worst case. The work could be substantially reduced by using the methods of the union-find data structure (also called a disjoint set data structure in [6]), but it would then have to be shown that the result is not super-linear, which is the case for generalized union-find operations. Either way, [19] presents neither this more sophisticated parenting strategy nor the required proof.

Perhaps the simplest strategy to solve this problem is not to adopt a complex parenting strategy and present a complex proof, but rather to let the parent pointer of all children of a C-node simply be *nil*, indicating they are part of a C-node, and keeping a pointer from each C-node child to its entry in the RBC. To find the parent of any node whose parent pointer is *nil*, traverse both directions around the RBC in parallel. This will obtain the parent of the C-node by the shorter path, so that the work done will not exceed a constant factor of the length of RBC that will be eliminated during the planarity reduction in the same step. This is analogous to the method used in [3] to traverse the external faces of biconnected components that are merged during the processing of a vertex.

6 Empirical Results and Future Work

This paper has reported and solved a number of additional theoretical complexities that arise in the published version of the Shih-Hsu PC-tree planarity algorithms [19]. A few years earlier, Thomas [20] provided an alternate formulation of the Shih-Hsu planarity algorithm that achieved linear-time performance for triconnected graphs. Thomas points out that significant additional technical complications would arise when accounting for graphs of lower connectivity and when one requires a planar embedding.

Our implementation efforts have been based on extending the formulation in Thomas' notes as a way of better understanding and correcting the problems with the PC-tree formulation in [19]. The resulting LEDA-based implementation contains code manifestations of the PC-tree problem solutions reported in this paper. We have achieved a linear-time implementation, both for producing a planar embedding and for isolating a Kuratowski subgraph in a nonplanar graph. We have performed the same empirical tests used in LEDA to compare the Hopcroft-Tarjan and Booth-Lueker implementations, and all results are consistent with the results for maximal planar graphs (MP) and their non-planar counterparts created by adding one random edge (MP+e). Figure 9 presents the MP and MP+e empirical comparisons of our current implementation with the Hopcroft-Tarjan (HT) and Booth-Lueker (BL) implementations in LEDA, as well as a non-LEDA implementation of the Boyer-Myrvold (BM) algorithm reported in [2]. Note that there are no HT results for nonplanar graphs because LEDA does not implement Williamson's Kuratowski subgraph isolator (indeed Williamson [24] knows of no $O(n)$ implementation).

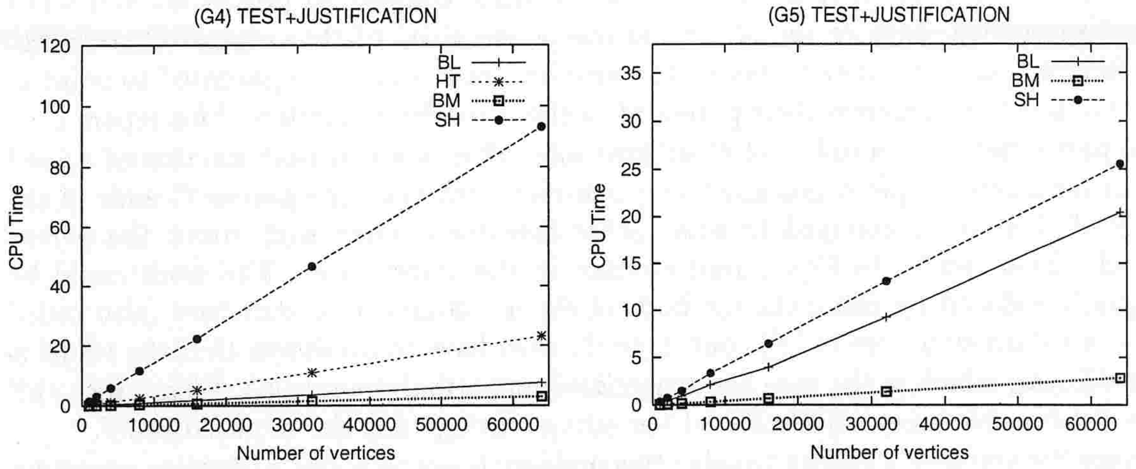


Figure 9: Empirical results comparing the SH, HT, BL and BM implementations on testing and justifying maximal planar graph (left) and their nonplanar counterparts (right) consisting of an extra random edge. The justification consists of creating an embedding for a planar graph or isolating a Kuratowski subgraph of a nonplanar graph. Results for HT on nonplanar graphs cannot be obtained from LEDA.

Although our implementation is not yet competitive with HT and BL, we believe that it can be made more competitive, in part through further application of some of the methods of the BM algorithm, which currently has the fastest implementation by about 2.5 times on planar graphs and about 8 times on nonplanar graphs. Our implementation efforts to date have been principally concerned with correctness and linear-time performance. The correctness concerns led us to extend Thomas' formulation based on an understanding of the original LEC algorithm (this formulation appears in [17]). We believe that the success of this approach in finding and solving problems with the PC-tree formulation substantiates the further investigation and exposition of the SH algorithm as an LEC-type algorithm. Indeed, future work shall consist of refining this alternate formulation with the ultimate goal of developing a unified LEC-type framework for describing the SH, BL and BM algorithms. As Williamson [22] notes, "it would be desirable to have not one but several basically different [linear time Kuratowski subgraph isolators]" because the condition of linearity "forces the emergence of a certain level of insight into the structure of nonplanar graphs and Kuratowski's theorem." The PC-tree formulation [19], augmented by the corrections in this paper, two variations of the BM algorithm [3, 2], and Karabeg's analysis [15] of the BL algorithm collectively demonstrate four different methods for the discovery of nonplanarity. Along with the correspondence drawn between HT and BL in [4], we believe that a more generalized LEC-type formulation could unify all of these methods and increase our graph theoretic understanding of planarity.

References

[1] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ–tree algorithms. *Journal of Computer and Systems Sciences*, 13:335–379, 1976.

[2] J. Boyer. *Simplified $O(n)$ algorithms for planar graph embedding, Kuratowski subgraph isolation and related problems.* Ph.D. Thesis, University of Victoria, 2001.

[3] J. Boyer and W. Myrvold. Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm. *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 140–146, 1999.

[4] E. R. Canfield and S. G. Williamson. The two basic linear time planarity algorithms: Are they the same? *Linear and Multilinear Algebra*, 26:243–265, 1990.

[5] N. Chiba, T. Nishizeki, A. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ–trees. *Journal of Computer and Systems Sciences*, 30:54–76, 1985.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Massachusetts, 1990.

[7] N. Deo. Note on Hopcroft and Tarjan planarity algorithm. *Journal of the Association for Computing Machinery*, 23:74–75, 1976.

[8] S. Even and R. E. Tarjan. Computing an *st*-numbering. *Theoretical Computer Science*, 2:339–344, 1976.

[9] J. Hopcroft and R. Tarjan. Efficient planarity testing. *Journal of the Association for Computing Machinery*, 21(4):549–568, 1974.

[10] W.-L. Hsu. Pc-trees vs. pq-trees. *Lecture Notes in Computer Science*, 2108:207–217, 2001.

[11] W.-L. Hsu. A simple test for the consecutive ones property. *Journal of Algorithms*, 42:1–16, 2002.

[12] W.-L. Hsu and R. McConnell. Pc-trees. to appear in *Handbook of Data Structures and Applications*, Dinesh P Mehta and Sartaj Sahni ed., 2003.

[13] W.-L. Hsu and R. McConnell. Pc-trees and circular-ones arrangements. *Theoretical Computer Science*, 296(1):59–74, 2003.

[14] M. Jünger, S. Leipert, and P. Mutzel. Pitfalls of using PQ-trees in automatic graph drawing. In G. Di Battista, editor, *Proc. 5th International Symposium on Graph Drawing '97*, volume 1353 of *Lecture Notes in Computer Science*, pages 193–204. Springer Verlag, Sept. 1997.

[15] A. Karabeg. Classification and detection of obstructions to planarity. *Linear and Multilinear Algebra*, 26:15–38, 1990.

[16] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In P. Rosenstiehl, editor, *Theory of Graphs*, pages 215–232, New York, 1967. (Proc. Int. Symp. Rome, July 1966), Gordon and Breach.

[17] A. Noma. Análise experimental de algoritmos de planaridade. Master's thesis, Universidade de São Paulo, May 2003. in Portuguese.

[18] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

[19] W.-K. Shih and W.-L. Hsu. A new planarity test. *Theoretical Computer Science*, 223:179–191, 1999.

[20] R. Thomas. Planarity in linear time. `http://www.math.gatech.edu/~thomas/`, June 1997.

[21] S. G. Williamson. Embedding graphs in the plane- algorithmic aspects. *Ann. Disc. Math.*, 6:349–384, 1980.

[22] S. G. Williamson. Depth-first search and Kuratowski subgraphs. *Journal of the Association for Computing Machinery*, 31(4):681–693, 1984.

[23] S. G. Williamson. *Combinatorics for Computer Science.* Computer Science Press, Rockville, Maryland, 1985.

[24] S. G. Williamson. *Personal Communication during Boyer's Ph.D. Defense.* August 24, 2001.

Lempel, Even, and Cederbaum Planarity Method

John M. Boyer[1], Cristina G. Fernandes[2*], Alexandre Noma[2**], and José Coelho de Pina[2*].

[1] PureEdge Solutions Inc. jboyer@acm.org
[2] University of São Paulo, Brazil {cris,noma,coelho}@ime.usp.br

Abstract. We present a simple pedagogical graph theoretical description of Lempel, Even, and Cederbaum (LEC) planarity method based on concepts due to Thomas. A linear-time implementation of LEC method using the PC-tree data structure of Shih and Hsu is provided and described in details. We report on an experimental study involving this implementation and other available linear-time implementations of planarity algorithms.

1 Introduction

The first linear-time planarity testing algorithm is due to Hopcroft and Tarjan [9]. Their algorithm is an ingenious implementation of the method of Auslander and Parter [1] and Goldstein [8]. Some notes to the algorithm were made by Deo [6], and significant additional details were presented by Williamson [20, 21] and Reingold, Nievergelt, and Deo [16].

The second method of planarity testing proven to achieve linear time is due to Lempel, Even, and Cederbaum (LEC) [13]. This method was optimized to linear time thanks to the *st*-numbering algorithm of Even and Tarjan [7] and the PQ-tree data structure of Booth and Lueker (BL) [2]. Chiba, Nishizeki, Abe, and Ozawa [5] augmented the PQ-tree operations so that a planar embedding is also computed in linear time.

All these algorithms are widely regarded as being quite complex [5, 12, 17]. Recent research efforts have resulted in simpler linear-time algorithms proposed by Shih and Hsu (SH) [10, 17, 18] and by Boyer and Myrvold (BM) [3, 4]. These algorithms implement LEC method and present similar and very interesting ideas. Each algorithm uses its own data structure to efficiently maintain relevant information on the (planar) already examined portion of the graph.

The description of SH algorithm made by Thomas [19] provided us with the key concepts to give a simple graph theoretical description of LEC method. This description increases the understanding of BL, SH, and BM algorithms, all based on LEC method.

* Research partially supported by PRONEX/CNPQ 664107/1997-4 (Brazil).
** Supported by FAPESP 00/03969-2 (Brazil).

Section 2 contains definitions of the key ingredients used by LEC method. In Section 3, an auxiliary algorithm is considered. LEC method is presented in Section 4 and an implementation of SH algorithm is described in Section 5. This implementation is available at `http://www.ime.usp.br/~coelho/sh/` and, as far as we know, is the unique available implementation of SH algorithm, even though the algorithm was proposed about 10 years ago. Finally, Section 6 reports on an experimental study.

2 Frames, XY-paths, XY-obstructions and planarity

This section contains the definitions of some concepts introduced by Thomas [19] in his presentation of SH algorithm. We use these concepts in the coming sections to present both LEC method and our implementation of SH algorithm.

Let H be a planar graph. A subgraph F of H is a *frame of* H if F is induced by the edges incident to the external face of a planar embedding of H (Figs. 1(a) and 1(b)).

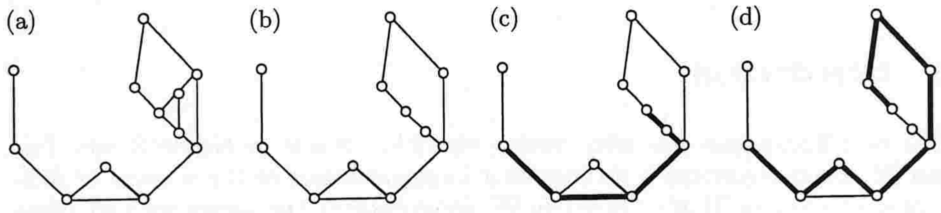


Fig. 1. (a) A graph H. (b) A frame of H. (c) A path P in a frame. (d) The complement of P.

If G is a connected graph, H is a planar induced subgraph of G and F is a frame of H, then we say that F is a *frame of* H *in* G if it contains all vertices of H that have a neighbor in $V_G \setminus V_H$. Neither every planar induced subgraph of a graph G has a frame in G (Fig. 2(a)) nor every induced subgraph of a planar graph G has a frame in G (Fig. 2(b)). The connection between frames and planarity is given by the following lemma.

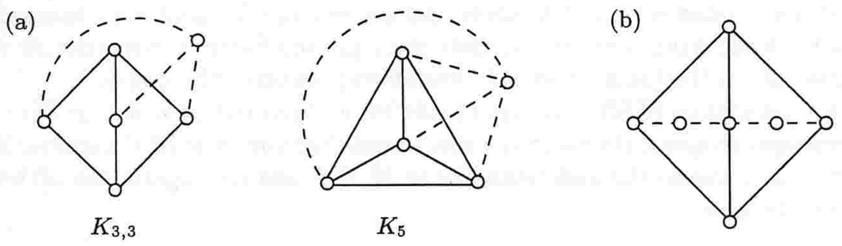


Fig. 2. (a) Subgraphs of $K_{3,3}$ and K_5 induced by the solid edges have no frames. (b) Subgraph induced by the solid edges has no frame in the graph.

Lemma 1 (Thomas [19]). *If H is an induced subgraph of a planar graph G such that $G - V_H$ is connected, then H has a frame in G.* ■

Let F be a frame of H and P be a path in F. The *basis of* P is the subgraph of F formed by all blocks of F which contain at least one edge of P. Let $C_1, C_2, \ldots, C_k$ be the blocks in the basis of P. For $i = 1, 2, \ldots, k$, let $P_i := P \cap C_i$ and, if C_i is a cycle, let $\bar{P}_i := C_i \setminus P_i$, otherwise let $\bar{P}_i := P_i$. The *complement of* P *in* F is the path $\bar{P}_1 \cup \bar{P}_2 \cup \ldots \cup \bar{P}_k$, which is denoted by $\bar{P}$. If $E_P = \emptyset$ then $\bar{P} := P$ (Figs. 1(c) and 1(d)).

Let W be a set of vertices in H and Z be a set of edges in H. A vertex v in H *sees* W *through* Z if there is a path in H from v to a vertex in W with all edges in Z. Let X and Y be sets of vertices of a frame F of H. A path P in F with basis S is an *XY-path* (Fig. 3) if

(p1) the endpoints of P are in X;
(p2) each vertex of S that sees X through $E_F \setminus E_S$ is in P;
(p3) each vertex of S that sees Y through $E_F \setminus E_S$ is in $\bar{P}$;
(p4) no component of $F - V_S$ contains vertices both in X and in Y.

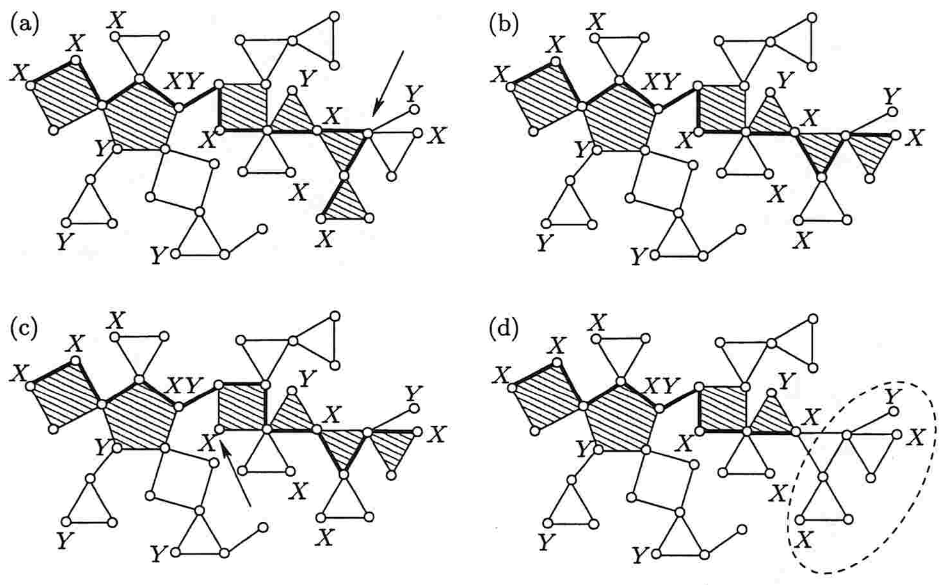


Fig. 3. In (a), (b), (c), and (d), let P denote the thick path; its basis is shadowed. (a) P is not an XY-path since it violates (p3). (b) P is an XY-path. (c) P is not an XY-path since it violates (p2). (d) P is not an XY-path since it violates (p4).

There are three types of objects that obstruct an XY-path to exist. They are called *XY-obstructions* and are defined as

(o1) a 5-tuple (C, v_1, v_2, v_3, v_4) where C is a cycle of F and v_1, v_2, v_3, and v_4 are distinct vertices in C that appear in this order in C, such that v_1 and v_3 see X through $E_F \setminus E_C$ and v_2 and v_4 see Y through $E_F \setminus E_C$;
(o2) a 4-tuple (C, v_1, v_2, v_3) where C is a cycle of F and v_1, v_2, and v_3 are distinct vertices in C that see X and Y through $E_F \setminus E_C$;
(o3) a 4-tuple (v, K_1, K_2, K_3) where $v \in V_F$ and K_1, K_2, and K_3 are distinct components of $F - v$ such that K_i contains vertices in X and in Y.

The existence of an XY-obstruction is related to non-planarity as follows.

Lemma 2 (Thomas [19]). *Let H be a planar connected subgraph of a graph G and w be a vertex in $V_G \setminus V_H$ such that $G - V_H$ and $G - (V_H \cup \{w\})$ is connected. Let F be a frame of H in G, let X be the set of neighbors of w in V_F and let Y be the set of neighbors of $V_G \setminus (V_H \cup \{w\})$ in V_F. If F has an XY-obstruction then G has a subdivision of $K_{3,3}$ or K_5.*

Sketch of the proof: An XY-obstruction of type (o1) or (o3) indicates a $K_{3,3}$-subdivision. An XY-obstruction of type (o2) indicates either a K_5-subdivision or a $K_{3,3}$-subdivision (Fig. 4). ■

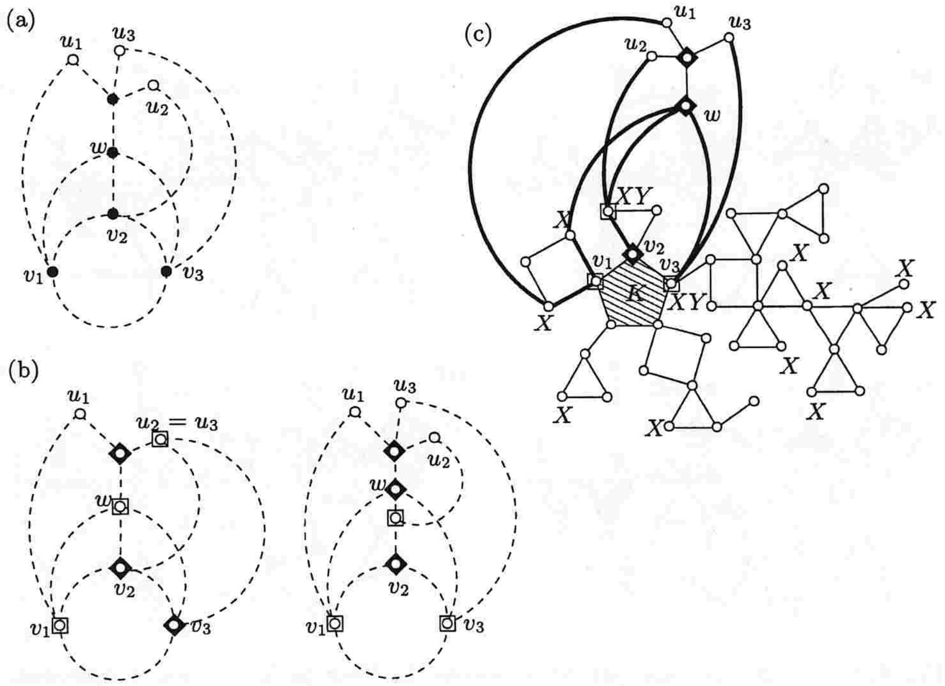


Fig. 4. Some situations with an XY-obstruction (C, v_1, v_2, v_3) of type (o2). The dashed lines indicate paths. In u_iv_i-path, u_i is the only vertex in $(V_H \setminus (V_H \cup \{w\}))$, for $i = 1, 2, 3$. (a) Subdivision of K_5 coming from an XY-obstruction. (b) Subdivisions of $K_{3,3}$ coming from an XY-obstruction. (c) Concrete example of an XY-obstruction leading to a $K_{3,3}$-subdivision.

3 Finding XY-paths or XY-obstructions

Let F be a connected frame and let X and Y be subsets of V_F. If F is a tree, then finding an XY-path or an XY-obstruction is an easy task. The following algorithm finds either an XY-path or an XY-obstruction in F manipulating a tree that represents F.

Let $\mathcal{B}$ be the set of blocks of a connected graph H and let T be the tree with vertex set $\mathcal{B} \cup V_H$ and edges of the form Bv where $B \in \mathcal{B}$ and $v \in V_B$. We call T the *block tree*[3] of H (Fig. 5). Each node of T in $\mathcal{B}$ is said a *C-node* and each node of T in V_H is said a *P-node*.

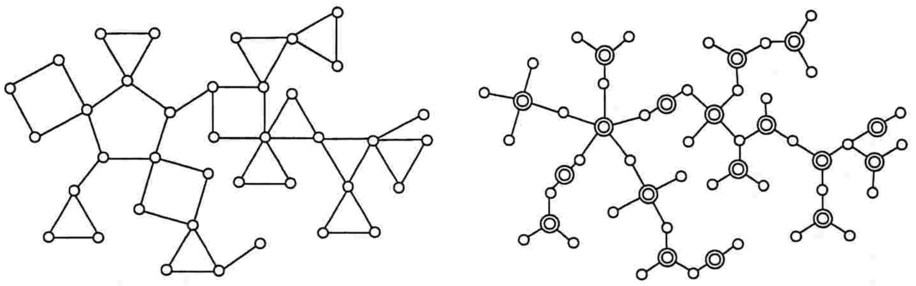


Fig. 5. A graph and its block tree.

Algorithm Central(F, X, Y). Receives a connected frame F and subsets X and Y of V_F and returns either an XY-path or an XY-obstruction in F.

Let T_0 be the block tree of F. The algorithm is iterative and each iteration begins with a subtree T of T_0, subsets X_T and Y_T of V_T and subsets W and Z of V_F. The sets X_T and Y_T are formed by the nodes of T that see X and Y through $E_{T_0} \setminus E_T$, respectively. The sets W and Z contain the P-nodes of T_0 that see X and Y through $E_{T_0} \setminus E_T$, respectively. At the beginning of the first iteration, $T = T_0$, $X_T = X$, $Y_T = Y$, $W = X$, and $Z = Y$. Each iteration consists of the following:

```
Case 1: Each leaf of $T$ is in $X_T \cap Y_T$ and $T$ is a path.
    Let $R$ be the set of P-nodes of $T$.
    For each C-node $C$ of $T$, let $X_C := V_C \cap (W \cup R)$ and $Y_C := V_C \cap (Z \cup R)$.
    Case 1A: Each C-node $C$ of $T$ has a path $P_C$ containing $X_C$ and internally
            disjoint from $Y_C$.
        Let $P_T$ be the path in $F$ obtained by the concatenation of the paths in
            $\{P_C : C$ is a C-node of $T\}$.
        Let $P$ be a path in $F$ with endpoints in $X$, containing $P_T$ and containing
            $V_C \cap W$ for each block $C$ in the basis of $P$.
        Return $P$ and stop.
```

[3] The leaves in V_H make the definition slightly different than the usual.

```
    **Case 1B:** There exists a C-node $C$ of $T$ such that no path containing $X_C$ is
        internally disjoint from $Y_C$.
        Let $v_1, v_2, v_3$, and $v_4$ be distinct vertices of $C$ appearing in this order in $C$,
        such that $v_1$ and $v_3$ are in $X_C$ and $v_2$ and $v_4$ are in $Y_C$.
        Return $(C, v_1, v_2, v_3, v_4)$ and stop.
**Case 2:** Each leaf of $T$ is in $X_T \cap Y_T$ and there exists a node $v$ of $T$ with degree greater
    than 2.
    **Case 2A:** $v$ is a C-node.
        Let $C$ be the block of $F$ corresponding to $v$.
        Let $v_1, v_2$, and $v_3$ be distinct P-nodes adjacent to $v$ in $T$.
        Return $(C, v_1, v_2, v_3)$ and stop.
    **Case 2B:** $v$ is a P-node.
        Let $C_1, C_2$, and $C_3$ be distinct C-nodes adjacent to $v$ in $T$.
        Let $K_1, K_2$, and $K_3$ be components of $F - v$ such that $C_i$ is a block of $K_i + v$
        $(i = 1, 2, 3)$.
        Return $(v, K_1, K_2, K_3)$ and stop.
**Case 3:** There exists a leaf $f$ of $T$ not in $X_T \cap Y_T$.
    Let $u$ be the node of $T$ adjacent to $f$.
    Let $T' := T - f$.
    Let $X_{T'} := (X_T \setminus \{f\}) \cup \{u\}$ if $f$ is in $X_T$; otherwise $X_{T'} := X_T$.
    Let $Y_{T'} := (Y_T \setminus \{f\}) \cup \{u\}$ if $f$ is in $Y_T$; otherwise $Y_{T'} := Y_T$.
    Let $W' := W \cup \{u\}$ if $f$ is in $X_T$ and $u$ is a P-node; otherwise $W' := W$.
    Let $Z' := Z \cup \{u\}$ if $f$ is in $Y_T$ and $u$ is a P-node; otherwise $Z' := Z$.
    Start a new iteration with $T', X_{T'}, Y_{T'}, W'$, and $Z'$ in the roles of $T, X_T, Y_T, W$,
    and $Z$, respectively.
```

The execution of the algorithm consists of a sequence of "reductions" made by Case 3 followed by an occurrence of either Case 1 or Case 2. At the beginning of the last iteration, the leaves of T are called *terminals*. The concept of a terminal node is used in a fundamental way by SH algorithm. The following theorem follows from the correctness of the algorithm.

Theorem 3 (Thomas [19]). *If F is a frame of a connected graph and X and Y are subsets of V_F, then either there exists an XY-path or an XY-obstruction in F.* ■

4 LEC planarity testing method

One of the ingredients of LEC method is a certain ordering $v_1, v_2, \ldots, v_n$ of the vertices of the given graph G such that, for $i = 1, \ldots, n$, the induced subgraphs $G[\{v_1, \ldots, v_i\}]$ and $G[\{v_{i+1}, \ldots, v_n\}]$ are connected. Equivalently, G is connected and, for $i = 2, \ldots, n-1$, vertex v_i is adjacent to v_j and v_k for some j and k such that $1 \leq j < i < k \leq n$. A numbering of the vertices according to such an ordering is called a *LEC-numbering* of G. If the ordering is such that $v_1 v_n$ is an edge of the graph, the numbering is called an *st-numbering* [13]. One can show that every biconnected graph has a LEC-numbering.

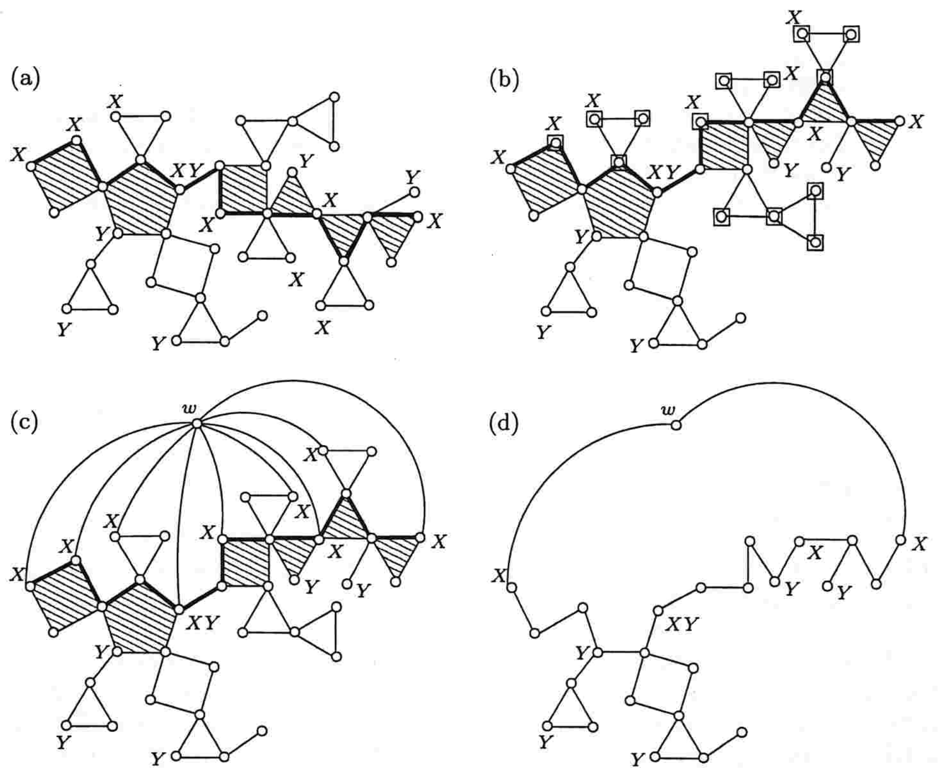


Fig. 6. (a) A frame F and an XY-path P in thick edges. (b) F after moving the elements of X to one side and the elements of Y to the other side of P. Squares mark vertices in $V_F \setminus V_{\bar{P}}$ that do not see $Y \setminus V_{\bar{P}}$ through $E_F \setminus E_S$, where $\bar{P}$ is the complement and S is the basis of P. (c) F together with the edges with one endpoint in F and the other in w. (d) A frame of $H + w$.

LEC method examines the vertices of a given biconnected graph, one by one, according to a LEC-numbering. In each iteration, the method tries to extend a frame of the subgraph induced by the already examined vertices. If this is not possible, the method declares the graph is non-planar and stops.

Method LEC(G). Receives a biconnected graph G and returns YES if G is planar, and NO otherwise.

Number the vertices of G according to a LEC-numbering. Each iteration starts with an induced subgraph H of G and a frame F of H in G. At the beginning of the first iteration, H and F are empty. Each iteration consists of the following:

Case 1: $H = G$.
  Return YES e stop.
Case 2: $H \neq G$.
  Let w be the smallest numbered vertex in $G - V_H$.
  Let $X := \{u \in V_F : uw \in E_G\}$.
  Let $Y := \{u \in V_F :$ there exists $v \in V_G \setminus (V_H \cup \{w\})$ such that $uv \in E_G\}$.

Case 2A: There exists an XY-obstruction in F.
Return NO and stop.
Case 2B: There exists an XY-path P in F.
Let $\bar{P} := \langle w_0, w_1, \ldots, w_k \rangle$ be the complement of P and let S be the basis of P.
Let R be the set of vertices in $V_F \setminus V_{\bar{P}}$ that do not see $Y \setminus V_{\bar{P}}$ through $E_F \setminus E_S$ (Figs. 6(a) and 6(b)).
Let F' be the graph resulting from the addition of w and the edges ww_0 and ww_k to the graph $F - R$ (Fig. 6(c)).
Let $H' := H + w$ (Fig. 6(d)).
Start a new iteration with H' and F' in the roles of H and F respectively.

The following invariants hold during the execution of the method.

(lec1) H and $G - V_H$ are connected graphs;
(lec2) F is a frame of H in G.

These invariants together with Lemmas 1 and 2 and Theorem 3 imply the correctness of the method and the following classical theorem.

Theorem 4 (Kuratowski). *A graph is planar if and only if it has no subdivision of $K_{3,3}$ or K_5.* ■

Three of the algorithms mentioned in the introduction are very clever linear-time implementations of LEC method. BL use an *st*-numbering instead of an arbitrary LEC-numbering of the vertices and use a PQ-tree to store F. SH use a DFS-numbering and a PC-tree to store F. BM also use a DFS-numbering and use still another data structure to store F. One can use the previous description easily to design a quadratic implementation of LEC method.

5 Implementation of SH algorithm

SH algorithm, as all other linear-time planarity algorithms, is quite complex to implement. The goal of this section is to share our experience in implementing it.

Let G be a connected graph. A *DFS-numbering* is a numbering of the vertices of G obtained from searching a DFS-tree of G in post-order. SH algorithm uses a DFS-numbering instead of a LEC-numbering. If the vertices of G are ordered according to a DFS-numbering, then the graph $G[\{i+1, \ldots, n\}]$ is connected, for $i = 1, \ldots, n$. As a DFS-numbering does not guarantee that $H := G[\{1, \ldots, i-1\}]$ is connected, if there exists a frame F of H and H is not connected, then F is also not connected. Besides, to compute (if it exists) a frame of $H + i$, it is necessary to compute an XY-path for each component of F that contains a neighbor of i.

Let v be a vertex of F and C be a block of F containing v and, if possible, a higher numbered vertex. We say v is *active* if v sees $X \cup Y$ through $E_F \setminus E_C$.

PC-tree

The data structure proposed by SH to store F is called a *PC-tree* and is here denoted by T. Conceptually, a PC-tree is an arborescence representing the relevant information of the block forest of F. It consists of *P-nodes* and *C-nodes*.

There is a P-node for each active vertex of F and a C-node for each cycle of F. We refer to a P-node by the corresponding vertex of F. There is an arc from a P-node u to a P-node v in T if and only if uv is a block of F. Each C-node c has a circular list, denoted $RBC(c)$, with all P-nodes in its corresponding cycle of F, in the order they appear in this cycle. This list starts by the largest numbered P-node in it, which is called its *head*. The head of the list has a pointer to c. Each P-node appears in at most one RBC in a non-head cell. It might appear in the head cell of several RBCs. Each P-node v has a pointer *nonhead_RBC_cell*(v) to the non-head cell in which it appears in an RBC. This pointer is NULL if there is no such cell. The name RBC extends for *representative bounding cycle* (Figs. 7(a)-(c)).

Let T' be the rooted forest whose node set coincides with the node set of T and the arc set is defined as follows. Every arc of T is an arc of T'. Besides these arcs, there are some *virtual* arcs: for every C-node c, there is an arc in T' from c to the P-node which is the head of $RBC(c)$ and there is an arc to c from all the other P-nodes in $RBC(c)$ (Fig. 7(d)). In the exposition ahead, we use on nodes of T concepts such as *parent*, *child*, *leaf*, *ancestral*, *descendant* and so on. By these, we mean their counterparts in T'.

Forest T' is not really kept by the implementation. However, during each iteration, some of the virtual arcs are determined and temporarily stored to avoid traversing parts of the PC-tree more than once. So, each non-head cell in an RBC and each C-node has a pointer to keep its virtual arc, when it is determined. The pointer is NULL while the virtual arc is not known.

Values $h(u)$ and $b(v)$

For each vertex u of G, denote by $h(u)$ the largest numbered neighbor of u in G. This value can be computed together with a DFS-numbering, and can be stored in an array at the beginning of the algorithm.

For each node v of T, let $b(v) := \max\{h(u) : u \text{ is a descendant of } v \text{ in } T\}$. For a C-node of T, this number does not change during the execution of the algorithm. On the other hand, for a P-node of T, this number might decrease because its set of descendants might shrink when T is modified. So, in the implementation, the value of $b(c)$ for a C-node c is computed and stored when c is created. It is the maximum over $b(u)$ for all u in the path in T corresponding to the XY-path in F that originated c. One way to keep $b(v)$ for a P-node v is, at the beginning of the algorithm, to build an adjacency list for G sorted by the values of h, and to keep, during the algorithm, for each P-node of T, a pointer to the last traversed vertex in its sorted adjacency list. Each time the algorithm needs to access $b(v)$ for a P-node v, it moves this pointer ahead on the adjacency list (if necessary) until (1) it reaches a vertex u which has v as its parent, in which case $b(v)$ is the maximum between $h(v)$ and $b(u)$, or (2) it reaches the end of the list, in which case $b(v) = h(v)$.

Traversal of the PC-tree

The traversal of the PC-tree T, inspired by Boyer and Myrvold [3, 4], is done as follows. To go from a P-node u to a node v which is an ancestral of u in T,

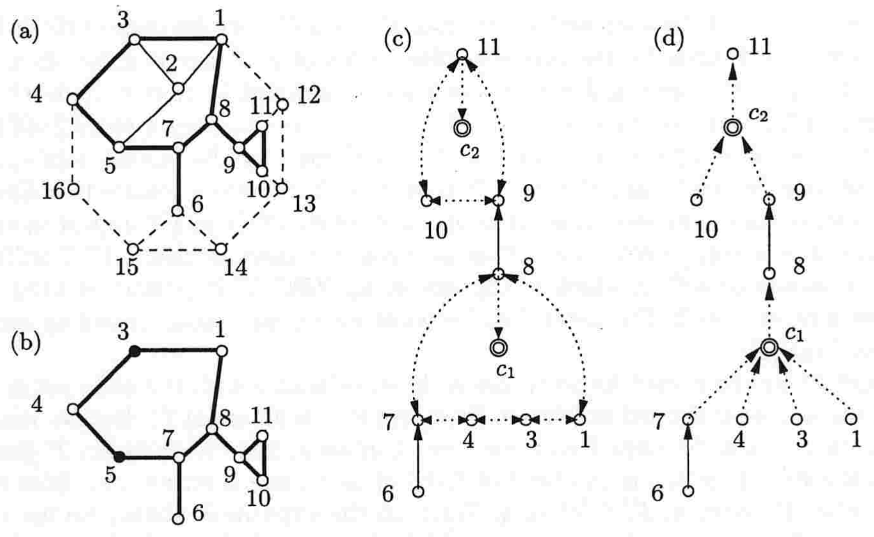


Fig. 7. (a) A graph G, a DFS-numbering of its vertices and, in thick edges, a frame F of $G[1..11]$ in G. (b) Black vertices in frame F are inactive. (c) The PC-tree T for F, with *RBC*s indicated in dotted. (d) Rooted tree T' corresponding to T; virtual arcs are dashed.

one starts with $x = u$ and repeats the following procedure until $x = v$. If x is a P-node and *nonhead_RBC_cell*(x) is NULL, move x up to its parent. If x is a P-node and *nonhead_RBC_cell*(x) is non-NULL, either its virtual arc is NULL or not. If it is non-NULL, move x to the C-node pointed by the virtual arc. Otherwise, starting at *nonhead_RBC_cell*(x), search the *RBC* in an arbitrary direction until either (1) the head of the *RBC* is reached or (2) a cell in the *RBC* with its virtual arc non-NULL is reached or (3) a P-node y such that $b(y) > w$ is reached. If (3) happens before (1), search the *RBC*, restarting at *nonhead_RBC_cell*(x), but in the other direction, until either (1) or (2) happens. If (1) happens, move x to the C-node pointed by the head. If (2) happens, move x to the C-node pointed by the virtual arc. In any case, search all visited cells in the *RBC* again, setting their virtual arcs to x. Also, set the virtual arc from x to the head of its *RBC*.

In a series of moments, the implementation traverses parts of T. For each node of T, there is a mark to tell whether it was already visited in this iteration or not. By visited, we mean a node which was assigned to x in the traversal process described above. Every time a new node is included in T, it is marked as unvisited. Also, during each phase of the algorithm where nodes are marked as visited, the algorithm stacks each visited node and, at the end of the phase, unstacks them all, undoing the marks. This way, at the beginning of each iteration, all nodes of T are marked as unvisited.

The same trick with a stack is done to unset the virtual arcs. When a virtual arc for a node v is set in the traversal, v is included in a second stack and, at the end of the iteration, this stack is emptied and all corresponding virtual arcs are set back to NULL.

Terminals

The next concept, introduced by SH, is the key on how to search efficiently for an XY-obstruction. A node t of T is a *terminal* if

(t1) $b(t) > w$;
(t2) t has a descendant in T that is a neighbor of w in G;
(t3) no proper descendant of t satisfies properties (t1) and (t2) simultaneously.

Because of the orientation of the PC-tree, one of the terminals from Section 4 might not be a terminal here. This happens when one of the terminals from Section 4 is an ancestor in the PC-tree of all others. An extra effort in the implementation is necessary to detect and deal with this possible extra terminal.

The first phase of an iteration of the implementation is the search for the terminals. This phase consists of, for each neighbor v of w such that $v < w$, traversing T starting at v until a visited node z is met. (Mark all nodes visited in the traversal; this will be left implicit from now on.) On the way, if a node u such that $b(u) > w$ is seen, mark the first such node as a *candidate-terminal* and, if z is marked as such, unmark it. The result from this phase is the list of terminals for each component of F.

Search for XY-obstructions

The second phase is the search for an XY-obstruction. First, if there are three or more terminals for some component of F, then there is an XY-obstruction of type either (o2) or (o3) in F (Case 2 of Central algorithm). We omit the details on how to effectively find it because this is a terminal case of the algorithm. Second, if there are at most two terminals for each component of F, then, for each component of F with at least one terminal, do the following. If it has two terminals, call them t_1 and t_2. If it has only one terminal, call it t_1 and let t_2 be the highest numbered vertex in this component. Test each C-node c on the path in T between t_1 and t_2 for an XY-obstruction of type (o1) (Case 1B of Central algorithm). The test decides if the cycle in F corresponding to c plays or not the role of C in (o1). Besides these tests, the implementation performs one more test in the case of two terminals. The least common ancestor m of t_1 and t_2 in T is tested for an XY-obstruction of type (o2), if m is a C-node, or an XY-obstruction of type (o3), if m is a P-node. This extra test arises from the possible undetected terminal.

To perform each of these tests, the implementation keeps one more piece of information for each C-node c. Namely, it computes, in each iteration, the number of P-nodes in $RBC(c)$ that see X through $E_F \setminus E_C$, where C is the cycle in F corresponding to c. This number is computed in the first phase. Each C-node has a counter that, at the beginning of each iteration, values 1 (to account for the head of its RBC). During the first phase, every time an RBC is entered through a P-node which was unvisited, the counter of the corresponding C-node is incremented by 1. As a result, at the end of the first phase, each (relevant) C-node knows its number.

For the test of a C-node c, the implementation searches $RBC(c)$, starting at the head of $RBC(c)$. It moves in an arbitrary direction, stopping only when it finds a P-node u (distinct from the head) such that $b(u) > w$. On the way, the implementation counts the number of P-nodes traversed. If only one step is given, it starts again at the head of $RBC(c)$ and moves to the other direction until it finds a P-node u such that $b(u) > w$, counting the P-nodes, as before. If the counter obtained matches the number computed for that C-node in the first phase, it passed the test, otherwise, except for two cases, there in an XY-obstruction of type (o1). The first of the two cases missing happens when there are exactly two terminals and c is the lower common ancestor of them. The second of the two cases happens when there is exactly one terminal and c is (potentially) the upper block in which the XY-path ends. The test required in these two cases is slightly different, but similar, and might give raise to an XY-obstruction of type (o1) or (o2). We omit the details.

PC-tree update

The last phase refers to Case 2B in LEC method. It consists of the modification of T according to the new frame. First, one has to add to T a P-node for w. Then, parts of T referring to a component with no neighbor of w remain the same. Parts of T referring to a component with exactly one neighbor of w are easily adjusted. So we concentrate on the parts of T referring to components with two or more neighbors of w. Each of these originates a new C-node. For each of them, the second phase determined the basis of an XY-path, which is given by a path Q in T. Path Q consists basically of the nodes visited during the second phase. Let us describe the process in the case where there is only one terminal. The case of two terminals is basically a double application of this one.

Call c the new C-node being created. Start $RBC(c)$ with its head cell, which refers to w, and points back to c. Traverse Q once again, going up in T. For each P-node u in Q such that *nonhead_RBC_cell*(u) is NULL, if $b(u) > w$ (here we refer to the possibly new value of $b(u)$, as u might have lost a child in the traversal), then an RBC cell is created, referring to u. It is included in $RBC(c)$ and *nonhead_RBC_cell*(u) is set to point to it. For each P-node u such that *nonhead_RBC_cell*(u) is non-NULL, let c' be its parent in T. Concatenate to $RBC(c)$ a part of $RBC(c')$, namely, the part of $RBC(c')$ that was not used to get to c' in any traversal in the second phase. To be able to concatenate without traversing this part, one can use a simple data structure proposed by Boyer and Myrvold [4, 3] to keep a doubled linked list. (The data structure consists of the cells with two indistinct pointers, one for each direction. To move in a certain direction, one starts making the first move in that direction, then, to keep moving in the same direction, it is enough to choose always the pointer that does not lead back to the previous cell.)

During the traversal of Q, one can compute the value of $b(c)$. Its value is simply the maximum of $b(u)$ over all node u traversed. This completes the description of the core of the implementation.

Certificate

To be able to produce a certificate for its answer, the implementation carries still more information. Namely, it carries the DFS-tree that originated the DFS-numbering of the vertices and, for each C-node, a combinatorial description of a planar embedding of the corresponding biconnected component where the P-nodes in its *RBC* appear all on the boundary of the same face. We omit the details, but one can find at `http://www.ime.usp.br/~coelho/sh/` the complete implementation, that also certificates its answer.

6 Experimental study

The main purpose of this study was to confirm the linear-time behavior of our implementation and to acquire a deeper understanding of SH algorithm. Boyer *et al.* [11] made a similar experimental study that does not include SH algorithm.

The LEDA platform has a planarity library that includes implementations of Hopcroft and Tarjan's (HT) and BL algorithms and an experimental study comparing them. The library includes the following planar graph generator routines: `maximal_planar_map` and `random_planar_map`. Neither of them generates plane maps according to the uniform distribution [14], but they are well-known and widely used. The following classes of graphs obtained through these routines are used in the LEDA experimental study:

(G1) random planar graphs;
(G2) graphs with a $K_{3,3}$: six vertices from a random planar graph are randomly chosen and edges among them are added to form a $K_{3,3}$;
(G3) graphs with a K_5: five random vertices from a random planar graph are chosen and all edges among them are added to form a K_5;
(G4) random maximal planar graphs;
(G5) random maximal planar graphs plus a random edge connecting two non-adjacent vertices.

Our experimental study extends the one presented in LEDA including our implementation of SH algorithm made on the LEDA platform and an implementation of BM algorithm developed in `C`. We performed all empirical tests used in LEDA to compare HT and BL implementations [15]. The experimental environment was a PC running GNU/Linux (RedHat 7.1) on a Celeron 700MHz with 256MB of RAM. The compiler was the `gcc 2.96` with options `-DLEDA_CHECKING_OFF -O`.

In the experiments [15, p. 123], BL performs the planarity test 4 to 5 times faster than our SH implementation in all five classes of graphs above. For the planar classes (G1) and (G4), it runs 10 times faster than our SH to do the planarity test and build the embedding. On (G2) and (G3), it is worse than our SH, requiring 10% to 20% more time for testing and finding an obstruction. On (G5), it runs within 65% of our SH time for testing and finding an obstruction. For the planarity test only, HT runs within 70% of our SH time for the planar classes (G1) and (G4), but performs slightly worse than our SH on (G2) and (G3). On (G5), it outperforms our SH, running in 40% of its time. For the planar classes (G1) and (G4), HT is around 4 times faster when testing and building the embedding. (The HT implementation in question has no option to produce an obstruction when the input graph is non-planar; indeed, there is no linear-time implementation known for finding the obstruction for it [22].) BM performs better than all, but, remember, it is the only one implemented in C and not in the LEDA platform. It runs in around 4% of the time spent by our SH for testing and building the embedding and, for the non-planar classes, when building the obstruction, it runs in about 15% of our SH time on (G2) and (G3) and in about 10% of our SH time on (G5). (There is no implementation of BM available that only does the planarity testing.) The time execution used on these comparisons is the average CPU time on a set of 10 graphs from each class.

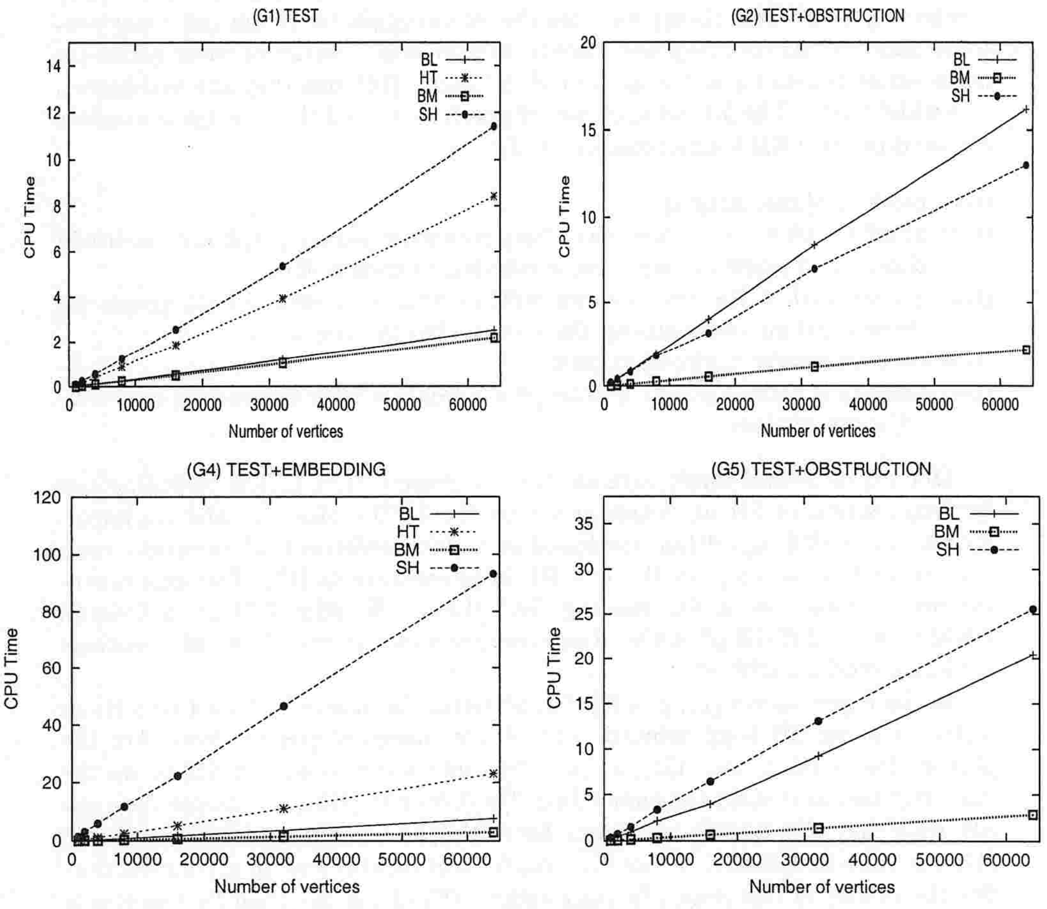


Fig. 8. Empirical results comparing SH, HT, BL, and BM implementations.

Figure 8 shows the average CPU time of each implementation on (a) (G1) for only testing planarity (against BM with testing and building an embedding, as there is no testing only available), (b) (G2) for testing and finding an obstruction (HT is not included in this table, by the reason mentioned above), (c) (G4) for testing and building an embedding, and (d) for testing and finding an obstruction (again, HT excluded).

We believe the results discussed above and shown in the table are initial and still not conclusive because our implementation is yet a prototype. (Also, in our opinion, it is not fair to compare LEDA implementations with C implementations.)

Our current understanding of SH algorithm makes us believe that we can design a new implementation which would run considerably faster. Our belief comes, first, from the fact that our current code was developed to solve the planarity testing only, and was later on modified to also produce a certificate for its answer to the planarity test. Building an implementation from the start thinking about the test and the certificate would be the right way, we believe, to have a more efficient code. Second, during the adaptation to build the certificate (specially the embedding when the input is planar) made us notice several details in the way the implementation of the test was done that could be improved. Even though, we decide to go forward with the implementation of the complete algorithm (test plus certificate) so that we could understand it completely before rewriting it from scratch. The description made on Section 5 already incorporates some of the simplifications we thought of for our new implementation. It is our intention to reimplement SH algorithm from scratch.

References

1. L. Auslander and S.V. Parter. On imbedding graphs in the plane. *Journal of Mathematics and Mechanics*, 10:517–523, 1961.
2. K.S. Booth and G.S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ–tree algorithms. *Journal of Computer and Systems Sciences*, 13:335–379, 1976.
3. J.M. Boyer and W. Myrvold. On the cutting edge: Simplified $O(n)$ planarity by edge addition. Preprint, 29pp.
4. J.M. Boyer and W. Myrvold. Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm. *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 140–146, 1999.
5. N. Chiba, T. Nishizeki, A. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ–trees. *Journal of Computer and Systems Sciences*, 30:54–76, 1985.
6. N. Deo. Note on Hopcroft and Tarjan planarity algorithm. *Journal of the Association for Computing Machinery*, 23:74–75, 1976.
7. S. Even and R.E. Tarjan. Computing an *st*-numbering. *Theoretical Computer Science*, 2:339–344, 1976.
8. A.J. Goldstein. An efficient and constructive algorithm for testing whether a graph can be embedded in a plane. In *Graph and Combinatorics Conf.* Contract No. NONR 1858-(21), Office of Naval Research Logistics Proj., Dep.of Math., Princeton U., 2 pp., 1963.

9. J. Hopcroft and R. Tarjan. Efficient planarity testing. *Journal of the Association for Computing Machinery*, 21(4):549–568, 1974.
10. W.L. Hsu. An efficient implementation of the PC-tree algorithm of Shih & Hsu's planarity test. Technical report, Inst. of Information Science, Academia Sinica, 2003.
11. M.P̃atrignani J.M.B̃oyer, P.F.C̃ortese and G.D̃i Battista. Stop minding your P's and Q's: Implementing a fast and simple DFS-based planarity testing and embedding algorithm. In G. Liotta, editor, *Graph Drawing (GD 2003)*, volume 2912 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2004.
12. M. Jünger, S. Leipert, and P. Mutzel. Pitfalls of using PQ-trees in automatic graph drawing. In G. Di Battista, editor, *Proc. 5th International Symposium on Graph Drawing '97*, volume 1353 of *Lecture Notes in Computer Science*, pages 193–204. Springer Verlag, Sept. 1997.
13. A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In P. Rosenstiehl, editor, *Theory of Graphs*, pages 215–232, New York, 1967. Gordon and Breach.
14. K. Mehlhorn and St. Näher. *The LEDA Platform of Combinatorial and Geometric Computing.* Cambridge Press, 1997.
15. A. Noma. Análise experimental de algoritmos de planaridade (in Portuguese). Master's thesis, Universidade de São Paulo, 2003. `http://www.ime.usp.br/dcc/posgrad/teses/noma/dissertation.ps.gz`.
16. E.M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.
17. W.-K. Shih and W.-L. Hsu. A new planarity test. *Theoretical Computer Science*, 223:179–191, 1999.
18. W.K. Shih and W.L. Hsu. A simple test for planar graphs. In *Proceedings of the International Workshop on Discrete Math. and Algorithms*, pages 110–122. University of Hong Kong, 1993.
19. R. Thomas. Planarity in linear time. `http://www.math.gatech.edu/~thomas/`, 1997.
20. S.G. Williamson. Embedding graphs in the plane – algorithmic aspects. *Ann. Disc. Math.*, 6:349–384, 1980.
21. S.G. Williamson. *Combinatorics for Computer Science.* Computer Science Press, Maryland, 1985.
22. S.G. Williamson. Personal Communication, August 2001.

Finding Large Planar Subgraphs and Large Subgraphs of a Given Genus

Gruia Călinescu * and Cristina G. Fernandes **

Georgia Institute of Technology

Abstract. We consider the MAXIMUM PLANAR SUBGRAPH problem - given a graph G, find a largest planar subgraph of G. This problem has applications in circuit layout, facility layout, and graph drawing. We improve to 4/9 the best known approximation ratio for the MAXIMUM PLANAR SUBGRAPH problem. We also consider a generalization of the previous problem, the MAXIMUM GENUS D SUBGRAPH problem - given a connected graph G, find a maximum subgraph of G of genus at most D. For the latter problem, we present a simple algorithm whose approximation ratio is 1/4.

1 Introduction

The MAXIMUM PLANAR SUBGRAPH problem is: given a graph G, find a subgraph of G of maximum size, where *size* is the number of edges. This problem has applications in circuit layout, facility layout, and graph drawing [F92, TDB88].

MAXIMUM PLANAR SUBGRAPH is known to be NP-Complete [LG77]. Therefore we are looking for polynomial-time approximation algorithms. For a graph G, let $Opt(G)$ be the maximum size of a planar subgraph of G. Given an algorithm A that takes (representations of) graphs G as input and outputs planar subgraphs of G, define A(G) to be the size of the planar graph A produces when G is the input. Now let us define A's *performance* or *approximation ratio* to be the infimum, over all (representations of) graphs G, of A$(G)/Opt(G)$ (if $Opt(G) > 0$, and 1 otherwise). In the literature, authors sometimes ensure that their performance ratio is at least one by defining it to be the reciprocal of ours. The goal is to find a polynomial-time approximation algorithm with approximation ratio as big as possible.

The following concept is useful for finding large planar subgraphs. A *triangular structure* is a graph whose all cycles are triangles (i.e., have length three). Note that triangular structures are planar: a triangular structure cannot contain

* College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, USA, e-mail: gruia@cc.gatech.edu. Research supported in part by NSF grant CCR-9319106.

** College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, USA, e-mail: cris@cc.gatech.edu. Research supported in part by CNPq (Brazil), under contract 200975/92-7.

a subdivision of K_5 or $K_{3,3}$. In fact, all the blocks of a triangular structure are edges or triangles.

Before the algorithm presented in [CFFK96], the best known approximation ratio for the MAXIMUM PLANAR SUBGRAPH problem was 1/3. The basic idea of that paper was to look for large triangular structures. More specifically, the best algorithm presented in [CFFK96], from now on denoted by A, simply outputs a maximum triangular structure in the input graph G (that is, which is a subgraph of G). Finding a maximum triangular structure in G is solvable in polynomial time, by using a matroid parity algorithm [LP86, GS85].

Given a graph H, let $mts(H)$ denote the number of edges in a maximum triangular structure in H. Define $\rho(H) = mts(H)/|E(H)|$ if $E(H) \neq \emptyset$, and $\rho(H) = 1$ if $E(H) = \emptyset$. If H is a maximum planar subgraph of G, then $mts(G) \geq mts(H) = \rho(H)|E(H)|$, implying that the approximation ratio of algorithm A is at least $\rho(H)$. In fact, it is exactly $\rho(G)$, if the input graph G happens to be planar.

In [CFFK96], it was proved that $\rho(H) \geq 2/5 = 0.4$, provided that H is planar. It was also shown that there is a family of planar graphs H_i for which $\rho(H_i)$ tends to 4/9. In this paper, we show that for any planar graph H, $\rho(H) \geq 4/9$. It follows that the approximation ratio of algorithm A is exactly $\frac{4}{9} = 0.444\ldots$ This improves the best known ratio for the MAXIMUM PLANAR SUBGRAPH to 4/9.

Next we consider the MAXIMUM GENUS D SUBGRAPH problem: given a connected graph G and an integer $d \geq 0$, find a subgraph H of G of maximum size with an embedding of genus at most d. This is, to our knowledge, the first time this problem is considered in the literature.

This problem is NP-hard. The particular case when $d = 0$ is the MAXIMUM PLANAR SUBGRAPH problem. And, deciding if G is the optimum solution is equivalent to deciding if the genus of G is at most d, which is an NP-complete problem [Th89].

There is a trivial polynomial-time approximation algorithm for the MAXIMUM GENUS D SUBGRAPH problem with performance ratio of 1/6. We describe a new algorithm and prove its performance ratio is at least 1/4.

In section 2, we give a better lower bound on the size of a maximum triangular structure in a planar graph. Also, we conclude that 4/9 is the approximation ratio of algorithm A. In section 3, we describe and analyze the new algorithm for the MAXIMUM GENUS D SUBGRAPH problem.

2 Maximum Triangular Structures in Planar Graphs

A *triangular cactus* is a graph all of whose edges lie in cycles and all cycles are triangles. In this section, we give a better lower bound on the number of triangles in a maximum triangular cactus in a planar graph. Using this lower bound, we prove that $\rho(H) \geq 4/9$, for any planar graph H. Previously, the best lower bound for $\rho(H)$ was 2/5 [CFFK96].

Let G be a graph on n vertices. Let $\mathcal{P} = \{V_1, \ldots, V_k\}$ be a partition of the vertices of G and $\mathcal{Q} = \{E_1, \ldots, E_m\}$ be a partition of the edges of G. For $1 \le i \le m$, let u_i denote the number of classes (from now on, sometimes referred to as *color classes*) V_j of $\mathcal{P}$ met by E_i. We call the ordered pair $(\mathcal{P}, \mathcal{Q})$ *valid for* G if every triangle of G has either at least two vertices in the same block of $\mathcal{P}$ or all three edges in the same block of $\mathcal{Q}$. Set

$$\Phi(\mathcal{P}, \mathcal{Q}) = n - k + \textstyle\sum_{i=1}^{m} \lfloor \frac{u_i - 1}{2} \rfloor. \tag{1}$$

Note that $\Phi(\mathcal{P}, \mathcal{Q}) \ge 0$, and that there is always a valid pair $(\mathcal{P}, \mathcal{Q})$ for G (e.g., $\mathcal{P} = \{V(G)\}, \mathcal{Q} = \{E(G)\}$).

According to Lovász and Plummer [LP86], we have the following theorem:

Theorem 1. *The number of triangles in a maximum triangular cactus in a graph G is equal to the minimum of $\Phi(\mathcal{P}, \mathcal{Q})$ taken over all valid pairs $(\mathcal{P}, \mathcal{Q})$ for G.* [3]

Let G be a planar graph with n vertices. Embed G in the plane without crossing edges, obtaining a plane graph. Let t be the number of edges missing for this embedding to be triangulated. A triangulated plane graph has $3n - 6$ edges, if $n \ge 3$. So $t = (3n - 6) - |E(G)|$, that is, it does not depend on the embedding.

We will prove the following theorem.

Theorem 2. *Let G be a connected planar graph with $n \ge 3$ vertices. Let t be the number of missing edges, defined as above. Then*

$$\Phi(\mathcal{P}, \mathcal{Q}) \ge \frac{1}{3}(n - 2 - t), \quad \textit{for all valid pairs } (\mathcal{P}, \mathcal{Q}) \textit{ for } G. \tag{2}$$

Proof. By induction on the number of edges of G. Let us denote by E_G the edge set of G and by V_G its vertex set.
Basis: $|E_G| \le 2n - 4$.

Then $t \ge (3n - 6) - (2n - 4) = n - 2$, and Theorem 2 holds, since $\Phi(\mathcal{P}, \mathcal{Q}) \ge 0 \ge \frac{1}{3}(n - 2 - t)$.
Inductive step: $|E_G| > 2n - 4$.

Suppose that, for any connected planar graph H on n vertices such that $|E_H| < |E_G|$, Theorem 2 holds for H. Let us prove that theorem 2 holds for G.

We begin by embedding G in the plane without crossing edges, obtaining a plane graph. Next, we augment G to get a triangulated plane graph G'. The edges we add are called *missing edges*. The number of missing edges in G' is t.

Let $(\mathcal{P}, \mathcal{Q})$ be a valid pair for G, with $\mathcal{P} = \{V_1, \ldots, V_k\}$ and $\mathcal{Q} = \{E_1, \ldots, E_m\}$. As before, for $1 \le i \le m$, let u_i denote the number of classes V_j of $\mathcal{P}$ met by E_i. We may assume that if $u_i \le 2$, then $|E_i| = 1$. (If $u_i \le 2$ and $|E_i| > 1$, then we can split E_i into individual edges, obtaining a new edge partition $\mathcal{Q}'$. Any triangle in G with three edges in the old E_i has also at least

[3] Lovász was contacted and agreed that the formula given here, which differs slightly from that in [LP86], is correct.

two vertices of the same color, because $u_i \leq 2$. Therefore $(\mathcal{P}, \mathcal{Q}')$ is also a valid pair for G. And, moreover, $\Phi(\mathcal{P}, \mathcal{Q}') = \Phi(\mathcal{P}, \mathcal{Q})$, because $\lfloor \frac{u-1}{2} \rfloor = 0$ when $u \leq 2$.)

Assume that for $1 \leq i \leq q$, $|V_i| = p_i \geq 2$ and for $q < i \leq k$, $|V_i| = 1$. The last $k - q$ blocks are singletons. Let us call a vertex x a *singleton* if $\{x\} \in \mathcal{P}$. Let $s = k - q$ be the number of singletons and $p = \sum_{i=1}^{q} p_i$ be the number of vertices which are not singletons. So

$$n = p + s \quad \text{and} \quad k = q + s. \tag{3}$$

Observe that $\mathcal{P}$ is a partition of the vertices of G' (since G and G' have the same vertex set), $\mathcal{Q}$ is a partition of the edges of G (not of G'), and for any triangle of G' (not of G), facial or not, at least one of the following three conditions holds.
(1) At least two of its vertices are of the same color. We say that the triangle *is covered by a color.*
(2) All three edges are in the same class E_i of $\mathcal{Q}$. We say that the triangle *is covered by an edge class.*
(3) One or more of its edges is a missing edge. This corresponds to the fact that this triangle does not exist in G. We say that the triangle *is covered by a missing edge.*

A *facial triangle* is a triangle in G' which is the boundary of some face of G'. A facial triangle T of G' is a *facial triangle neighboring vertex v* if it contains v.

Let us partition the set of singletons into three sets A, B, C, according to how the facial triangles neighboring these singletons are covered.

A: The set of singletons all of whose neighbors are of the same color.
B: The set of singletons x not in A such that there is an edge class which covers one of the facial triangles neighboring x.
C: The set of all singletons not in $A \cup B$.

Notice that all the facial triangles neighboring a singleton in A (even the ones containing missing edges) are covered by a color. Also not all facial triangles neighboring a singleton in C are covered by a color class (otherwise, all of its neighbors would have the same color and this singleton would be in A). And no facial triangle neighboring a singleton in C is covered by an edge class (otherwise, this singleton would be in set B).

Let a, b and c be the sizes of A, B and C, respectively. Observe that s, the number of singletons, satisfies

$$s = a + b + c. \tag{4}$$

For each singleton x in B, we choose an i such that E_i covers a facial triangle neighboring x. We denote by s_i the number of singletons x which have chosen E_i. Observe that E_i meets all singletons which have chosen E_i; therefore

$$s_i \leq u_i. \tag{5}$$

Moreover, if $|E_i| = 1$, then $s_i = 0$, because E_i cannot cover any triangle.

Case 1: There is an i such that $u_i = s_i = 4$.

This means that there are four distinct singletons x, y, w, z in B which are the only vertices E_i meets. Each of x, y, w, z has chosen E_i, implying that E_i must cover a facial triangle neighboring x. This facial triangle meets only a subset of these four vertices. Without loss of generality, we may assume this facial triangle is xyw. E_i also covers a facial triangle neighboring z. Also without loss of generality, we may assume it is ywz. Then $\{xy, yw, xw, yz, wz\} \subseteq E_i \subseteq \{xy, yw, xw, yz, wz, xz\}$.

Let G_1 be G after we remove the edges yw and xz (if edge xz exists). From the pair $(\mathcal{P}, \mathcal{Q})$, which is valid for G, we obtain a pair $(\mathcal{P}, \mathcal{Q}_1)$ which is valid for G_1 by taking $\mathcal{Q}_1 = \mathcal{Q}$, except that the i^{th} block of $\mathcal{Q}_1$ is $E_i - \{yw, xz\}$ split into individual edges (and otherwise the partitions are the same). Note that this pair $(\mathcal{P}, \mathcal{Q}_1)$ is indeed valid for G_1: each triangle in G_1 is covered exactly as it was covered in G (the triangles covered by class E_i do not exist in G_1).

Observe that G_1 is connected, and so it is a connected planar graph. Moreover, it has fewer edges than G. Thus, we can apply induction, and conclude that $\Phi(\mathcal{P}, \mathcal{Q}_1) \geq \frac{1}{3}(n - 2 - t_1)$, where t_1 is the number of missing edges for G_1 to be triangulated. Because we removed at most two edges from G to get G_1, $t_1 \leq t + 2$.

The new classes in $\mathcal{Q}_1$ have zero contribution to $\Phi(\mathcal{P}, \mathcal{Q}_1)$ (since any edge class of size one contributes zero), and, since the contribution of E_i to $\Phi(\mathcal{P}, \mathcal{Q})$ is $\lfloor \frac{4-1}{2} \rfloor = 1$, it follows that $\Phi(\mathcal{P}, \mathcal{Q}_1) = \Phi(\mathcal{P}, \mathcal{Q}) - 1$.

Putting all this together, we have that

$$\Phi(\mathcal{P}, \mathcal{Q}) = \Phi(\mathcal{P}, \mathcal{Q}_1) + 1 \geq \frac{1}{3}(n-2-t_1)+1 \geq \frac{1}{3}(n-2-(t+2))+1 \geq \frac{1}{3}(n-2-t).$$

Case 2: There is no i such that $u_i = s_i = 4$.

Consider pairs (j, F), where $1 \leq j \leq q$ and F is a face of $G'[V_j]$, the plane subgraph of G' induced by the vertex set V_j. Let c' be the number of such pairs (j, F) where some vertex not in A is embedded in F. Call each of these c' pairs *special.*

Recall that p is the number of vertices which are not singletons, q is the number of color classes with more than one vertex, t is the number of missing edges in G', and a, b, c are the numbers of singletons in A, B, C, respectively.

The next lemmas give upper bounds on a, b and c, which will be needed to complete the proof of Theorem 2.

Lemma 3. $a \leq 2p - 3q - c'$.

Proof. Each singleton x in A has all neighbors of some color $j \leq q$. Clearly x is embedded in a face of $G'[V_G - \{x\}]$. Because all neighbors of x are in V_j, this face F is also a face of $G'[V_j]$, and there cannot be any other vertex embedded in F. Therefore the size a of A is at most the number of pairs (j, F) which are not special.

The maximum number of pairs (j, F), where F is a face of $G'[V_j]$, is $2p_j - 3$ (if $p_j = 2$ then it has only one face; if $p_j > 2$ then it has the maximum number of faces when it is triangulated, and in this case, it has $2p_j - 4$ faces). Therefore there are at most $\sum_{j=1}^{q}(2p_j - 3) = 2p - 3q$ pairs (j, F), where $j \leq q$ and F is a face of $G'[V_j]$. From these pairs (j, F), c' of them (the special ones) have a vertex not in A embedded in F. Thus the number of pairs (j, F) which are not special is at most $2p - 3q - c'$, and therefore $a \leq 2p - 3q - c'$. □

Lemma 4. $b \leq 3\sum_{i=1}^{m} \lfloor \frac{u_i - 1}{2} \rfloor$.

Proof. If we prove that $s_i \leq 3\lfloor \frac{u_i-1}{2} \rfloor$, then by summing $s_i \leq 3\lfloor \frac{u_i-1}{2} \rfloor$ over $i \in \{1, 2, ..., m\}$, the lemma follows, since $b = \sum_{i=1}^{m} s_i$.

First, if $u_i \leq 2$, then $|E_i| = 1$, which implies that $s_i = 0$. And, so $s_i = 0 = 3\lfloor \frac{u_i-1}{2} \rfloor$. If $u_i > 2$, then by equation (5), we know that $s_i \leq u_i$. As a consequence, $s_i \leq 3\lfloor \frac{u_i-1}{2} \rfloor$ can only be false if $u_i = s_i = 4$. But this does not happen because we are given that for no i does $u_i = s_i = 4$. □

Lemma 5. $c \leq t + c'$.

Proof. A missing edge covers at most two facial triangles. Therefore, it suffices to prove the existence of at least $2c - 2c'$ facial triangles which must be covered by missing edges.

Let us associate a set of facial triangles with each component of $G'[C]$. More specifically, for a component of $G'[C]$ whose vertex set is D, let us associate a set of either $2|D|$ or $2|D| - 2$ facial triangles, each with at least one vertex in D, such that each such facial triangle must be covered by a missing edge. We will make sure that each of these facial triangles will have its three vertices in different color classes and at least one of them is in C. (The triangle is not covered by a color class and, because of the vertex in C, it is not covered by an edge class. So, it has to be covered by a missing edge.)

Observe that the sets of facial triangles corresponding to two different components of $G'[C]$ are disjoint (since there is no edge between two different components of $G'[C]$, there cannot be a triangle with vertices in two different components of $G'[C]$). If at most c' of the components are associated with a set with $2|D| - 2$ facial triangles, then in total we must have at least $2c - 2c'$ facial triangles which must be covered by missing edges (since $\sum_D |D| = c$). This would complete the proof of Lemma 5.

Let D be the vertex set of a component of $G'[C]$. A *relevant facial triangle for D* is a facial triangle of G' with at least two vertices in D. Let f_D be the number of faces of $G'[D]$ (which is a connected graph), f_3 be the number of faces of $G'[D]$ which are also faces of G', and e_D be the number of edges in $G'[D]$.

Because $|E_G| > 2n - 4$, there are at most $(3n - 6) - (2n - 4 + 1) = n - 3$ missing edges. A missing edge covers at most two facial triangles. So there are at most $2n - 6$ facial triangles covered by missing edges. But in G' there are $2n - 4$ facial triangles (all the faces of G'). Thus, there are facial triangles covered by colors or by edge classes. If there are facial triangles covered by colors, then some vertices have the same color. This means that there are vertices which are not singletons, implying $C \neq V_G$. If all vertices are singletons and a facial triangle is covered by an edge class, then the singletons neighboring this facial triangle are in B. It follows that $C \neq V_G$.

Embedded in the faces of $G'[D]$ are all the vertices of $V_G - D$. Now $V_G - D \supseteq V_G - C \neq \emptyset$. Thus there is at least one face of $G'[D]$ which contains some vertex of $V_G - D \neq \emptyset$. This face of $G'[D]$ is not a face of G', since no face of G' contains any vertex. Thus $f_3 < f_D$.

Claim 6 *If $f_3 = f_D - 1$ then there are at least $2|D| - 2$ relevant facial triangles for D. If $f_3 \leq f_D - 2$ then there are at least $2|D|$ relevant facial triangles for D.*

Proof. A relevant facial triangle for D has at least two vertices in D, which means it contains an edge e in $G'[D]$. But e is contained in exactly two facial triangles of G'. These are the only two relevant facial triangles for D that contain e. This would give us $2e_D$ of the desired relevant facial triangles for D, except that not all of them are distinct. If a facial triangle is counted by two edges, then all three of its vertices are in D, which implies it is a face of $G'[D]$ (and, being a facial triangle, also of G'). Moreover, it was counted exactly three times (once for each of its edges). Therefore we have $2e_D - 2f_3$ relevant facial triangles for D. Now, if $f_3 = f_D - 1$, applying Euler's formula, we get $2e_D - 2f_3 = 2e_D - 2(f_D - 1) = 2(|D| - 2) + 2 = 2|D| - 2$ relevant facial triangles. If $f_3 \leq f_D - 2$, we get $2e_D - 2f_3 \geq 2e_D - 2(f_D - 2) = 2(|D| - 2) + 4 = 2|D|$. □

Include these $2|D|$ (if possible) or $2|D| - 2$ relevant facial triangles in the set of facial triangles corresponding to $G'[D]$.

To guarantee that at most c' of the components of $G'[C]$ have only $2|D| - 2$ facial triangles in their corresponding sets, we will need to add two more facial triangles to some of the sets that currently have only $2|D| - 2$. Now let D be the vertex set of a component of $G'[C]$ such that all the faces of $G'[D]$ but one are faces of G', that is, $f_3 = f_D - 1$. (Any component induced by a set D' with $f_3 \leq f_{D'} - 2$ already has $2|D'|$ associated facial triangles.)

Since all the faces of $G'[D]$ but one are faces of G', by the previous paragraph, all the vertices in $V_G - D$ must be embedded in the same face of $G'[D]$ (that face is the only face which is not a face of G').

Claim 7 *Let G' be a triangulated plane graph with vertex set V. Let D be a (non-empty) set of vertices of G' such that the subgraph of G' induced by D, $G'[D]$, is connected, and all vertices in $V - D$ are embedded in the same face of $G'[D]$. Then $G'[V - D]$ is connected.*

For this extended abstract, we omit the proof of this claim, and also the proof of next claim.

Because $G'[D]$ is connected, all vertices of D are embedded in the same face of $G'[V_G - D]$. Call this face F'.

Claim 8 *One of the following holds:*
(a) *All the vertices on the boundary of F' are of the same color $j \leq q$. In this case, F' is a face of $G'[V_j]$.*

(b) *There are two distinct facial triangles of G', each with exactly one vertex in D and the other two vertices of different colors.*

If (b) holds, add the two extra facial triangles to the set of facial triangles corresponding to $G'[D]$. Note that they are different from the ones already in the set because they contain exactly one vertex of D and all the others contain at least two vertices of D. Each set D for which (b) holds is now associated with a set of at least $2|D|$ facial triangles.

For how many components of $G'[C]$ can (a) hold? Let us show that each component of $G'[C]$ for which (a) holds corresponds to one of the c' special pairs, and that different components correspond to different special pairs.

Let D be the vertex set of a component of $G'[C]$ for which (a) holds. Let us prove that there is a special pair (j, F') such that the only vertices of C embedded in F' are the vertices in D. Let F' be the face of $G'[V_G - D]$ in which the vertices of D are embedded. Recall that $G'[V_G - D]$ is connected. Because (a) holds for this component, F' is a face of $G'[V_j]$. Since the vertices of D are embedded in F', and vertices of D are not in A, some vertex not in A is embedded in F'; this makes the pair (j, F') special. The only vertices of G' embedded in F' are in D (because F' is a face of $G'[V_G - D]$). Since the only vertices of C embedded in F' are the vertices in D, no other component of $G'[C]$ (with a different set D) will correspond to this pair (j, F').

So the number of components such that (a) holds is at most the number c' of special pairs. Thus there are at most c' components of $G'[C]$ such that (a) holds, which completes the proof of Lemma 5, by the discussion in the third paragraph of the proof. □

Using (1) and the upper bounds given by the previous lemmas, we conclude the proof of Theorem 2 as follows:

$$\begin{aligned}
\Phi(\mathcal{P}, \mathcal{Q}) &= n - k + \textstyle\sum_{i=1}^{m} \lfloor \frac{u_i - 1}{2} \rfloor, \text{ by (1)},\\
&= p - q + \textstyle\sum_{i=1}^{m} \lfloor \frac{u_i - 1}{2} \rfloor, \text{ by (3)},\\
&\geq p - q + b/3, \text{ by Lemma 4},\\
&= p - q + (s - a - c)/3, \text{ by equation (4)},\\
&\geq p - q + \frac{s - (2p - 3q - c') - (t + c')}{3}, \text{ by Lemmas 3 and 5},\\
&= (p + s)/3 - t/3 = n/3 - t/3, \text{ by the first equation in (3)},\\
&\geq (n - 2 - t)/3.
\end{aligned}$$

□

Corollary 9. *If H is a plane graph, then $\rho(H) \geq \frac{4}{9}$.*

Proof. We may assume H is connected and has at least three vertices. Let t be the number of missing edges for H to be triangulated. By Euler's formula, $|E(H)| = 3n - 6 - t$, where $n \geq 3$ is the number of vertices of H.

A maximum triangular structure in H can be obtained by extending a maximum triangular cactus to a connected graph (by adding edges without forming any new cycles). Also, a maximum triangular structure in H has one more edge per triangle than a spanning tree of H.

From theorems 1 and 2, the number of triangles in a maximum triangular cactus is at least $\frac{1}{3}(n-2-t)$. From this, we conclude that $mts(H) \geq n-1+\frac{1}{3}(n-2-t)$, and then

$$\rho(H) \geq \frac{n-1+\frac{1}{3}(n-2-t)}{3n-6-t} = \frac{4n-5-t}{9n-18-3t} \geq \frac{4}{9}, \quad \text{for all } t \geq 0. \qquad \square$$

Recall, the output of algorithm A is a maximum triangular structure in the input graph.

Corollary 10. *The performance ratio of algorithm A is $\frac{4}{9}$.*

3 Subgraphs of a Given Genus

In this section, we describe and analyze an algorithm for finding a large subgraph (of a given graph) with an embedding of genus at most d.

Without loss of generality, we may assume that the input graph G has at least $n-1+d$ edges. Recall that G is connected. So, consider P a connected spanning subgraph of G with $n-1+d$ edges. It is easy to find an embedding of P of genus d: a spanning tree of P is planar, and adding one edge increases the genus of the embedding by at most one. One can think of this as putting a new handle for each of the d edges we add. Since, by Euler's formula, $Opt(G) \leq 3n+6d-6$, we have $|E(P)| \geq \frac{1}{6}Opt(G)$. It follows that achieving an approximation ratio of 1/6 is trivial.

Notice that for the above P to have a ratio of exactly 1/6, the embedding of genus d of an optimum solution H must be almost triangulated (that is, almost all the faces have length three). Therefore G has many triangles, which we can use as follows: if P is a connected graph with an embedding of genus g, and T is a triangle with all vertices in P, then the graph $(V(P), E(P) \cup E(T))$ has an embedding of genus at most $g+2$.

We hope to embed three more edges, but increase the genus by only two. A detailed analysis shows that, using this idea, we can develop an algorithm with approximation ratio of 1/4. Using more complicated graphs instead of triangles, one can obtain a more complicated algorithm with approximation ratio of $\frac{1}{4}+\epsilon$, for some small positive ϵ.

For simplicity, we shall only give a high level description of the algorithm.

Algorithm B

Step 1: Start with a plane embedding of P, a spanning tree of the input graph G.

Step 2: Repeatedly (as long as possible, and while the genus of the resulting embedding of P is at most d) find a triangle T of G which has at least two edges not in P. Add the edges of the triangle (not in P) to P, modifying the embedding

of P, so that its genus increases by at most one if two edges are added, or at most two if all three edges of T are added.
Step 3: Repeatedly (as long as possible, and while the genus of the resulting embedding of P is at most d) add an edge of $E(G) \setminus E(P)$ to P, modifying the embedding of P, so that its genus increases by at most one.
Step 4: Output P and the embedding of P of genus at most d.

Note that the algorithm can be implemented in polynomial time, since one can explicitly list all the triangles of G in time $|E(G)|^{\frac{3}{2}}$ [CN85]. Also steps 2 and 3 iterate at most $|E(G)|$ times.

Theorem 11. *The approximation ratio of algorithm B is $\frac{1}{4}$.*

We omit the proof of this theorem by lack of space.

4 Open Problems

To our knowledge, nothing is known about a polynomial-time algorithm with constant approximation ratio for finding the genus of a given graph.

5 Acknowledgments

The authors would like to thank Howard Karloff and Robin Thomas for useful discussions and suggestions.

References

[CFFK96] G. Călinescu, C. G. Fernandes, U. Finkler and H. Karloff, "A Better Approximation Algorithm for Finding Planar Subgraphs", *Proc. 7th Annual ACM-SIAM Symp. on Discrete Algorithms*, 1996.

[CN85] N. Chiba and T. Nishizeki, "Arboricity and Subgraph Listing Algorithms", *SIAM Journal of Computing*, 14:210-223, 1985.

[F92] L. R. Foulds, *Graph Theory Applications*, Springer-Verlag, New York, 1992.

[GS85] H. N. Gabow and M. Stallmann, "Efficient Algorithms for Graphic Matroid Intersection and Parity", *Automata, Language and Programming: 12th Colloq., Lecture Notes in Computer Science*, Vol. 194, 210–220, 1985.

[LG77] P. C. Liu and R. C. Geldmacher, "On the Deletion of Nonplanar Edges of a Graph", *Proc. 10th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, 727–738, 1977.

[LP86] L. Lovász and M. D. Plummer, *Matching Theory*, Elsevier Science, Amsterdam, 1986.

[TDB88] R. Tamassia, G. Di Battista and C. Batini, "Automatic Graph Drawing and Readability of Diagrams", *IEEE Transactions on Systems, Man and Cybernetics*, 18:61-79, 1988.

[Th89] C. Thomassen, "The Graph Genus Problem is NP-Complete", *Journal of Algorithms*, 10:568-576, 1989.

This article was processed using the LaTeX macro package with LLNCS style

Multicuts in Unweighted Digraphs with Bounded Degree and Bounded Tree-Width

Gruia Călinescu [*] Cristina G. Fernandes [†]

Abstract

The Directed Multicut problem can be defined as: given a digraph D and a collection of k pairs of distinct vertices (s_i, t_i) of D, find a minimum set of edges whose removal ensures that none of the strongly connected components includes one of the k pairs of vertices. We restrict our attention to bounded-degree and bounded-tree-width digraphs (the directed version of tree-width) and present some bad and good news. Bad news: Directed Multicut is still NP-hard in unweighted digraphs with tree-width one and bounded degree. Good news: there is a polynomial-time approximation scheme (PTAS) for Directed Multicut in unweighted digraphs with bounded tree-width and bounded degree, that is, for any $\epsilon > 0$, there is a polynomial-time $(1+\epsilon)$-approximation algorithm.

1 Introduction

We consider the **Directed Multicut** problem, introduced by Klein, Plotkin, Rao and Tardos [3]: given a digraph D and a collection of k pairs of distinct vertices (s_i, t_i) of D, find a minimum set of edges whose removal ensures that none of the strongly connected components includes one of the k pairs of vertices. In other words, for all i, either all paths from s_i to t_i *or* all paths from t_i to s_i must be broken.

A particular case of this problem is the **Feedback Arc Set** problem: given a digraph D, find a minimum set of arcs of D whose removal results in an acyclic digraph. This problem is known to be NP-hard [2], therefore so it is the Directed Multicut. We thus search for efficient approximation algorithms. The *performance ratio of an approximation algorithm A* for a minimization problem is the supremum, over all possible instances I, of the ratio between the size of the output of A when running on I and the size of an optimal solution for I. We say A is an *α-approximation algorithm* if its performance ratio is at most α. The smaller the performance ratio, the better.

Klein et al. [3] presented a $O(\log^2 k)$-approximation algorithm for edge-weighted Directed Multicut in general digraphs. In this paper we extend the results in [1] to digraphs. We give a PTAS for Directed Multicut in unweighted digraphs with bounded degree and bounded tree-width. It is worth mentioning that we mean here the concept of tree-width in digraphs, introduced by Johnson, Robertson, Seymour, and Thomas [4], and which differs from the one in undirected graphs. For example, the class of digraphs of tree-width zero consists of all acyclic digraphs. We will present the formal definition of tree-width in the next section.

As in [1], we find useful a variation of the Directed Multicut problem. The **Directed Unrestricted Vertex Multicut** problem is: given a digraph D and a collection of k pairs of vertices (s_i, t_i) of D called *terminals*, find a minimum set of vertices whose removal ensures that none of the strongly connected components includes one of the k pairs of vertices. (Note that terminals might be removed.)

We prove that Directed Multicut is NP-hard in digraphs of tree-width one and bounded degree and that Directed Unrestricted Vertex Multicut is NP-hard in tree-width two and bounded-degree digraphs.

[*]Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616. Research partially done while at Georgia Institute of Technology and supported in part by NSF grant CCR-9319106. E-mail: calinesc@cs.iit.edu.

[†]Department of Computer Science, University of São Paulo, Brazil. E-mail: cris@ime.usp.br. Research partially done while at Georgia Institute of Technology and supported in part by NSF grant CCR-9319106, FAPESP (Proc. 96/04505–2) and PRONEX (MCT/FINEP)(Proj. 107/97).

Our PTAS for Directed Multicut in unweighted digraphs with bounded degree and bounded tree-width follows very much the idea of a PTAS for the undirected case [1]. We first give a straightforward PTAS for Directed Unrestricted Vertex Multicut in digraphs of bounded tree-width. Then we present an approximation-ratio preserving reduction from Directed Multicut to Directed Unrestricted Vertex Multicut. If the Directed Multicut instance digraph has bounded degree and bounded tree-width, the Directed Unrestricted Vertex Multicut instance obtained by the reduction has bounded tree-width. (This reduction is very different from the undirected reduction in [1].) Combining the reduction with the PTAS for Directed Unrestricted Vertex Multicut in digraphs of bounded tree-width, we obtain a PTAS for Directed Multicut in digraphs with bounded degree and bounded tree-width.

2 Tree-Width and Multicuts in Digraphs

Johnson et al. [4] presented the following definition of tree-width in digraphs.

An *arborescence* is a digraph R such that R has a vertex r_0, called the root of R, with the property that for every vertex $r \in V(R)$ there is a unique directed walk from r_0 to r. Thus every arborescence arises from a tree by selecting a root, and directing all edges away from the root. If $r, r' \in V(R)$, we write $r' > r$ if $r' \neq r$ and there exists a directed walk in R with initial vertex r and terminal vertex r'. If $e \in E(R)$, we write $r' > e$ if either $r' = r$ or $r' > r$, where r is the head of e. We also write $e \sim r$ when e is incident with r.

Let D be a digraph and let $Z \subseteq V(D)$. The digraph obtained from D by deleting Z will be denoted by $D \setminus Z$. We say that a set $S \subseteq V(D)$ is *Z-normal* if there is no directed walk in $D \setminus Z$ with first and last vertex in S that uses a vertex of $D \setminus (Z \cup S)$.

An *arboreal decomposition of a digraph* D is a triple (R, X, W), where R is an arborescence, and $X = (X_e \subseteq V(D) : e \in E(R))$ and $W = (W_r \subseteq V(D) : r \in V(R))$ satisfy

(1) $(W_r : r \in V(R))$ is a partition of $V(D)$, and

(2) if $e \in E(R)$, then $\cup\{W_r : r \in V(R), r > e\}$ is X_e-normal.

The *width of* (R, X, W) is the least integer w such that $|\cup_{e \sim r} X_e \cup W_r| \leq w + 1$ for all $r \in V(R)$. The *tree-width of* D, denoted by $dtw(D)$, is the least integer w such that D has an arboreal decomposition of width w.

Next we describe a PTAS for Unrestricted Vertex Multicut in digraphs with bounded tree-width. This PTAS is very similar to the one for undirected graphs in [1]. Recall that a PTAS consists of, for each $\epsilon > 0$, a polynomial-time algorithm for the problem with a performance ratio of at most $1 + \epsilon$. Let us describe such an algorithm.

The input of the algorithm is a digraph D, an arboreal decomposition $\Theta = (R, X, W)$ of D, and a set $\mathcal{C}$ of pairs of vertices of D. Given a subdigraph D' of D, denote by $\mathcal{C}(D')$ the set of pairs in $\mathcal{C}$ whose two vertices are in D', and by $D \setminus D'$ the subdigraph of D induced by $V(D) \setminus V(D')$. For the description of the algorithm, all the instances we mention are on a subdigraph D' of D and the set of pairs to be disconnected is $\mathcal{C}(D')$. So we will drop $\mathcal{C}(D')$ of the notation and refer to an instance only by the digraph D'. Denote by $opt(D')$ the size (i.e., the number of vertices) of an optimal solution for D'.

Consider R rooted at r_0 and an arbitrary ordering of the children of each vertex of R. For a vertex u of R, let $R(u)$ be the subarborecence of R rooted at u. Let $D(u)$ be the subdigraph of D induced by the set $Z = \cup\{W_{r'} : r' \in V(R), r' \geq u\}$. Let $t = \lceil (dtw(D) + 1)/\epsilon \rceil$.

In this abstract, we give only a general description of the algorithm: label the vertices of R in postorder. Find the lowest labeled vertex r such that an optimal solution for $D(r)$ has at least t vertices. If there is no such vertex, let r be the root of R. Let e be the edge of R entering r (if r is the root, let e undefined). Using the already computed optimal solutions for the children of r, find an approximate solution S_r for $D(r)$ such that $|S_r| \leq (1+\epsilon)opt(D(r))$ and $X_e \cup W_r \subseteq S_r$. If r is the root of R, then output S_r. Otherwise, let $D' = D \setminus (D(r) \cup X_e)$ and let $\Theta' = (R', X', W')$ be the arboreal decomposition of D' where $R' = R \setminus R(r)$ and $X'_f = X_f \setminus (X_e \cup V(D(r)))$, for all $f \in E(R')$, and $W'_s = W_s \setminus X_e$, for all $s \in V(R')$. Recursively get a solution S' for D'. Output $S = S' \cup S_r$.

The set S produced is a solution for the Directed Unrestricted Vertex Multicut in D. Indeed if C is a directed circuit either entirely in $D(r) \cup X_e$ or entirely in D' and containing s and t for some $(s,t) \in \mathcal{C}$, then $V(C) \cap S \neq \emptyset$ (C either intersects S_r or S'). If C is a directed circuit with vertices in $D(r)$ (that is, in Z) and in D', containing s and t for some $(s,t) \in \mathcal{C}$, then C has a vertex in $X_e \subseteq S_r \subseteq S$ because Z is X_e-normal. Besides, the set S has size at most $(1+\epsilon)opt(D)$.

3 Reduction from Directed Multicut to Unrestricted Vertex Multicut

Now we present an approximation-ratio preserving reduction from Directed Multicut to Directed Unrestricted Vertex Multicut. The reduction is very different from the undirected reduction in [1]. Consider an instance of Directed Multicut, that is, a digraph $D = (V, E)$ and a set $\mathcal{C}$ of pairs of distinct vertices of D. Let us describe the corresponding instance of Directed Unrestricted Vertex Multicut.

The input graph for Directed Unrestricted Vertex Multicut is the following digraph D'. (To avoid confusion, we will refer to vertices of D' as nodes and to edges of D' as arcs.) The set of nodes of D' is E plus m copies $u_1, \ldots, u_m$ of each vertex $u \in V$, where m is the number of edges in D. There is an arc in D' from a copy of $u \in V$ to $e \in E$ if u is the tail of e, and there is an arc in D' from $e \in E$ to a copy of $u \in V$ if u is the head of e. These are all the arcs in D'. Note that D' is a bipartite digraph (edges of D in one side and copies of vertices of D in the other).

Now let us describe the set of pairs of vertices of D'. For each pair (s,t) in $\mathcal{C}$, we have in $\mathcal{C}'$ pairs (s_i, t_j) for all $1 \leq i, j \leq m$. Clearly D' can be obtained from D in polynomial time. Note that $\mathcal{C}'$ has km^2 pairs. Also $\mathcal{C}'$ can be obtained from D and $\mathcal{C}$ in polynomial time.

Lemma 3.1 *For any solution S' for the Directed Unrestricted Vertex Multicut in D' there is a solution $S \subseteq E(D)$ such that $|S| \leq |S'|$.*

Proof. We may assume S' is a minimal solution for the Directed Unrestricted Vertex Multicut in D'. If $S' \subseteq E(D)$ then we take $S := S'$. If not, let $u \in V(D)$ and $1 \leq i \leq m$ be such that $u_i \in S'$. Then $S' - \{u_i\}$ is not a solution in D', which implies that there is a pair $(s,t) \in \mathcal{C}'$ and a directed circuit Q in $D' \setminus (S' - \{u_i\})$ containing s and t (because s and t are in the same strongly connected component). If $u_j \notin S'$ for some $1 \leq j \leq m$, the circuit obtained from Q by substituting u_i by u_j is a directed circuit in $D' \setminus S'$ and contains a pair in $\mathcal{C}'$, a contradiction. Thus $u_j \in S'$ for all $1 \leq j \leq m$. In this case, $|S'| \geq m = |E(D)|$ and we take $S := E(D)$. In both cases S is a solution in D' and $|S| \leq |S'|$, which completes the proof of the lemma. ■

Lemma 3.1 together with the following theorem, whose proof is similar to the proof of Theorem 4 in [1], completes the reduction.

Theorem 3.2 *$S \subseteq E(D)$ is a solution for Directed Multicut in D if and only if S is a solution for Directed Unrestricted Vertex Multicut in D'.*

The next lemma shows that, if the instance of Directed Multicut is a digraph with bounded degree and bounded tree-width, then the corresponding instance of Directed Unrestricted Vertex Multicut has bounded tree-width. For this, we need some notation. For a vertex v in a digraph D, we denote by $\delta^+_D(v)$ the set of edges in D with v as tail. For a set S of vertices of D, $\delta^+_D(S)$ is the set of edges with tail in S and head not in S.

Lemma 3.3 *If D has bounded degree and bounded tree-width, then D' has bounded tree-width.*

Proof. Let D be a digraph with bounded degree and bounded tree-width and D' be obtained from the reduction described above. Let us present an arboreal decomposition of D' whose tree-width is at most

$(dtw(D)+1)(\Delta+1)$, where Δ is the maximum outdegree of D.

Let $\Theta = (R, X, W)$ be an arboreal decomposition of D of width $dtw(D)$. Let R' be an arborescence obtained from R by substituting each edge $e \in E(R)$ from a vertex r' to a vertex r in R by a directed path $P_e = \langle u_0, \ldots, u_m \rangle$ of length m, starting at $u_0 = r'$ and ending at $u_m = r$, where $u_1, \ldots, u_{m-1}$ are new vertices. Let f be the edge in $E(P_e)$ from u_{i-1} to u_i, $1 \le i \le m$. If $i = 1$, then let $Y_f := \delta^+_D(X_e)$, else let $Y_f := \delta^+_D(X_e \cup W_r)$. For $i = 1, \ldots, m$, let $W^i_r \subseteq V(D')$ be the set containing the i^{th} copy in D' of each vertex in W_r. Set $Z_{u_1} := W^1_r \cup \delta^+_D(W_r)$ and, for $i = 2, \ldots, m$, set $Z_{u_i} := W^i_r$. Let us prove that $\Theta' = (R', Y, Z)$ is an arboreal decomposition of D'.

First note that $(Z_u : u \in V(R'))$ is a partition of $V(D')$. Now let f be an edge of R'. We need to show that $Z' := \cup\{Z_u : u \in V(R'), u > f\}$ is Y_f-normal. Let $e \in E(R)$ be such that $f \in E(P_e)$. Let $x, y \in Z'$ be such that there is a directed path Q' in D' from x to y which uses a node in $V(D') - Z'$. Choose $x, y \in Z'$ so that Q' has minimum length. We need to show that $V(Q') \cap Y_f \neq \emptyset$. We analyze two cases.

If $x \notin E(D)$ then let $x' \in V(D')$ be the node following x in Q'. Note that $x' \in E(D) - Z'$, because of the bipartition of D' and the choice of x and y. This implies that f is not the first edge of the path P_e. Thus x is a copy of a vertex in W_r and $x' \in \delta^+_D(W_r) \subseteq Y_f$.

If $x \in E(D)$ then let Q be the path in D which starts at the tail of x and visits the vertices of D in the same order in which they (their copies in fact) appear in Q'. Let $Z \subseteq V(D)$ be the set of vertices with at least one copy in Z'. Note that $Z = \cup\{W_r : r \in V(R), r > e\}$. Both endpoints of Q are in Z. Indeed, the tail of any edge of D in Z' is also in Z' (because, by construction, if $a \in E(D) \cap Z_u$ then one copy of its tail is also in Z_u) so the first vertex of Q is in Z. Also the last vertex of Q' is not in $E(D)$ (for the same reason), and this implies that the last vertex of Q is in Z. Thus Q is a path in D between two vertices of Z. Since Θ is an arboreal decomposition of D, there is a vertex of X_e in Q. But this implies Q' has a node in Y_f, because $Y_f \supseteq \delta^+_D(X_e)$. This completes the proof that Θ' is an arboreal decomposition of D'.

Now we just need to verify that the width of Θ' is at most $(dtw(D)+1)(\Delta+1)$. For that, let $r' \in V(R')$. If r' is a vertex of R, then let $r = r'$, otherwise, r' was added when an edge e of R was substituted by a path between its endpoints. Let r be the tail of e. Note that $| \cup_{f \sim r'} Y_f \cup Z_{r'}| \le |(\cup_{e \sim r} X_e \cup W_r) \cup \delta^+_D(\cup_{e \sim r} X_e \cup W_r)| \le (dtw(D)+1)(\Delta+1)$. ■

Combining the reduction described above and the PTAS from the previous section, we obtain a PTAS for Directed Unweighted Multicut in digraphs with bounded degree and bounded tree-width.

4 Complexity results

A *bidirected tree* is a digraph which has as underlying graph a tree and two opposite directed edges for each edge in its underlying graph. Similarly, a *bidirected series-parallel digraph* has as underlying graph a series-parallel graph, and two opposite directed edges for each edge in its underlying graph. Bidirected trees have tree-width one and bidirected series-parallel digraphs have tree-width two, by Proposition 2.1 in [4]. By a simple modification of the proofs for the undirected case [1, Theorems 7 and 10], we can prove the following.

Theorem 4.1 *Directed Multicut and Directed Unrestricted Vertex Multicut are NP-hard respectively in bidirected trees and bidirected series-parallel digraphs with maximum in and out degree three.*

The modification consists of using, in each of the reductions, a digraph whose underlying graph is exactly the one described in the corresponding undirected case. Besides, the digraph has two opposite directed edges for each edge in its underlying graph. Note that any directed path in this digraph corresponds to a path in its underlying graph and *vice-versa*. For the edge version, we also note that any feasible solution can be converted in a feasible solution with edges directed away from the root, without increasing the size of the solution, by simply replacing any edge directed towards the root by the parallel edge directed away from the root. The reductions work exactly as in the undirected case, so we omit them in this abstract.

5 Acknowledgments

We would like to acknowledge the help of Robin Thomas in understanding the directed tree-width concept.

References

[1] G. Călinescu, C.G. Fernandes and B. Reed, "Multicuts in Unweighted Graphs with Bounded Degree and Bounded Tree-Width." In *Integer Programming and Combinatorial Optimization* (Berlin, 1998), R. E. Bixby and E. A. Boyd and R. Z. Ríos-Mercado, Eds., v.1412 of *Lecture Notes in Computer Science.* Springer, pp. 137–152.

[2] G. Even, J.S. Naor, B. Schieber and M. Sudan, "Approximating Minimum Feedback Sets and Multicuts in Directed Graphs," *Proc. 4th International Conference on Integer Programming and Combinatorial Optimization* (1995), Lecture Notes in Computer Science 920, Springer, pp. 14–18.

[3] P. Klein, S. Plotkin, S. Rao and E. Tardos, "Approximation Algorithms for Steiner and Directed Multicuts," *Journal of Algorithms,* 22 (2), 241-269, 1997.

[4] T. Johnson, N. Robertson, P.D. Seymour and R. Thomas, "Directed Tree-Width," *preprint* (available at `http://www.math.gatech.edu/~thomas/`).

JOURNAL OF ALGORITHMS **27**, 269–302 (1998)
ARTICLE NO. AL970920

A Better Approximation Algorithm for Finding Planar Subgraphs

Gruia Călinescu*

College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332-0280

Cristina G. Fernandes[†]

College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332-0280

Ulrich Finkler[‡]

Max-Planck-Institut für Informatik, D-66123 Saarbrücken, Germany

and

Howard Karloff*

College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332-0280

Received March 28, 1996; revised June 2, 1997

The MAXIMUM PLANAR SUBGRAPH problem—given a graph G, find a largest planar subgraph of G—has applications in circuit layout, facility layout, and graph drawing. No previous polynomial-time approximation algorithm for this NP-Complete problem was known to achieve a performance ratio larger than $1/3$, which is achieved simply by producing a spanning tree of G. We present the first approximation algorithm for MAXIMUM PLANAR SUBGRAPH with higher performance ratio ($4/9$ instead of $1/3$). We also apply our algorithm to find large outerplanar subgraphs. Last, we show that both MAXIMUM PLANAR SUBGRAPH and its complement, the problem of removing as few edges as possible to leave a planar subgraph, are Max SNP-Hard. © 1998 Academic Press

* Research supported in part by NSF Grant CCR-9319106.
[†] Research supported in part by the CNPq (Brazil), under contract 200975/92-7.
[‡] Research supported in part by the Graduiertenkolleg Effizienz und Komplexität von Algorithmen und Rechenanlagen, Universität Saarbrücken.

1. INTRODUCTION

MAXIMUM PLANAR SUBGRAPH is the following problem: given a graph G, find a planar subgraph of G with the maximum number of edges. This problem has applications in circuit layout, facility layout, and graph drawing [F92, TDB88].

MAXIMUM PLANAR SUBGRAPH is known to be NP-complete [LG77]. For a graph G, let us define $Opt(G)$ to be the maximum size of a planar subgraph of G, where *size* is the number of edges. Given an algorithm A that takes graphs G as input and outputs subgraphs of G, define $\mathrm{A}(G)$ to be the size of the planar graph A produces when G is the input. Now let us define A's *performance* or *approximation ratio* $r(\mathrm{A})$ to be the infimum, over all graphs G, of $\mathrm{A}(G)/Opt(G)$ (if $Opt(G) > 0$, and 1 otherwise). In the literature, authors sometimes ensure that their performance ratio is at least 1 by defining it to be the reciprocal of ours.

Numerous approximation algorithms for MAXIMUM PLANAR SUBGRAPH appear in the literature, the simplest ones being Spanning Tree (output any spanning tree of G, assuming G is connected) and Maximal Planar Subgraph (output any planar subgraph to which the addition of any new edge would violate planarity). Spanning Tree is known to have performance ratio $1/3$ (see below). Dyer, Foulds, and Frieze [DFF85] proved that Maximal Planar Subgraph has performance ratio $1/3$. Cimikowski [Cim95] proved that a path-embedding heuristic of Chiba, Nishioka, and Shirakawa [CNS79] and an edge-embedding heuristic of Cai, Han, and Tarjan [CHT93] have performance ratios not exceeding $1/3$. In the same paper, Cimikowski studied two other polynomial-time heuristics: the "vertex-addition heuristic" and the "cycle-packing heuristic." The performance ratio of the former, to the authors' knowledge, is not known, whereas for the cycle-packing algorithm, it is 0. Dyer, Foulds, and Frieze [DFF85] studied two other algorithms and proved that each has performance ratio at most $2/9$. Also see [JM93].

In short, to the authors' knowledge, no previously proposed algorithm was known to have a performance ratio exceeding $1/3$. What makes the problem more tantalizing is that achieving a performance ratio of $1/3$ is trivial. In fact, Spanning Tree has performance ratio $1/3$, since every spanning tree of a connected graph on n vertices has $n - 1$ edges and every planar graph on n vertices has at most $3n - 3 = 3(n - 1)$ edges (and there are planar graphs on n vertices with $3n - 6$ edges, for all $n \geq 3$). No previous algorithm could beat the bound achieved by a trivial algorithm.

In this paper, we present two new approximation algorithms for MAXIMUM PLANAR SUBGRAPH. Each achieves a performance ratio exceeding $1/3$. The higher performance ratio is $4/9 = 0.444\ldots$ and is achieved by an algorithm which (surprisingly) invokes an algorithm for the graphic matroid parity problem as a subroutine and which runs in time $O(m^{3/2}n \log^6 n)$, where m is the number of edges in G and n is the number of vertices in G. A greedy variant still has performance ratio $7/18 = 0.3888\ldots$ and runs in linear time on graphs of bounded degree.

Next, we provide an extension of the main algorithm. We provide a nontrivial approximation algorithm for MAXIMUM OUTERPLANAR SUBGRAPH, which is this problem: given G, find an outerplanar subgraph of G of maximum size. (An *outerplanar* graph is a graph which can be drawn in the plane without crossing edges, with all vertices on the boundary of the exterior face [H72].) This new algorithm has performance ratio at least $2/3$, which surpasses the bound of $1/2$ which is trivially obtained by producing a spanning tree.

Last, we show that MAXIMUM PLANAR SUBGRAPH is Max SNP-Hard, implying that there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at least $1 - \epsilon$ would imply that $P = NP$ [ALMSS92]. In addition, we show that the complementary problem, called NONPLANAR DELETION or NPD—given $G = (V, E)$, produce a smallest subset $L \subseteq E$ such that $(V, E - L)$ is planar—is also Max SNP-Hard.

2. THE APPROXIMATION ALGORITHMS

In this section we present the two new algorithms for MAXIMUM PLANAR SUBGRAPH.

Let us give some motivation for our algorithm. As we said, given a (connected) graph G, an algorithm which outputs a spanning tree of G achieves a performance ratio of $1/3$. A graph whose cycles all have length 3, that is, are triangles, is planar, as it cannot contain a subdivision of K_5 or $K_{3,3}$. Moreover, note that a connected spanning subgraph of G whose cycles are triangles, besides being planar, has one more edge per triangle than a spanning tree of G has.

Our better algorithm produces a subgraph of G whose cycles are triangles, and, among these subgraphs, has the maximum number of edges. It can be implemented in time $O(m^{3/2}n \log^6 n)$, where m is the number of edges in G and n is the number of vertices in G, using a graphic matroid parity algorithm, as we will see later. We first present a greedy version of the algorithm.

2.1. *A Greedy Version of the Algorithm*

Algorithm A, presented below, is a greedy version of our algorithm. It has a performance ratio of $7/18 = 0.3888\ldots$. After presenting the algorithm and proving its performance ratio is $7/18$, we will show it can be implemented in linear time for graphs with bounded degree. We begin with some definitions.

A *triangular cactus* is a graph whose cycles (if any) are triangles and such that all edges appear in some cycle. A *triangular cactus in a graph* G is a subgraph of G which is a triangular cactus.

A *triangular structure* is a graph whose cycles (if any) are triangles. A *triangular structure in a graph* G is a subgraph of G which is a triangular structure. Note that every triangular cactus is a triangular structure, but not vice versa.

Algorithm A produces a triangular structure in the given graph G. The algorithm consists of two phases. First, A greedily constructs a maximal triangular cactus S_1 in G. Second, A extends S_1 to a triangular structure S_2 in G by adding as many edges as possible to S_1 without forming any new cycles.

Given a graph $G = (V, E)$ and $E' \subseteq E$, we denote by $G[E']$ the *spanning* subgraph of G induced by E', that is, the graph (V, E'). (Note that this is not the usual definition of a subgraph induced by an edge set, since we require the subgraph to be spanning.)

ALGORITHM A. Starting with $E_1 = \emptyset$, repeatedly (as long as possible) find a triangle T whose vertices are in different components of $G[E_1]$, and add the edges of T to E_1.
Let $S_1 := G[E_1]$.
Starting with $E_2 = E_1$, repeatedly (as long as possible) find an edge e in G whose endpoints are in different components of $G[E_2]$, and add e to E_2.
Let $S_2 := G[E_2]$.
Output S_2.

Note that S_2 is indeed a triangular structure in G. As we mentioned before, S_2 is planar since it does not contain cycles of length greater than 3.

THEOREM 2.1. *The performance ratio of Algorithm* A *is* $7/18$.

Proof. First, let us show that the performance ratio is at least $7/18$. Without loss of generality, we may assume G is connected, and has at least three vertices. Observe that the number of edges in S_2 is the number of edges in a spanning tree of G plus the number of triangles in S_1. So it suffices to count the number of triangles in S_1.

Let H be a plane embedding of a maximum planar spanning subgraph of G. Let $n \geq 3$ be the number of vertices in G, and let $t \geq 0$ be such that $3n - 6 - t$ is the number of edges in H. We can think of t as the number of edges missing for H to be a triangulated plane graph. The number of triangular faces in H is at least $2n - 4 - 2t$. (This is a lower bound on the number of triangular faces of H since if H were triangulated, it would have $2n - 4$ triangular faces, and each missing edge can destroy at most two of these triangular faces.)

Let k be the number of components of S_1 each with at least one triangle, and let $p_1, p_2, \dots, p_k$ be the number of triangles in each of these components. Let $p = \Sigma_{i=1}^{k} p_i$. We will prove that p, the number of triangles in S_1, is at least a constant fraction of $n - 2 - t$. Note that if a triangle cannot be added to S_1, it is because two of its vertices are in the same component of S_1. Hence one of its edges has its two endpoints in the same component of S_1. This means that at the end of the first phase, every triangle in G must have some two vertices in the same component of S_1. In particular, every triangular face in H must have some two vertices in the same component of S_1, and therefore one of its three edges must be in the subgraph of H induced by the vertices in a component of S_1. Thus we can associate with each triangular face F in H an edge e in F whose endpoints are in the same component of S_1. But any edge e in H lies in at most two triangular faces of H, so e could have been chosen by at most two triangular faces of H. It follows that the number of triangular faces in H is at most twice the number of edges in H whose endpoints are in the same component of S_1.

Let H' be the subgraph of H induced by the edges of H whose endpoints are in the same component of S_1. Note that $p_i \geq 1$, for all i, and that the number of vertices in the ith component of S_1 is $2p_i + 1 \geq 3$. Since H' is planar, H' has at most $\Sigma_{i=1}^{k}(3(2p_i + 1) - 6) = 6p - 3k$ edges. By the observation at the end of the previous paragraph, $2(6p - 3k) \geq 2|E(H')| \geq$ (number of triangular faces in H) $\geq 2n - 4 - 2t$. From this, we have

$$p \geq \frac{n - 2 - t + 3k}{6} \geq \frac{n - 2 - t}{6}.$$

Therefore the number of triangles in S_1 is at least $(n - 2 - t)/6$, and the ratio between the number of edges in S_2 and the number of edges in H is at least

$$\frac{n - 1 + (n - 2 - t)/6}{3n - 6 - t} = \frac{7n - 8 - t}{18n - 36 - 6t} \geq \frac{7}{18},$$

since $t \geq 0$. This completes the proof that the performance ratio of Algorithm A is at least $7/18$.

Now, we will prove that the performance ratio is at most $7/18$. Let S be any connected triangular cactus with $p > 0$ triangles. S has $2p + 1 \geq 3$ vertices. Let S' be any triangulated plane supergraph of S on the same set of vertices (S' can be obtained from S by adding edges to S until it becomes triangulated). Since S' is triangulated, S' has $2(2p+1)-4 = 4p-2$ (triangular) faces. For each face of S', add a new vertex in the face and adjacent to all vertices on the boundary of that face. Let G be the new graph. Observe that G is a triangulated plane graph and has $(2p+1) + (4p-2) = 6p-1$ vertices. This means that G has $3(6p-1)-6 = 18p - 9$ edges. With G as input for Algorithm A, in the first phase it can produce $S_1 = S$, and S_2 can be S plus one edge for each of the new vertices (the vertices in G not in S). The number of edges in S is $3p$. Hence S_2 can have $3p + (4p-2) = 7p-2$ edges, while G has $18p-9$ edges. Thus the ratio between the number of edges in S_2 and the number of edges in G is

$$\frac{7p-2}{18p-9}.$$

By choosing p as large as we wish, we get a ratio as close to $7/18$ as we want. ▮

2.1.1. *Linear Time for Bounded-Degree Graphs*

In case G has bounded degree d, we can implement Algorithm A in linear time. First, we describe the implementation of the first phase. At any time, the vertices of the graph are partitioned into three sets: new, active, and used. At the beginning, all the vertices are new and $E_1 = \varnothing$. The following process is repeated indefinitely. Test if there are any new vertices. If there are none, halt. Test if there are any active vertices. If there are none, choose a new vertex and make it active. Choose an active vertex x and "use" it: that is, while there exist two new vertices y and z adjacent to x and to each other, choose such a pair, add the edges of the triangle xyz to E_1 and make y and z active. When there are no two new vertices adjacent to x and adjacent to each other, mark x used.

We will show in detail how "using" one vertex takes time $O(d^2)$. All the other operations—keeping lists of new and active vertices, initializing an array with the status of each vertex, initializing an auxiliary array, and maintaining the list of edges in E_1—can easily be implemented in $O(n)$ time. This means the running time of this implementation is $O(nd^2)$.

Before and after "using" a vertex x, the auxiliary array has a 0 for each vertex. Going through the adjacency list of x, we change to 1 the entry of the auxiliary array for each new vertex encountered. Then, going again through the adjacency list of x, for each new vertex u encountered, go through the adjacency list of u, checking if some neighbor of u is marked 1 in the auxiliary array. This takes time $O(d)$ for each neighbor of x, for a total of $O(d^2)$ time. In this way we find, one after the other, all triangles formed by x and two new vertices. Each time such a triangle is found, mark as active the two new vertices in the triangle and place a 0 for each in the auxiliary array. At the end of this process, go through the adjacent list of x and change back to 0 all remaining entries which are 1 in the auxiliary array.

First, we note that E_1 is a triangular cactus. Indeed, at any time when we add a triangle to E_1, two of the vertices of the triangle are of type new, and since a new vertex is always an isolated vertex in $G[E_1]$, adding the three edges of the triangle keeps $G[E_1]$ a triangular cactus.

To prove that E_1 is maximal, we maintain the following invariant. All vertices which are active at a given time are in the same connected component of $G[E_1]$ at that time. To see this, note that a node u is made active either because there are no active nodes left, in which case u is the only node active, or is made active by an active vertex x, a neighbor in G of u which is being used, and in this case the edge ux is included in $G[E_1]$.

Now, let us prove that E_1 at the end of the process is maximal, in that no triangles can be added to it. Consider a triangle xyz not in E_1. The triangle xyz has at least two vertices in the same component of $G[E_1]$ for the following reason. When the process stops, it is because there are no new vertices. Thus either x, y, and z are active, or at least one of them is used. If x, y, and z are active, by the invariant, the three of them are in the same component of $G[E_1]$. In the other case, assume x is the first among x, y, and z to be used. While x was being used, the triangle xyz was processed. It was not added to E_1 because at least one of y and z, say y, was active. Since x was also active at that moment, x and y were (and still are) in the same component of $G[E_1]$, by the invariant.

In this paragraph we describe how to implement in linear time the second phase of Algorithm A. The time is $O(|V(G)| + 1E(G)|)$ and does not depend on the graph's having bounded degree. Recall that at the end of the first phase we have a set E_1 of edges. First, compute the connected components of $G[E_1]$. Let $C(v)$ be the connected component of $G[E_1]$ containing vertex $v \in V(G)$. Then construct a multigraph H in the following way: $V(H)$ is the set of connected components of $G[E_1]$, and for each edge $uv \in E(G)$, include in $E(H)$ one edge (labeled uv) with endpoints $C(u)$ and $C(v)$ if $C(u) \neq C(v)$. Run a graph traversal algorithm (for example, DFS) on H and obtain a maximum spanning forest F (which we view as a subset of $E(H)$) of H. Output $E_2 = \{uv |\text{there is an edge in } F \text{ labeled } uv\}$.

2.2. *A Better Algorithm*

The new algorithm, Algorithm B below, finds a *maximum* triangular structure (one with the maximum number of edges) in a given graph G. Algorithm B has performance ratio 4/9, and can be implemented in time

$O(m^{3/2} n \log^6 n)$. Now let us present the algorithm and the analysis of its performance ratio.

Algorithm B also has two phases. In the first one, B constructs a maximum triangular cactus S_1 in G. We will show later how to use a graphic matroid parity algorithm to construct S_1. In the second phase, B extends S_1 to a triangular structure S_2 in G, as before, by adding to S_1 as many edges as possible which do not form new cycles.

ALGORITHM B. Let S_1 be a *maximum* triangular cactus in G.
Starting with $E_2 = E(S_1)$, repeatedly (as long as possible) find an edge e in G whose endpoints are in different components of $G[E_2]$, and add e to E_2.
Let $S_2 := G[E_2]$.
Output S_2.

Observe that S_2 is a triangular structure in G, and therefore is planar. To analyze the algorithm, we need a definition. In any graph L, let $mts(L)$ denote the number of edges in a maximum triangular structure in L. Define $\rho(L) = mts(L)/|E(L)|$ if $E(L) \neq \emptyset$, and $\rho(L) = 1$ if $E(L) = \emptyset$.

The main result of this section is Corollary 2.11, which states that the performance ratio of Algorithm B is 4/9. As we will see, the proof of Corollary 2.11 is not difficult, given the next two theorems.

Let $G = (V, E)$ be a graph on n vertices. Let $\mathscr{P} = \{V_1, \dots, V_k\}$ be a partition of the vertices of G and $\mathscr{Q} = \{E_1, \dots, E_m\}$ be a partition of the edges of G. For $1 \leq i \leq m$, let u_i denote the number of classes (sometimes referred to as *color classes*) V_j of $\mathscr{P}$ met by E_i. We call the ordered pair $(\mathscr{P}, \mathscr{Q})$ *valid for* G if every triangle of G has either at least two vertices in the same part of $\mathscr{P}$ or all three edges in the same part of $\mathscr{Q}$. Set

$$\Phi(\mathscr{P}, \mathscr{Q}) = n - k + \sum_{i=1}^{m} \left\lfloor \frac{u_i - 1}{2} \right\rfloor. \qquad (1)$$

Note that $\Phi(\mathscr{P}, \mathscr{Q}) \geq 0$, and that there is always a valid pair $(\mathscr{P}, \mathscr{Q})$ for G (e.g., $\mathscr{P} = \{V(G)\}$, $\mathscr{Q} = \{E(G)\}$).

According to Lovász and Plummer [LP86], we have the following theorem:

THEOREM 2.2. *The number of triangles in a maximum triangular cactus in a graph G is equal to the minimum of $\Phi(\mathscr{P}, \mathscr{Q})$ taken over all valid pairs $(\mathscr{P}, \mathscr{Q})$ for G.*[1]

[1] Lovász was contacted and agreed that the formula given here, which differs slightly from that in [LP86], is correct.

Let G be a planar graph with n vertices. Embed G in the plane without crossing edges, obtaining a plane graph. Let t be the number of edges missing for this embedding to be triangulated. A triangulated plane graph has $3n - 6$ edges, if $n \geq 3$. So $t = (3n - 6) - |E(G)|$; t does not depend on the embedding.

We will prove the following theorem.

THEOREM 2.3. *Let G be a connected planar graph with $n \geq 3$ vertices. Let t be the number of missing edges, defined as above. Then*

$$\Phi(\mathscr{P}, \mathscr{Q}) \geq \frac{1}{3}(n - 2 - t) \tag{2}$$

for all valid pairs $(\mathscr{P}, \mathscr{Q})$ for G.

As shown in Theorem 2.10, it follows easily from Theorems 2.2 and 2.3 that $\rho(H) \geq 4/9$ for all planar graphs H.

The proof of Theorem 2.3 is technical and long, so let us start with a general description. The proof is by induction on the number of edges in G, the basis being simple to prove, as we will see. The inductive step starts with a connected plane graph G and a valid pair $(\mathscr{P}, \mathscr{Q})$, where $\mathscr{P} = \{V_1, \ldots, V_k\}$ and $\mathscr{Q} = \{E_1, \ldots, E_m\}$. We concentrate on *singletons*, which are vertices v of G such that $\{v\} \in \mathscr{P}$.

The singletons are partitioned into three sets A, B, and C as follows. Let G' be a triangulated plane supergraph of G with the same set of vertices. A is the set of singletons all of whose neighbors in G' have the same color (all neighbors in G' are in the same V_j for some j). A *facial triangle* is a triangle in G' which is the boundary of some face of G'. B is the set of singletons v not in A such that there is a facial triangle containing v with all edges in one edge class E_i for some i. We say singleton $v \in B$ *chooses* some such edge class E_i. C contains all remaining singletons.

Each singleton in B will choose an edge class E_i. For $1 \leq i \leq m$, let u_i be the number of classes V_j of $\mathscr{P}$ met by E_i, and let s_i be the number of singletons in B that have chosen E_i.

The inductive step is divided into two cases. The simpler case is the one in which there is an i such that $u_i = s_i = 4$. In this case, induction is applied to a smaller graph constructed from G, and after some accounting, the theorem follows.

The hard case is the one in which there is no i such that $u_i = s_i = 4$. In this case, the result will be proved directly—induction will not be applied. Recall that t is the number of edges missing for G to be triangulated (i.e., the number of edges in G' not in G). Let s be the number of singletons, let p be the number of nonsingleton vertices, that is, $p = n - s$ (where n

is the number of vertices of G), and let q be the number of nonsingleton colors, that is, $q = k - s$. Then the following upper bounds on $|A|$, $|B|$, and $|C|$ will be proved:

- $|A| \leq 2p - 3q - c'$ (Lemma 2.4),
- $|B| \leq 3\Sigma_{i=1}^{m}\lfloor(u_i - 1)/2\rfloor$ (Lemma 2.5),
- $|C| \leq t + c'$ (Lemma 2.6)

for some carefully chosen c'.

The upper bounds on $|A|$ and $|B|$ are not hard. The hard one is the upper bound on $|C|$. The theorem follows easily from these upper bounds and some accounting.

Proof of Theorem 2.3. By induction on the number of edges of G. Let us denote by E_G the edge set of G and by V_G its vertex set.

Basis. $|E_G| \leq 2n - 4$.

Then $t \geq (3n - 6) - (2n - 4) = n - 2$ and therefore $\Phi(\mathscr{P}, \mathscr{Q}) \geq 0 \geq (1/3)(n - 2 - t)$ for all valid pairs $(\mathscr{P}, \mathscr{Q})$ for G.

Inductive step. $|E_G| > 2n - 4$.

Suppose that for any connected planar graph H on n vertices such that $|E_H| < |E_G|$, $\Phi(\mathscr{P}, \mathscr{Q}) \geq (1/3)(n - 2 - t_H)$ for all valid pairs $(\mathscr{P}, \mathscr{Q})$ for H, where t_H denotes the number of edges missing for H to be triangulated. Let us prove that $\Phi(\mathscr{P}, \mathscr{Q}) \geq (1/3)(n - 2 - t)$ for all valid pairs $(\mathscr{P}, \mathscr{Q})$ for G.

We begin by embedding G in the plane without crossings, obtaining a plane graph. Next, we augment G to get a triangulated plane graph G'. The edges we add are called *missing edges*. The number of missing edges in G' is t.

Let $(\mathscr{P}, \mathscr{Q})$ be a valid pair for G, with $\mathscr{P} = \{V_1, \dots, V_k\}$ and $\mathscr{Q} = \{E_1, \dots, E_m\}$. As before, for $1 \leq i \leq m$, let u_i denote the number of color classes V_j of $\mathscr{P}$ met by E_i. We may assume that

$$u_i \leq 2 \Rightarrow |E_i| = 1. \tag{3}$$

If $u_i \leq 2$ and $|E_i| > 1$, then we can split E_i into individual edges, obtaining a new edge partition $\mathscr{Q}'$. Any triangle in G with three edges in the old E_i also has at least two vertices of the same color, because $u_i \leq 2$. Therefore $(\mathscr{P}, \mathscr{Q}')$ is also a valid pair for G. And, moreover, $\Phi(\mathscr{P}, \mathscr{Q}') = \Phi(\mathscr{P}, \mathscr{Q})$, because $\lfloor(u - 1)/2\rfloor = 0$ when $1 \leq u \leq 2$.

Assume that, for $1 \leq j \leq q$, $|V_j| = p_j \geq 2$ and, for $q < j \leq k$, $|V_j| = 1$. Let us call a vertex x a *singleton* if $\{x\} \in \mathscr{P}$; thus the last $k - q$ color classes are singletons. Let $s = k - q$ be the number of singletons and $p = \Sigma_{i=1}^{q} p_i$ be the number of vertices which are not singletons. So

$$n = p + s \qquad \text{and} \qquad k = q + s. \tag{4}$$

Observe that $\mathscr{P}$ is a partition of the vertices of G' (since G and G' have the same vertex set), $\mathscr{Q}$ is a partition of the edges of G (not of G'), and for any triangle of G' (not of G), facial or not, at least one of the following three conditions holds.

(1) At least two of the vertices are of the same color. We say that the triangle *is covered by a color*.

(2) All three edges are in the same class E_i of $\mathscr{Q}$. We say that the triangle *is covered by an edge class*.

(3) One or more of its edges is a missing edge. This corresponds to the fact that this triangle does not exist in G. We say that the triangle *is covered by a missing edge*.

A *facial triangle* is a triangle in G' which is the boundary of some face of G'. (Note that a facial triangle is not necessarily a triangle in G, but it is a triangle in G'.) A facial triangle T is a *facial triangle neighboring vertex* v if it contains v.

Let us partition the set of singletons into three sets A, B, and C, according to how the facial triangles neighboring these singletons are covered.

A: The set of singletons all of whose neighbors in G' are of the same color.

B: The set of singletons x not in A such that there is an edge class which covers one of the facial triangles neighboring x.

C: The set of singletons not in $A \cup B$.

Notice that all the facial triangles neighboring a singleton in A (even the ones containing missing edges) are covered by a color. Also, not all facial triangles neighboring a singleton in C are covered by a color class (otherwise, all of its neighbors in G' would have the same color and this singleton would be in A). And no facial triangle neighboring a singleton in C is covered by an edge class (otherwise, this singleton would be put in set B, if it is not already in A).

Let a, b, and c be the sizes of A, B, and C, respectively. Observe that s, the number of singletons, satisfies

$$s = a + b + c. \tag{5}$$

For each singleton x in B, we choose an i such that E_i covers a facial triangle neighboring x. We denote by s_i the number of singletons x in B which have chosen E_i. Observe that E_i meets all singletons which have chosen E_i; therefore

$$s_i \leq u_i. \tag{6}$$

Moreover, if $|E_i| = 1$, then $s_i = 0$, because E_i cannot cover any triangle.

Case 1. There is an i such that $u_i = s_i = 4$.

This is the simpler case, in which we apply induction.

In this case, there are four distinct singletons x, y, w, z in B which are the only vertices E_i meets. Each of x, y, w, z has chosen E_i, implying that E_i must cover a facial triangle neighboring x. This facial triangle meets only a subset of these four vertices. Without loss of generality, we may assume this facial triangle is xyw. E_i also covers a facial triangle neighboring z. Also without loss of generality, we may assume it is ywz. Then $\{xy, yw, xw, yz, wz\} \subseteq E_i \subseteq \{xy, yw, xw, yz, wz, xz\}$ (see Fig. 1).

Let G_1 be G after we remove the edges yw and xz (if edge xz exists). From the partition $\mathscr{Q}$ of $E(G)$ we obtain a partition $\mathscr{Q}_1$ of $E(G_1)$ in the following way: all parts of $\mathscr{Q}_1$ are the same as the corresponding parts of $\mathscr{Q}$, except that E_i is replaced by four parts. Each of these four parts contains exactly one edge from $E_i - \{yw, xz\}$. Note that indeed $\mathscr{Q}_1$ is a partition of $E(G_1)$. Also, the pair $(\mathscr{P}, \mathscr{Q}_1)$ is valid for G_1: each triangle in G_1 is covered exactly as it was covered in G (the triangles covered by class E_i do not exist in G_1).

Observe that G_1 is connected, and so it is a connected planar graph. Moreover, it has fewer edges than G. Thus we can apply induction, and conclude that $\Phi(\mathscr{P}, \mathscr{Q}_1) \geq (1/3)(n - 2 - t_1)$, where t_1 is the number of missing edges for G_1 to be triangulated. Because we removed at most two edges from G to get G_1, $t_1 \leq t + 2$.

The new classes in $\mathscr{Q}_1$ have zero contribution to $\Phi(\mathscr{P}, \mathscr{Q}_1)$ (since any edge class of size 1 contributes 0), and, since the contribution of E_i to $\Phi(\mathscr{P}, \mathscr{Q})$ is $\lfloor (4-1)/2 \rfloor = 1$, it follows that $\Phi(\mathscr{P}, \mathscr{Q}_1) = \Phi(\mathscr{P}, \mathscr{Q}) - 1$.

Putting all this together, we have

$$\begin{aligned}\Phi(\mathscr{P}, \mathscr{Q}) &= \Phi(\mathscr{P}, \mathscr{Q}_1) + 1 \geq \tfrac{1}{3}(n - 2 - t_1) + 1 \\ &\geq \tfrac{1}{3}(n - 2 - (t+2)) + 1 > \tfrac{1}{3}(n - 2 - t).\end{aligned}$$

Case 2. There is no i such that $u_i = s_i = 4$.

This is the hard case, whose proof requires several pages. We start by defining c', used in the upper bounds on a and c.

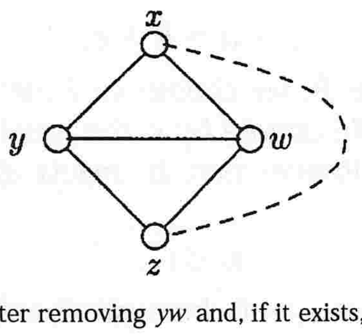


FIG. 1. Edges in E_i. After removing yw and, if it exists, xz, no triangles are left.

Consider pairs (j, F), where $1 \leq j \leq q$ and F is a face of $G'[V_j]$, the plane subgraph of G' induced by the vertex set V_j. Let c' be the number of such pairs (j, F) such that some vertex not in A is embedded in F. Call each of these c' pairs *special.*

Recall that p is the number of vertices which are not singletons, q is the number of color classes with more than one vertex, t is the number of missing edges in G', and a, b, c are the numbers of singletons in A, B, C, respectively.

The next lemmas give upper bounds on a, b, and c which will be needed to complete the proof of Theorem 2.3.

LEMMA 2.4. $a \leq 2p - 3q - c'$.

Proof. Each singleton x in A has all neighbors in G' of some color $j \leq q$; hence $p_j \geq 2$. Clearly, x is embedded in a face of $G'[V_G - \{x\}]$. Because all neighbors of x in G' are in V_j, this face F is also a face of $G'[V_j]$, and there cannot be any other vertex embedded in F. Therefore the size a of A is at most the number of pairs (j, F) which are not special.

The maximum number of pairs (j, F), where F is a face of $G'[V_j]$, is $2p_j - 3$ (if $p_j = 2$ then it has only one face; if $p_j > 2$ then it has the maximum number of faces when it is triangulated, and in this case, it has $2p_j - 4$ faces). Therefore there are at most $\Sigma_{j=1}^{q}(2p_j - 3) = 2p - 3q$ pairs (j, F), where $j \leq q$ and F is a face of $G'[V_j]$. From these pairs (j, F), c' of them are special (have a vertex which is not in A embedded in F). Thus the number of pairs (j, F) which are not special is at most $2p - 3q - c'$, and therefore $a \leq 2p - 3q - c'$. ∎

LEMMA 2.5. $b \leq 3\Sigma_{i=1}^{m}\lfloor(u_i - 1)/2\rfloor$.

Proof. If we prove that $s_i \leq 3\lfloor(u_i - 1)/2\rfloor$, then by summing $s_i \leq 3\lfloor(u_i - 1)/2\rfloor$ over $i \in \{1, 2, \dots, m\}$, the lemma follows, since $b = \Sigma_{i=1}^{m} s_i$.

First, by (3), if $u_i \leq 2$, then $|E_i| = 1$, which implies that $s_i = 0$. And, so $s_i = 0 = 3\lfloor(u_i - 1)/2\rfloor$. If $u_i > 2$, then we use the fact that inequality (6) implies that $s_i \leq u_i$. As a consequence, $s_i \leq 3\lfloor(u_i - 1)/2\rfloor$ can only be false if $u_i = s_i = 4$. But this does not happen because we are given that for no i does $u_i = s_i = 4$. ∎

Lemma 2.6 is the difficult upper bound. After this, we do some accounting, and the theorem follows.

LEMMA 2.6. $c \leq t + c'$.

Proof. Since there is nothing to prove if $c = 0$, we may assume $c \neq 0$, that is,

$$C \neq \varnothing. \tag{7}$$

A missing edge covers at most two facial triangles. Therefore it suffices to prove the existence of at least $2c - 2c'$ facial triangles which must be covered by missing edges.

Let us associate a set of facial triangles with each component of $G'[C]$. More specifically, for a component of $G'[C]$ whose vertex set is D, let us associate a set of either $2|D|$ or $2|D| - 2$ facial triangles, each with at least one vertex in D, such that each such facial triangle must be covered by a missing edge. We will make sure that each of these facial triangles will have its three vertices in different color classes and at least one of them is in C. (The triangle is not covered by a color class and, because of the vertex in C, it is not covered by an edge class. So it has to be covered by a missing edge.)

Observe that the sets of facial triangles corresponding to two different components of $G'[C]$ are disjoint (since there is no edge between two different components of $G'[C]$, there cannot be a triangle with vertices in two different components of $G'[C]$). If at most c' of the components were associated with a set with $2|D| - 2$ facial triangles, then in total we would have at least $2c - 2c'$ facial triangles which would have to be covered by missing edges (since $\Sigma_D |D| = c$). This would complete the proof of Lemma 2.6.

Let D be the vertex set of a component of $G'[C]$. A *relevant facial triangle for D* is a facial triangle of G' with at least two vertices in D. Let f_D be the number of faces of $G'[D]$ (which is a connected graph), let f_3 be the number of faces of $G'[D]$ which are also faces of G', and let e_D be the number of edges in $G'[D]$.

Let us start by proving that $C \neq V_G$. Because $|E_G| > 2n - 4$, there are at most $(3n - 6) - (2n - 3) = n - 3$ missing edges. A missing edge covers at most two facial triangles. So there are at most $2n - 6$ facial triangles covered by missing edges. But in G' there are $2n - 4$ facial triangles (all the faces of G'). Thus there are facial triangles covered by colors or by edge classes. If there are facial triangles covered by colors, then at least two vertices have the same color. This means that there are vertices which are not singletons, implying that $C \neq V_G$. If all vertices are singletons and a facial triangle is covered by an edge class, then the singletons neighboring this facial triangle are in B. It follows that in either case

$$C \neq V_G. \tag{8}$$

Embedded in the faces of $G'[D]$ are all the vertices of $V_G - D$. Now $V_G - D \supseteq V_G - C \neq \emptyset$. Thus there is at least one face of $G'[D]$ which contains some vertex of $V_G - D \neq \emptyset$. This face of $G'[D]$ is not a face of G', since no face of G' contains any vertex. Thus $f_3 < f_D$.

CLAIM 2.7. *If $f_3 = f_D - 1$ then there are at least $2|D| - 2$ relevant facial triangles for D. If $f_3 \leq f_D - 2$ then there are at least $2|D|$ relevant facial triangles for D.*

Proof. A relevant facial triangle for D has at least two vertices in D, which means it contains an edge e in $G'[D]$. But e is contained in exactly two facial triangles of G'. These are the only two relevant facial triangles for D that contain e. This would give us $2e_D$ of the desired relevant facial triangles for D, except that not all of them are distinct. If a facial triangle is counted by two edges, then all three of its vertices are in D, which implies it is a face of $G'[D]$ (and, being a facial triangle, also of G'). Moreover, it was counted exactly three times (once for each of its edges). Therefore we have $2e_D - 2f_3$ relevant facial triangles for D. Now, if $f_3 = f_D - 1$, applying Euler's formula, we get $2e_D - 2f_3 = 2e_D - 2(f_D - 1) = 2(|D| - 2) + 2 = 2|D| - 2$ relevant facial triangles. If $f_3 \leq f_D - 2$, we get $2e_D - 2f_3 \geq 2e_D - 2(f_D - 2) = 2(|D| - 2) + 4 = 2|D|$. ∎

Include these $2|D|$ (if possible) or $2|D| - 2$ relevant facial triangles in the set of facial triangles corresponding to $G'[D]$.

To guarantee that at most c' of the components of $G'[C]$ have only $2|D| - 2$ facial triangles in their corresponding sets, we will need to add more facial triangles to some of the sets that currently have only $2|D| - 2$. Now let D be the vertex set of a component of $G'[C]$ such that all the faces of $G'[D]$ but one are faces of G', that is, $f_3 = f_D - 1$. (Any component induced by a set D' with $f_3 \leq f_{D'} - 2$ already has $2|D'|$ associated facial triangles.)

Since $f_3 = f_D - 1$, all the faces of $G'[D]$ but one are faces of G'. It follows that all the vertices in $V_G - D$ must be embedded in the same face of $G'[D]$ (that face is the only face which is not a face of G'). Applying Claim 2.8 (presented below) to G' and D (as M and S, respectively), we conclude that $G'[V_G - D]$ is connected.

We use the terminology given in [BM76] for a facial walk and for the length of a face.

CLAIM 2.8. *Let $M = (V, E)$ be a connected plane multigraph (possibly with loops) whose faces all have length at most 3. Let S be a nonempty set of vertices of M such that the subgraph $M[S]$ of M induced by S is connected, and all vertices in $V - S$ are embedded in the same face of $M[S]$. Then $M[V - S]$ is connected.*

Proof. We may assume that $V - S \neq \emptyset$. By induction on the number of edges in $M[S]$.

Basis. $M[S]$ has no edge.

$M[S]$ consists of a unique vertex u. Suppose $M[V - S]$ is not connected. Because M is connected, all the components of $M[V - S]$ are adjacent to some vertex in S, that is, to u. The plane embedding of M determines a cyclic ordering of the edges of M incident with u. Then there exist two consecutive edges in the cyclic ordering, say e_1 and e_2, such that their other endpoints belong to different components of $M[V - S]$. But this is impossible, because then the length of the face of M lying between e_1 and e_2 would be at least 4, a contradiction.

Inductive Step. There is an edge in $M[S]$.

If there is an e in $M[S]$ with two distinct endpoints u and v, contract e obtaining a new connected plane multigraph H with a new vertex w corresponding to u and v. (H's vertex set V' is $(V - \{u, v\}) \cup \{w\}$.) Let $S' = (S - \{u, v\}) \cup \{w\}$. The hypotheses of the claim hold for H and S': H is connected, $H[S']$ is connected ($H[S']$ is $M[S]$ after contracting e), all the faces of H have length at most 3 (since contracting edges cannot increase the length of a face), $S' \neq \emptyset$, and all vertices in $V' - S' = V - S$ are embedded in the same face of $H[S']$. (There is a natural correspondence between faces of a graph and faces of the graph obtained after contracting an edge of the graph.) Furthermore, $H[S']$ has fewer edges than $M[S]$. Hence, by the inductive hypotheses, $H[V' - S']$ is connected, and hence $M[V - S] = H[V' - S']$ is connected.

If every edge in $M[S]$ is a loop, then let e be such a loop, and call its unique endpoint u. Note that because $M[S]$ is connected, $S = \{u\}$. Then e divides the plane (minus e and u) into two open regions R and R'. Because all vertices of $V - S$ are embedded in the same face of $M[S]$, there is no vertex in $V - S$ embedded in one of these two regions; without loss of generality, let it be R. So there is no vertex embedded in R. This implies that any edge embedded in R is a loop, with u as its endpoint.

We consider two cases. The first case is the one in which there is some edge embedded in R. Then let H be the graph obtained from M by removing all edges of M embedded in R (they are all loops). Note that H has fewer edges than M. We will prove that the hypotheses of the claim hold for H and S. H and $H[S]$ are connected since we removed only loops. The only face of H which is not a face of M is R, and the boundary of R consists of only e: R has length 1. Therefore all the faces of H have length at most 3. So, by the inductive hypothesis, $H[V - S]$ is connected, and hence $M[V - S] = H[V - S]$ is connected.

In the second case, there is no edge embedded in R. Hence R is a face of M (with only one edge on the boundary). Let H be the graph obtained from M by removing e. Again, H has fewer edges than M. Let F be the other face of M having e in its boundary. Removing e creates only one new face F' in H (which is the union of R, F, and e). The boundary of the new face F' is the union of the boundaries of R and F minus e. The boundary of R consists of only e. Thus the boundary of F' has one edge (edge e) fewer than the boundary of F, implying that the length of F' is 1 less than the length of F. Therefore all the faces of H have length at most 3. So, by the inductive hypotheses, $H[V - S]$ is connected, and therefore $M[V - S] = H[V - S]$ is connected. ▮

We proceed with the proof of Lemma 2.6.

Because $G'[D]$ is connected, all vertices of D are embedded in the same face of $G'[V_G - D]$. Call this face F'.

CLAIM 2.9. *One of the following holds*:

(a) *All the vertices on the boundary of F' are of the same color $j \leq q$. In this case, F' is a face of $G'[V_j]$.*

(b) *There are two distinct facial triangles of G', each with exactly one vertex in D and the other two vertices of different colors.*

Proof. The boundary of F' is a closed walk in $G'[V_G - D]$. Moreover, $V_G - D$ cannot be a set containing only one vertex x. This is true because of the following reasoning. Otherwise this vertex would have to be a singleton (because vertices in D are singletons, none can have the same color as x), more specifically, a singleton in A or in B (it could not be in C, because it is adjacent to at least one vertex of D, and hence would be in the same component of $G'[C]$ as defined by D), with all neighbors in (D and hence) C. Any vertex of G' has at least two neighbors, and a singleton x in A has all the neighbors of the same color, and therefore none of its neighbors can be a singleton. It follows that no neighbor of x can be in C; hence $x \notin A$. A singleton x in B has a neighboring facial triangle (with vertices x, y, and z) covered by an edge class. But then y cannot be in C (since it has a neighboring facial triangle covered by an edge class); hence $x \notin B$.

Therefore $|V_G - D| \geq 2$. Hence the boundary of F' cannot consist of only one vertex. (If there is only one vertex on the boundary of a face of a simple plane graph, then the graph consists of only this vertex, and this does not happen because $|V_G - D| \geq 2$.)

If all vertices on the boundary of F' have the same color j, then clearly $j \leq q$ (there are at least two vertices on the boundary of F' and hence in V_j, so $j \leq q$). Also, because the only vertices embedded in F' are the vertices of D, which are not of color j, F' is also a face of $G'[V_j]$. This is case (a).

If, on the other hand, not all the vertices on the boundary of F' have the same color, then, traversing the boundary of F' (which is a closed walk), we will find at least two "oriented" edges whose endpoints have different colors. Each edge on the boundary of F' is on the boundary of exactly two triangular faces of G'. At least one of these triangular faces is contained in F'. More specifically, if an edge on the boundary of F' is traversed twice, then the two triangular faces containing this edge on the boundary are contained in F'.

Let us show that any of these triangular faces contained in F' has a vertex of D on its boundary. Suppose not. A triangular face with the three vertices on its boundary in $V_G - D$ is a face of $G'[V_G - D]$. So this triangular face would be a face of $G'[V_G - D]$ contained in F', which is a face of $G'[V_G - D]$. Because faces of a graph are disjoint, this can only happen if F' coincides with this triangular face of G'. But this is impossible, because F' is not a face of G' (all vertices in D are embedded in F').

Traversing the boundary of F', for each "oriented" edge we traverse, we find a (distinct) facial triangle of G' contained in F', containing the (unoriented version of the) oriented edge and a vertex of D on its boundary (if the two "oriented" edges are not the same unoriented edge, then each appears on the boundary of a different triangle, otherwise, the one unoriented edge appears in two distinct triangles). The facial triangles corresponding to the two "oriented" edges we picked satisfy the requirements of (b). ▮

If (b) holds, add the two extra facial triangles to the set of facial triangles corresponding to $G'[D]$. Note that they are different from the ones already in the set because they contain exactly one vertex of D and all the others contain at least two vertices of D. Each set D for which (b) holds is now associated with a set of at least $2|D|$ facial triangles.

For how many components of $G'[C]$ can (a) hold? Let us show that each component of $G'[C]$ for which (a) holds corresponds to one of the c' special pairs, and that different components correspond to different special pairs.

Let D be the vertex set of a component of $G'[C]$ for which (a) holds. Let us prove that there is a special pair (j, F') such that the only vertices of C embedded in F' are the vertices in D. Let F' be the face of $G'[V_G - D]$ in which the vertices of D are embedded. Recall that $G'[V_G - D]$ is connected. Because (a) holds for this component, F' is a face of $G'[V_j]$. Since the vertices of D are embedded in F', and vertices of D are not in A, some vertex not in A is embedded in F'; this makes the pair (j, F') special. The only vertices of G' embedded in F' are in D (because F' is a face of $G'[V_G - D]$). Since the only vertices of C embedded in F' are the vertices in D, no other component of $G'[C]$ (with a different set D) will correspond to this pair (j, F').

So the number of components such that (a) holds is at most the number c' of special pairs. Thus there are at most c' components of $G'[C]$ such that (a) holds, which completes the proof of Lemma 2.6, by the discussion in the third paragraph of the proof. ▮

Using (1) and the upper bounds given by the previous lemmas, we conclude the proof of Theorem 2.3 as follows:

$$\begin{aligned}
\Phi(\mathscr{P},\mathscr{Q}) &= n - k + \sum_{i=1}^{m} \left\lfloor \frac{u_i - 1}{2} \right\rfloor && \text{by (1)} \\
&= p - q + \sum_{i=1}^{m} \left\lfloor \frac{u_i - 1}{2} \right\rfloor && \text{by (4)} \\
&\geq p - q + \frac{1}{3} b && \text{by Lemma 2.5} \\
&= p - q + \frac{1}{3}(s - a - c) && \text{by Eq. (5)} \\
&\geq p - q + \frac{1}{3}(s - (2p - 3q - c') - (t + c')) \\
& && \text{by Lemmas 2.4 and 2.6} \\
&= \frac{1}{3}(p + s) - \frac{1}{3}t \\
&= \frac{1}{3}n - \frac{1}{3}t && \text{by the first equation in (4)} \\
&> \frac{1}{3}(n - 2 - t).
\end{aligned}$$

This completes the proof of Theorem 2.3. ▮

Recall that a triangular structure is a graph all of whose cycles are triangles. Also that $mts(L)$ denotes the number of edges in a maximum triangular structure in a graph L. And $\rho(L) = mts(L)/|E(L)|$ is $E(L) \neq \varnothing$, and $\rho(L) = 1$ if $E(L) = \varnothing$.

THEOREM 2.10. *If H is a planar graph, then $\rho(H) \geq 4/9$.*

Proof. We may assume H is connected and has at least three vertices. Let t be the number of missing edges for H to be triangulated. By Euler's formula, $|E(H)| = 3n - 6 - t$, where $n \geq 3$ is the number of vertices of H.

A maximum triangular structure in H can be obtained by extending a maximum triangular cactus to a connected graph (by adding edges without forming any new cycles). Also, a maximum triangular structure in H has one more edge per triangle than a spanning tree of H.

From Theorems 2.2 and 2.3, the number of triangles in a maximum triangular cactus is at least $(1/3)(n-2-t)$. From this, we conclude that $mts(H) \geq n-1+(1/3)(n-2-t)$, and then

$$\rho(H) \geq \frac{n-1+(1/3)(n-2-t)}{3n-6-t} = \frac{4n-5-t}{9n-18-3t} \geq \frac{4}{9}$$

for all $t \geq 0$. ∎

COROLLARY 2.11. *The performance ratio of Algorithm* B *is exactly* $4/9$.

Proof. Let G be a graph and let H be a maximum planar subgraph of G. Clearly, $mts(G) \geq mts(H)$. Now $Opt(G) = |E(H)|$ implies that $B(G)/Opt(G) = mts(G)/|E(H)| \geq mts(H)/|E(H)| = \rho(H)$. By Theorem 2.10, $\rho(H) \geq 4/9$ for any planar graph H. And from this we infer that the performance ratio of B is at least $4/9$.

Next we prove that the performance ratio of Algorithm B is at most $4/9$. Let G' be any triangulated plane graph on n' vertices. Call V' the vertex set of G'. Since G' is triangulated, G' has $2n'-4$ (triangular) faces. For each face of G', add a new vertex in the face and adjacent to all three vertices on the boundary of that face. Let G be the new graph and let V be the vertex set of G. Observe that G is a triangulated plane graph, and has $n' + (2n'-4) = 3n'-4$ vertices. Therefore G has $3(3n'-4)-6 = 9n'-18$ edges. Let S be a maximum triangular structure in G.

Any edge in G has at least one endpoint in V'. Moreover, $|V'| = n'$. Therefore a maximum matching in G has at most n' edges (each with at least one distinct endpoint in V'). The following lemma is observed in [LP86, p. 440].

LEMMA 2.12. *If S is a triangular structure with t triangles in a given graph G, then there is a matching in G of size t.*

Using Lemma 2.12, we conclude that S has at most n' triangles. Recall that S, being a triangular structure, is a spanning tree of G plus one edge per triangle in S, which implies that S has at most $(3n'-5)+n' = 4n'-5$ edges. Furthermore, G has $9n'-18$ edges. Therefore the ratio between the number of edges in S and the number of edges in G is

$$\frac{4n'-5}{9n'-18}.$$

By choosing n' as large as we wish, we get a ratio as close to $4/9$ as we want. ∎

How can one find a maximum triangular cactus quickly? A graphic matroid parity algorithm can be used to construct a maximum triangular cactus in a given graph [LP86]. The problem solved by a graphic matroid parity algorithm is GRAPHIC MATROID PARITY (GMP): given a multigraph $G' = (V', E')$ and a partition of the edge set E' into pairs of distinct edges $\{f, f'\}$, find a (simple) forest F with the maximum number of edges, such that $f \in F$ if and only if $f' \in F$ for all $f \in E'$.

Let us show how to reduce the problem of finding a maximum triangular cactus in a given graph $G = (V, E)$ to GMP. This is done by describing a multigraph $G' = (V', E')$ and a partition $\mathscr{P}$ of E' into pairs of distinct edges of E', such that, from a solution to GMP for G' and $\mathscr{P}$, we can construct a maximum triangular cactus in G.

First, let $V' = V$. Now, let us describe E' and the partition $\mathscr{P}$. Initially, $E' = \varnothing$ and $\mathscr{P} = \varnothing$. For each triangle in G with edge set T, let e, e' be any pair of distinct edges in T. Add two new edges f and f' to E', f with the same endpoints as e, and f' with the same endpoints as e'. We say that T *corresponds to* $\{f, f'\}$. Insert $\{f, f'\}$ into $\mathscr{P}$. The construction of E' and $\mathscr{P}$ is complete.

We say a forest F in G' is *valid* if $f \in F$ if and only if $f' \in F$ for all f in E'. Observe that any valid forest has an even number of edges. The following lemma states a relation between valid forests in G' and triangular cacti in G. Let m and n be the number of edges and vertices, respectively, in G.

LEMMA 2.13. *There is a valid forest F in G' with $2p$ edges if and only if there is a triangular cactus S in G with p triangles. Moreover, S can be obtained from F in time $O(n)$.*

Proof. Assume that S is a triangular cactus in G with p triangles. Note that an edge in S appears in exactly one cycle; otherwise that edge is a chord in a cycle of length at least 4. So if in each triangle we replace the three edges by the two edges f, f', we are left with a valid forest in G', with $2p$ edges.

Assume now that F is a valid forest in G' with $2p$ edges. Note that $2p \leq n - 1$. For each pair $\{f, f'\}$ in F, substitute f, f' by e, e' and add the third edge of the coresponding triangle. The substitution can be done in $O(1)$ time per pair, in a total time of $O(p)$, which is $O(n)$.

Call the resulting graph S. Next we prove that S is a triangular cactus with p triangles. Note that S is a subgraph of G. Each edge in S appears in a cycle: the corresponding triangle. In order to show that S is a cactus, we need to prove that each cycle has length 3. We can substitute f, f' by e, e' and add the third edge for each pair $\{f, f'\}$ one at a time, and we maintain the invariant that each cycle consists of $\{e, e'\}$, corresponding to a matched pair $\{f, f'\}$ of F, and the third edge of the corresponding triangle.

This property is true before substituting the first pair and adding any third edge, since F is a forest. Let us prove that we maintain the invariant after substituting f, f' by e, e' and adding e'' of a triangle whose edges are $\{e, e', e''\}$. We know that before substituting f, f' by e, e' and adding e'', f and f' did not appear in any cycle, by the invariant. We need to show that substituting f, f' by e, e' and adding e'' only creates the cycle e, e', e''. Because e is parallel to f and e' is parallel to f', any cycle created by the substitution of f, f' by e, e' and the addition of e'' has to contain e'' (otherwise f and f' would appear in a cycle).

Assume for a contradiction that there is a cycle C which is not e, e', e''. Assume first that C contains neither e nor e'. By replacing e'' by f and f' in C, we either get a cycle with f and f' and without e, e', e'', a contradiction, or a union of two cycles (this is the case in which C goes through the common endpoint of f and f'). But one of these cycles contains f and does not contain e, e', e'', a contradiction. If C contains e and e'' but not e', then by replacing e and e'' by f' in C we get a cycle containing f' and not e, e', e'', contradicting the invariant. The case in which C contains e' and e'' but not e is similar. It follows that S is a cactus.

For each two edges in F we get a triangle in S, for a total of p triangles. This completes the proof. ∎

As described by Chiba and Nishizeki [CN85], we can explicitly list all the triangles in a graph G with m edges in time $O(m^{3/2})$. So $|E'|$ is $O(m^{3/2})$.

Gabow and Stallmann [GS85] describe an algorithm for GMP, which runs in time $O(m'n' \log^6 n')$, where m' and n' are the number of edges and vertices, respectively, in the input graph. In our case, $n' = n$ and $m' = |E'|$, which is $O(m^{3/2})$. This gives a time bound of $O(m^{3/2} n \log^6 n)$ for this phase.

From the output of the Gabow–Stallmann algorithm, it is easy to find a maximum triangular cactus in time $O(n)$ (Lemma 2.13). Therefore the total time is $O(m^{3/2} n \log^6 n)$.

3. OUTERPLANAR SUBGRAPHS

Serendipitously, Algorithm B produces outerplanar graphs, so it is an approximation algorithm for MAXIMUM OUTERPLANAR SUBGRAPH, which is NP-complete [GJ79, p. 197].

In fact, any algorithm which produces a spanning tree has performance ratio at least $1/2$, because any outerplanar graph on $n \geq 2$ vertices has at most $2n - 3$ edges (see below). A careful analysis shows that the performance ratio of B when used for MAXIMUM OUTERPLANAR SUB-

GRAPH is at least $2/3$. This is an easy consequence of Theorem 3.2, in order to prove which we need some preliminaries.

An outerplanar graph G is a *maximal outerplanar graph* if no edge can be added without losing outerplanarity. As mentioned in [H72, p. 106], every maximal outerplanar graph G with at least three vertices is a triangulation of a polygon (i.e., the boundary of the exterior face is a Hamiltonian cycle and each interior face is triangular). By [H72, Corollary 11.9], G must have a vertex of degree 2 and $2|V(G)| - 3$ edges (this last statement is also true for $|V(G)| = 2$).

LEMMA 3.1. *Let H be an embedding of a maximal outerplanar graph as the triangulation of a polygon. If H has an odd number $n = 2p + 1$ of vertices, then there is a triangular cactus in H with p triangles. If H has an even number $n = 2p$ of vertices and xy is an edge on the boundary of the exterior face, then there is a triangular cactus S in H with $p - 1$ triangles such that in S, x and y are not connected.*

Notice that we obtain the maximum number of triangles possible. In the former case all vertices are in the same component of the cactus, while in the latter, the cactus has two components.

Proof. We use a plane embedding of H.

The proof is by induction on n, the number of vertices of H. The case $n = 1$ is trivial. If $n = 2$ (in this case there is only one edge and $p = 1$), the theorem is true.

We inductively construct a triangular cactus of the given size.

Let $n = 2p + 1$. Let u be a vertex of degree 2. Let x and y be its neighbors. They are adjacent, since interior faces are triangles. The graph $H - \{u\}$ is maximal outerplanar (since it has $(2n - 3) - 2 = 2(n - 1) - 3$ edges) and has an even number of vertices. It is easy to check that if a triangular cactus S' in this smaller graph has the property that x and y are not connected in S', we can add the triangle xyu to get a triangular cactus in H. The size of this cactus is $p - 1$, by induction, plus 1, for a total of p.

Let $n = 2p$ and let the edge xy be on the boundary of the exterior face. This edge is on the boundary of a triangular face xyv on the inside. Walking along the Hamiltonian cycle which is the boundary of the exterior face, starting at v and in the direction that visits x just before y, let D_1 be the set of vertices visited between v and x, inclusive, and let $n_1 = |D_1|$. Walking along the Hamiltonian cycle in the opposite direction again starting at v, let D_2 be the set of vertices visited between v and y, inclusive, and let $n_2 = |D_2|$; $D_1 \cap D_2 = \{v\}$ and $D_1 \cup D_2 = V(H)$. The only edge in H between $D_1 - \{v\}$ and $D_2 - \{v\}$ is the edge xy.

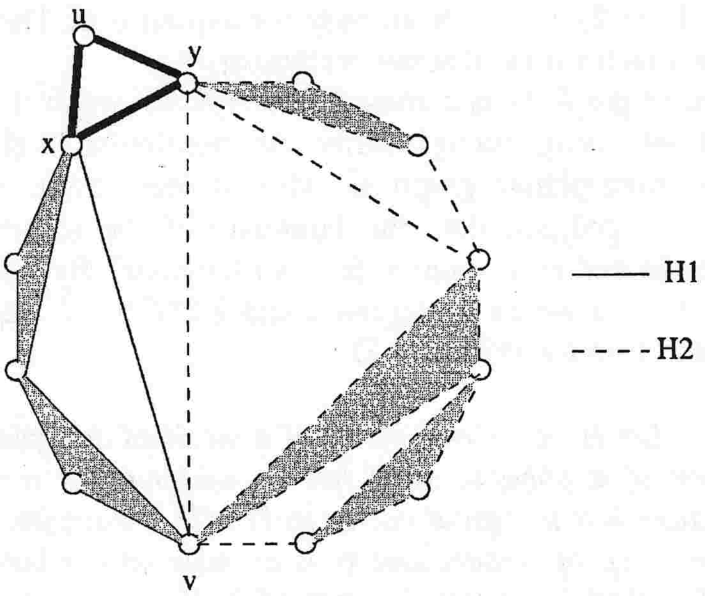


FIG. 2. Graphs H_1 and H_2, built from H. The shaded region gives an example of a triangular structure of the desired size, like the one constructed in the last paragraph of the proof of Lemma 3.1.

Let H_1 be the subgraph of H induced by vertex set D_1, with, say e_1 edges, and let H_2 be the subgraph of H induced by vertex set D_2, with, say, e_2 edges (see Fig. 2).

We have $n_1 + n_2$ is odd, since v is counted twice. Let us say without loss of generality that $n_1 = 2p_1 + 1$ is odd and $n_2 = 2p_2$ is even. Then $n = 2(p_1 + p_2)$. We have $e_1 + e_2 = (2n - 3) - 1$, as from H only the edge xy is not an edge of either H_1 or H_2. Since $e_1 \leq 2n_1 - 3$ and $e_2 \leq 2n_2 - 3$, we infer that $e_1 + e_2 \leq 2(n_1 + n_2) - 6 = 2(n + 1) - 6 = 2n - 4$. Since, in fact, $e_1 + e_2 = 2n - 4$, we infer that $e_1 = 2n_1 - 3$ and $e_2 = 2n_2 - 3$. Thus both H_1 and H_2 have to be maximal outerplanar, as they have the maximum allowable number of edges.

Then by the inductive hypothesis we can construct in H_1 a cactus S_1 with p_1 triangles. If we apply the inductive hypothesis to H_2 with vy being the edge on the exterior face, we obtain a triangular cactus S_2 with $p_2 - 1$ triangles in which y and v are not connected. Then, putting together the edges of S_1 and S_2, we get S, a cactus in H. In the new cactus S, any possible $x - y$-path must visit v, since neither S_1 nor S_2 has edge xy. But in S_2, y and v are not connected. It follows that x and y are not connected in S, so S is the desired cactus. S has $p_1 + (p_2 - 1)$ triangles, which is exactly the number we wanted. ▮

In conclusion, for a maximal outerplanar graph with n vertices, we can find a triangular structure with $\lfloor (n - 1)/2 \rfloor$ triangles.

We now prove a lower bound on $\rho(H)$.

THEOREM 3.2. *If H is outerplanar, then $\rho(H) \geq 2/3$.*

Proof. Let H be any 2-connected outerplanar graph. We add t edges to obtain a maximal outerplanar plane graph H'. Note that H' has $2n - 3$ edges and a triangular structure S with exactly $\lfloor (n-1)/2 \rfloor$ triangles.

However, the t missing edges can destroy at most t of these triangles in S, because S is a cactus. If $t \geq n/2$, we infer that

$$\rho(H) \geq \frac{n-1}{2n - 3 - n/2} \geq \frac{2}{3}.$$

Assume to the contrary that $t \leq \lfloor (n-1)/2 \rfloor$. Then the number of edges in the triangular structure S is at least $n - 1 + (\lfloor (n-1)/2 \rfloor - t)$. Thus

$$\rho(H) \geq \frac{n - 1 + \lfloor (n-1)/2 \rfloor - t}{2n - 3 - t}.$$

The worst case is achieved when $t = \lfloor (n-1)/2 \rfloor$ and is $2/3$.

If H is not 2-connected, we can do the above analysis for each of the 2-connected components (an edge appears in exactly one 2-connected component) and infer that a maximum triangular structure has $2/3$ of the edges in H. ∎

The theorem above is tight, in the sense that there are outerplanar graphs H for which $\rho(H)$ is arbitrarily close to $2/3$. In fact, there are outerplanar graphs H_i with $2i$ vertices and $3i - 2$ edges which do not have any triangle.

COROLLARY 3.3. *Algorithm* B *has performance ratio* $2/3$ *for MAXIMUM OUTERPLANAR SUBGRAPH.*

4. THE COMPLEXITY OF THE PROBLEMS

Papadimitriou and Yannakakis [PY91] defined a natural variant of NP for optimization problems: the complexity class Max SNP. This class, as they have shown, contains several well-known optimization problems, such as MAX 3-SAT and MAXIMUM CUT. In this section, we prove that MAXIMUM PLANAR SUBGRAPH (MPS) is Max SNP-hard, as its complementary version: given a graph, find a smallest subset of its edges whose removal results in a planar graph. This means, by results of Arora *et al.* [ALMSS92], that there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm for MPS with performance ratio at least $1 - \epsilon$ implies that $P = NP$, and that an analogous statement can be made about the complementary problem.

As in [PY91], we use the concept of *L-reduction*, which is a special kind of reduction that preserves approximability. Let A and B be two optimization problems. We say *A L-reduces to B* if there are two polynomial-time algorithms f and g and positive constants α and β, such that for each instance I of A,

1. Algorithm f produces an instance $I' = f(I)$ of B, such that the optima of I and I', of costs denoted $Opt_A(I)$ and $Opt_B(I')$, respectively, satisfy $Opt_B(I') \leq \alpha \cdot Opt_A(I)$.

2. Given any feasible solution of I' with cost c', algorithm g produces a solution of I with cost c such that $|c - Opt_A(I)| \leq \beta \cdot |c' - Opt_B(I')|$.

The main result of this section is

THEOREM 4.1. *MAXIMUM PLANAR SUBGRAPH is Max SNP-hard.*

Proof. Denote by $TSP_4(1,2)$ the following variant of the traveling salesman problem: given a complete graph, a pair of distinct vertices x, y, and cost 1 or 2 for each edge, such that the graph induced by the edges of cost 1 has maximum degree at most 4, find a Hamiltonian path from x to y of minimum cost. Papadimitriou and Yannakakis [PY93] showed that $TSP_4(1,2)$ is Max SNP-hard.

We shall prove $TSP_4(1,2)$ L-reduces to MPS. The basic idea of the reduction comes from Liu and Geldmacher [LG77], where the decision version of MPS is proved to be NP-complete.

The first part of the L-reduction is the polynomial-time algorithm f and the constant α. Given any instance I of $TSP_4(1,2)$, f produces an instance G of MPS such that the cost of the optimum of G in MPS, denoted $Opt_{MPS}(G)$, is at most α times the cost of the optimum of I in $TSP_4(1,2)$, denoted by $Opt_{TSP_4(1,2)}(I)$, that is, $Opt_{MPS}(G) \leq \alpha \cdot Opt_{TSP_4(1,2)}(I)$.

Consider an instance I of $TSP_4(1,2)$. I is a complete graph $K = (V, E)$, a pair of distinct vertices x, y of V, and a subset E_1 of E consisting of the edges of cost 1. Let $H = (V, E_1)$ and $H' = (V \cup T, E_1 \cup F_1 \cup F_2)$, where $T = \{t_0, t_1, t_2, t_3\}$, $T \cap V = \emptyset$, $F_1 = \{t_0t_1, t_0t_3, t_1t_2, t_1t_3, t_1x, t_2t_3, t_3y\}$, and $F_2 = \bigcup_{z \in V}\{t_0z, t_2z\}$ (see Fig. 3).

Denoting by n the number of vertices of H, let G be the graph obtained from H' by (1) replacing each edge e in F_1 by $2n$ parallel internally disjoint paths of length 2 (having new internal vertices) between the endpoints of e and (2) replacing each edge e in F_2 by eight parallel internally disjoint paths of length 2 (having new internal vertices) between the endpoints of e (see Fig. 4).

Clearly, G can be obtained from I in time polynomial to the size of I.

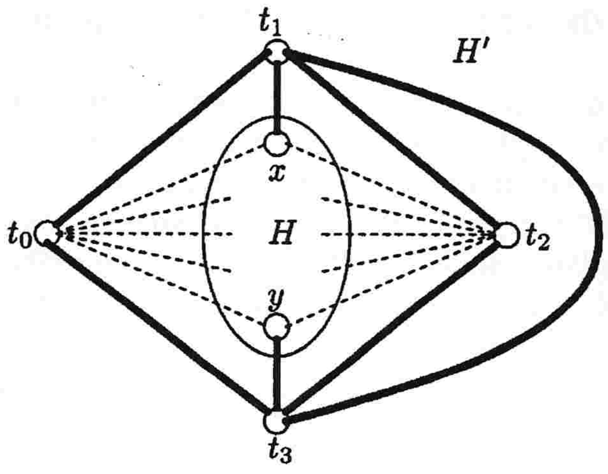


FIG. 3. Graph H' constructed from H.

LEMMA 4.2. $Opt_{MPS}(G) \leq 124 \cdot Opt_{TSP_4(1,2)}(I)$.

Proof. Observe that $Opt_{TSP_4(1,2)}(I) \geq n - 1$. A clear upper bound for $Opt_{MPS}(G)$ is the number of edges of G. To compute this, note first that H has maximum degree at most 4 by the definition of $TSP_4(1,2)$, and so H has at most $2n$ edges. Let us call the edges in F_1 $2n$-*edges* and the edges in F_2 8-*edges*. There are seven $2n$-edges: F_1 contains seven edges, each of them corresponding to $4n$ edges in G. There are $2n$ 8-edges: F_2 contains $2n$ edges, each of them corresponding to 16 edges in G. Hence the number of edges in G outside of H is $7 \cdot 4n + 16 \cdot 2n = 60n$. The total number of edges in G is therefore at most $2n + 60n = 62n \leq 124(n-1)$. Therefore $Opt_{MPS}(G) \leq 124(n-1) \leq 124 \cdot Opt_{TSP_4(1,2)}(I)$. ∎

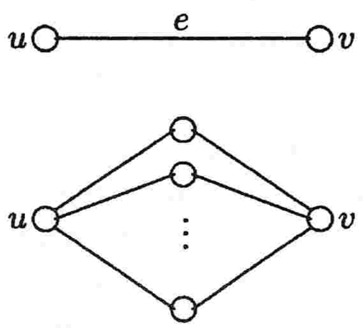


FIG. 4. Edge e and its substitution by internally disjoint paths of length 2.

This finishes the first part of the L-reduction, since we can take $\alpha = 124$.

The second and hard part of the L-reduction is the constant β and the algorithm g. Given a planar subgraph of G with m edges, g produces in polynomial time a Hamiltonian path from x to y of cost t in K such that $|t - Opt_{TSP_4(1,2)}(I)| \leq \beta|m - Opt_{MPS}(G)|$. We shall see that $\beta = 1$ suffices.

First, given a planar subgraph P of G, let us describe another planar subgraph P' of G with at least as many edges as P. Moreover, P' shall contain all edges of G not in H.

Let e be a $2n$-edge or an 8-edge of H'. We say *e appears in P* if P contains both edges of *all* the paths of length 2 corresponding to e. In this case, we also say that the endpoints of *e are adjacent in P by the $2n$-edge* or *8-edge e*. We say *e is missing in P* if P contains both edges of *none* of the paths (of length 2) corresponding to e. (In this case, if e is an 8-edge, then P is missing at least eight of its 16 edges in G corresponding to e.) It is possible that a $2n$-edge or an 8-edge of H' neither appears in P nor is missing in P.

Let us modify P so that any $2n$-edge or 8-edge of H' either appears in P or is missing in P. This is done as follows. If a $2n$-edge or an 8-edge e of H' neither appears nor is missing in P, then we insert in P all edges of G in the paths (of length 2) corresponding to e. Note that P remains planar. Clearly, the number of edges in P cannot decrease by this operation. The new graph is also called P.

Now we can describe P'. We have three cases: (1) if some $2n$-edge e does not appear in P, then define P' to be the graph induced by all edges of G not in H; (2) if all the $2n$-edges and the 8-edges of H' appear in P, then let P' be the same as P; and (3) if all the $2n$-edges of H' appear in P, but not all the 8-edges, then we modify P to obtain P', as described in the next two paragraphs.

The idea is to remove from P some edges of H and add to P edges of H' not in H so that all the 8-edges of H' appear in the modified graph, and it remains planar and has at least as many edges the original P.

Let U be the set of vertices v of H such that at least one of the two 8-edges incident to v in H' is missing in P. Observe that $|U| \geq 1$, as case (2) considered $|U| = 0$. For each vertex v in U, remove from P all edges of H incident to v in P (at most four edges are removed per vertex) and add to P all the edges outside of H so that the two 8-edges incident to v appear in P (at least eight edges are added, corresponding to the 8-edges incident to v missing in P). To guarantee that the graph obtained this way is planar, we must make room to embed the modified 8-edges. This is done by also removing from P all edges of H incident to y (if they were not already removed). Let P' be the graph obtained after all these modifications.

LEMMA 4.3. *P' is planar and has at least as many edges as P.*

Proof. In case (1), we include in P' at least $2n$ edges that do not appear in P (at least one in each of the $2n$ paths corresponding to e), and we remove at most $2n$ edges, the maximum number of edges in H. So P' has at least as many edges as P. Moreover P' is planar.

There is nothing to be proved in case (2).

Case (3) is the complicated one. First, note that P' has at least as many edges as G, since, for each vertex in U, we remove at most four edges and add at least eight. Furthermore, we remove at most four edges incident to y. Hence we gain at least $(8-4)|U| - 4 = 4|U| - 4 \geq 0$ edges, since $|U| \geq 1$.

Now, let us show that P' is planar. We can think of the $2n$-edges and 8-edges as single edges, as they are in H' (since if we can embed a single edge, we can embed a $2n$-edge or an 8-edge as well). We will modify a given embedding of P into an embedding for P'.

Let C be the cycle (using four $2n$-edges) t_0, t_1, t_2, t_3, t_0. Observe that the $2n$-edges in C appear in P, since we are in case (3). Given an embedding for P, cycle C divides the plane into two regions, R_1, containing the $2n$-edge t_1t_3, and R_2 (see Fig. 5). The $2n$-edge t_1t_3 separates t_0 from t_2 in R_1. Moreover, each vertex in $V - U$ (the vertices of H not in U) is adjacent in P by 8-edges to t_0 and t_2. Because t_0 and t_2 are separated in R_1, none of these vertices can be embedded in R_1, which implies they must be embedded in R_2. Keep these vertices (t_0, t_1, t_2, t_3 and the vertices in $V - U$) embedded as they are.

Now, observe that y is adjacent in P' only to t_0, t_2 (by 8-edges), and t_3 (by a $2n$-edge). Furthermore, y is the only vertex in H which is adjacent to t_3. This means, before we embed y, the vertices t_0, t_2, and t_3 are not

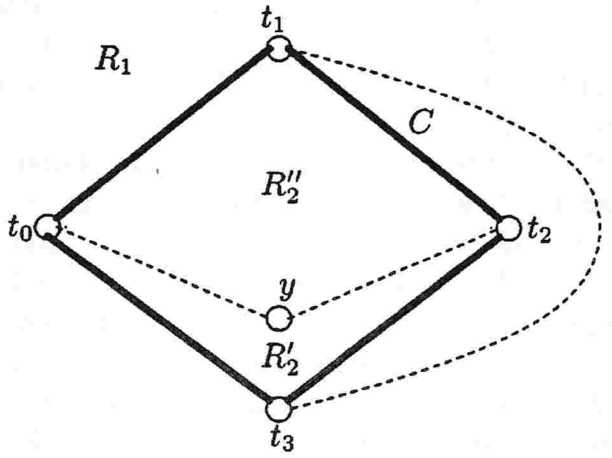


FIG. 5. Cycle C, regions R_1 and $R_2 = R_2' \cup R_2''$.

separated in R_2. Therefore y can be embedded in R_2 with the $2n$-edge t_3y and the 8-edges t_0y and yt_2 "next to" the $2n$-edges t_0t_3 and t_3t_2. The edges t_0y and yt_2 together split region R_2 into two regions, R'_2, containing edge t_3y, and R''_2, containing vertex t_1. Observe that t_0 and t_2 are not separated in R''_2, since t_3 is the only vertex, besides t_0 and t_2, which is adjacent to y (by a $2n$-edge).

All the vertices in $U - \{x\}$ are adjacent in P' only to t_0 and t_2 (by 8-edges), and therefore they all can be embedded in R''_2 with their two 8-edges "next to" the 8-edges t_0y and yt_2. If $x \in U$ then observe that x is the only vertex in H which is adjacent in P' to t_1 (by a $2n$-edge). This means, before we embed x, the vertices t_0, t_1, and t_2 are not separated in R''_2. Therefore x can be embedded in R''_2 with the $2n$-edge xt_1 and the 8-edges t_0x and xt_2 "next to" the $2n$-edges t_0t_1 and t_1t_2. If $x \notin U$ then it does not need to be moved in the embedding. The embedding obtained this way is a plane embedding of P', completing the proof that P' is planar. ∎

Observe that P' contains all the edges of G not in H. Let F be the set of edges of H appearing in P'.

LEMMA 4.4. *The graph $G_F = (V, F)$ is a collection of vertex-disjoint paths which can be extended in K (the complete graph on V) to a Hamiltonian path from x to y, in polynomial time.*

Proof. Let us prove that G_F satisfies the following four conditions: (1) There is no vertex of degree greater than 2 in G_F. (2) Vertices x and y have degree at most 1 in G_F. (3) There is no cycle in G_F. (4) If x and y are in the same component of G_F, then this component spans all vertices in V. We will prove each of these conditions holds by contradiction.

Suppose (1) does not hold. Let z_0 be a vertex in V of degree at least 3 in G_F. Let z_1, z_2, z_3 be three of its neighbors in G_F. (Notice that z_0, z_1, z_2, z_3 are distinct vertices of H, so they are distinct of t_0, t_2.) Then each one of t_0, t_2, z_0 is adjacent in P' to each one of z_1, z_2, z_3 (some of them are adjacent in P' by 8-edges). Therefore $t_0, t_2, z_0; z_1, z_2, z_3$ define a subdivision of $K_{3,3}$ in P', a contradiction, because P' is planar. Thus (1) holds.

Suppose (2) does not hold. If x has degree more than 1 in G_F, let z_1 and z_2 be two of its neighbors in G_F. (Notice that z_1 and z_2 are distinct vertices of H, distinct of x, so they are distinct of t_0, t_2.) Then each one of t_0, t_2, x is adjacent in P' to each one of t_1, z_1, z_2 (some of them are adjacent in P' by $2n$-edges or 8-edges). Therefore $t_0, t_2, x; t_1, z_1, z_2$ define a subdivision of $K_{3,3}$ in P', a contradiction, because P' is planar. Analogously, we have a contradiction if y has degree more than 1 in G_F. Thus (2) holds.

Suppose (3) does not hold. Let z_1, z_2, z_3 be three vertices in a cycle of G_F. (Since z_1, z_2, z_3 must have degree at least 2, they are not x or y by condition (2), and they are not t_0 or t_2 since they are vertices of H.) Then z_1, z_2, z_3, t_0, t_2 are pairwise linked by internally vertex-disjoint paths (the path between t_0 and t_2 uses the two 8-edges incident to x, while the others use one $2n$-edge or 8-edge). Therefore, t_0, t_2, z_1, z_2, z_3 define a subdivision of K_5 in P', a contradiction, because P' is planar. Hence (3) holds.

Suppose (4) does not hold. Let z_0 be a vertex in V which is not in the component having x and y in G_F. In this case, t_0, t_1, t_2, x, y are pairwise linked by internally vertex-disjoint paths (the path between t_0 and t_2 uses z_0, the path between t_1 and y uses t_3, the path between x and y is in G_F, and the others use one $2n$-edge or 8-edge). Therefore, t_0, t_1, t_2, x, y define a subdivision of K_5 in P', a contradiction, because P' is planar. Hence (4) holds.

Therefore, the conditions hold. From (1) and (2), we conclude that G_F is a collection of paths. From (2) and (3), these paths can be extended in K (the complete graph on V) to a Hamiltonian path from x to y. Furthermore, note that this can be done in polynomial time. ▮

Let HP be a Hamiltonian path from x to y containing all edges in F and some edges (in K) of cost 2. HP exists by Lemma 4.4. Denote by m' the number of edges of P' and by t the cost of HP.

Now the following lemma states that β exists, and specifically, $\beta = 1$.

LEMMA 4.5.

$$t - Opt_{TSP_4(1,2)}(I) = Opt_{MPS}(G) - m'$$

and hence

$$|t - Opt_{TSP_4(1,2)}(I)| = |m' - Opt_{MPS}(G)|.$$

Proof. As in the proof of Lemma 4.2, the number of edges in G outside of H is $60n$. All these edges are in P'. Therefore the number of edges in F is $m' - 60n$. And the cost of HP is

$$t = 2(n-1) - (m' - 60n). \tag{9}$$

Let Q be an optimal solution of MPS for G. Using the same argument for Q that we used for P, Lemma 4.4 and the argument above imply the existence of a Hamiltonian path of cost $2(n-1) - (Opt_{MPS}(G) - 60n)$. Therefore

$$Opt_{TSP_4(1,2)}(I) \leq 2(n-1) - (Opt_{MPS}(G) - 60n). \tag{10}$$

Given an optimal solution HP^* of $TSP_4(1,2)$ for I, we can construct a solution of MPS for G by selecting all the edges outside of H plus the edges of cost 1 in HP^*. Observe that these edges really determine a planar subgraph of G. Let z be the number of these edges. Since we have $60n$ edges in G outside of H, HP^* has $z - 60n$ edges of cost 1 and the remaining ones (of its $n-1$ edges) have cost 2. This means that

$$\begin{aligned} Opt_{TSP_4(1,2)}(I) &= 2(n-1) - (z - 60n) \\ &\geq 2(n-1) - (Opt_{MPS}(G) - 60n), \qquad (11) \end{aligned}$$

since $Opt_{MPS}(G) \geq z$.

Therefore, from (10) and (11), we have

$$Opt_{TSP_4(1,2)}(I) = 2(n-1) - (Opt_{MPS}(G) - 60n).$$

And this together with (9) means

$$t - Opt_{TSP_4(1,2)}(I) = Opt_{MPS}(G) - m'.$$

Hence $|t - Opt_{TSP_4(1,2)}(I)| = |m' - Opt_{MPS}(G)|$. ∎

From $m \leq m'$, it follows that

$$t - Opt_{TSP_4(1,2)}(I) \leq Opt_{MPS}(G) - m$$

and

$$|t - Opt_{TSP_4(1,2)}(I)| \leq 1 \cdot |m - Opt_{MPS}(G)|.$$

This completes the proof of Theorem 4.1. ∎

Let us denote the complementary version of MPS by NPD: given a graph G, find a smallest set of edges of G whose removal results in a planar graph.

A slight modification of the L-reduction presented above proves the following.

THEOREM 4.6. *NPD is Max SNP-hard.*

Proof. The first part of the L-reduction is almost the same. From an instance I of $TSP_4(1,2)$, we construct G in exactly the same way. As before, $Opt_{TSP_4(1,2)}(I) \geq n-1$. As in the proof of Lemma 4.2, the maximum number of edges of G is $62n$. Thus the optimum of G in NPD, denoted as $Opt_{NPD}(G)$, is at most $62n$. And then $Opt_{NPD}(G) \leq 62n \leq 124(n-1) \leq 124 \cdot Opt_{TSP_4(1,2)}(I)$. We can take $\alpha = 124$, as before.

In the second part, given an instance I of $TSP_4(1,2)$, let G be constructed from I as in the previous reduction. Let D be a subset of the edges of G whose removal results in a planar subgraph of G. We shall find in polynomial time a Hamiltonian path HP such that $t - Opt_{TSP_4(1,2)}(I) \leq d - Opt_{NPD}(G)$, where t is the number of edges in HP and $d = |D|$. Just as we took a planar subgraph P of G and found a planar subgraph P' which contains all edges of G not in H, and is at least as large as P, from D we can find a set D' of edges of G containing *none* of the edges of G not in H, which is at least as *small* as D. Applying Lemma 4.4, we can obtain a Hamiltonian path HP, as before, in polynomial time.

Now, let us prove that β exists. Let m' be the number of edges in P'. Note that $d' + m' = |E(G)|$. Moreover, $Opt_{MPS}(G) + Opt_{NPD}(G) = |E(G)|$. Therefore, $d' - Opt_{NPD}(G) = Opt_{MPS}(G) - m'$. Applying Lemma 4.5, we conclude that $t - Opt_{TSP_4(1,2)}(I) = d' - Opt_{NPD}(G)$, which, together with $d' \leq d$, implies that we can take $\beta = 1$. ▮

5. OPEN PROBLEMS

Many open problems are suggested by this research. How large a performance ratio one can achieve is an obvious one. Is there a linear-time approximation algorithm for MAXIMUM PLANAR SUBGRAPH with performance ratio $1/3 + \epsilon$? (A *maximal* planar subgraph can be found in linear time [H95, D95].) Is there any approximation algorithm with a constant performance ratio for NPD? What performance ratio can be achieved for THICKNESS (given G, partition the edges of G into as few planar subgraphs as possible)? A factor of 3 here is trivial, via arboricity.

ACKNOWLEDGMENTS

The authors are very grateful to David Williamson for his help. We thank Laci Lovász for informing us that a maximum triangular structure can be found in polynomial time. We thank Robert Cimikowski for kindly sending us drafts of his papers. The third author would like to thank Kurt Mehlhorn, for suggesting the problem and supporting his work. We are also very grateful to Robin Thomas for helpful discussions, and specifically for providing a simpler proof of Claim 2.8. We also would like to thank the program committee of SODA'96 for informing us about a faster way to find all triangles in a graph.

REFERENCES

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, Proof verification and hardness of approximation problems, *in* "Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science," pp. 14–23, 1992.

[BM76] J. A. Bondy and U. S. R. Murty, "Graph Theory with Applications," Macmillan, New York, 1976.

[CHT93] J. Cai, X. Han, and R. E. Tarjan, An $O(m \log n)$-time algorithm for the maximal planar subgraph problem, *SIAM J. Comput.* **22** (1993), 1142–1162.

[Cim95] R. Cimikowski, An analysis of some heuristics for the maximum planar subgraph problem, *in* "Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms," pp. 322–331, 1995.

[CNS79] T. Chiba, I. Nishioka, and I. Shirakawa, An algorithm of maximal planarization of graphs, *in* "Proceedings of the IEEE Symposium on Circuits and Systems," pp. 649–652, 1979.

[CN85] N. Chiba and T. Nishizeki, Arboricity and subgraph listing algorithms, *SIAM J. Comput.* **14** (1985), 210–223.

[D95] H. N. Djidjev, A linear algorithm for the maximal planar subgraph problem, *in* "Proceedings of the Fourth International Workshop on Algorithms and Data Structures (WADS '95)," pp. 369–380, 1995.

[DFF85] M. E. Dyer, L. R. Foulds, and A. M. Frieze, Analysis of heuristics for finding a maximum weight planar subgraph, *European J. Oper. Res.* **20** (1985), 102–114.

[F92] L. R. Foulds, "Graph Theory Applications," Springer-Verlag, New York, 1992.

[GS85] H. N. Gabow and M. Stallmann, Efficient algorithms for graphic matroid intersection and parity, *in* "Automata, Language and Programming: 12th Colloq.," Lecture Notes in Computer Science, Vol. 194, pp. 210–220, 1985.

[GJ97] M. R. Garey and D. S. Johnson, "Computers and Intractability," Freeman, New York, 1979.

[H72] F. Harary, "Graph Theory," Addison–Wesley, Reading, MA, 1972.

[H95] W. Hsu, A linear time algorithm for finding maximal planar subgraphs, *in* "Proceedings of the Sixth Annual International Symposium on Algorithms and Computation (ISAAC95)," 1995.

[JM93] M. Jünger and P. Mutzel, Maximum planar subgraph and nice embeddings: Practical layout tools, *Algorithmica*, to appear.

[LG77] P. C. Liu and R. C. Geldmacher, On the deletion of nonplanar edges of a graph, *in* "Proceedings of the 10th Southeastern Conference on Combinatorics, Graph Theory, and Computing," pp. 727–738, 1977.

[LP86] L. Lovász and M. D. Plummer, "Matching Theory," Elsevier, Amsterdam, 1986.

[PY91] C. H. Papadimitriou and M. Yannakakis, Optimization, approximation, and complexity classes, *J. Comput. System Sci.* **43** (1991), 425–440.

[PY93] C. H. Papadimitriou and M. Yannakakis, The traveling salesman problem with distances one and two, *Math. Oper. Res.* **18**(1) (1993), 1–11.

[TDB88] R. Tamassia, G. Di Battista, and C. Batini, Automatic graph drawing and readability of diagrams, *IEEE Trans. Systems Man Cybernet.* **18** (1988), 61–79.

Chapter 1
A Better Approximation Algorithm for Finding Planar Subgraphs

Gruia Călinescu [*] Cristina G. Fernandes [†] Ulrich Finkler [‡] Howard Karloff [*]

Abstract

The MAXIMUM PLANAR SUBGRAPH problem—given a graph G, find a largest planar subgraph of G—has applications in circuit layout, facility layout, and graph drawing. No previous polynomial-time approximation algorithm for this NP-Complete problem was known to achieve a performance ratio larger than 1/3, which is achieved simply by producing a spanning tree of G. We present the first approximation algorithm for MAXIMUM PLANAR SUBGRAPH with higher performance ratio (2/5 instead of 1/3). We also apply our algorithm to find large outerplanar subgraphs. Last, we show that both MAXIMUM PLANAR SUBGRAPH and its complement, the problem of removing as few edges as possible to leave a planar subgraph, are Max SNP-Hard.

1 Introduction

MAXIMUM PLANAR SUBGRAPH is this problem: given a graph G, find a planar subgraph of G of maximum size, where *size* is the number of edges. This problem has applications in circuit layout, facility layout, and graph drawing [F92, TDB88].

MAXIMUM PLANAR SUBGRAPH is known to be NP-Complete [LG77]. For a graph G, let us define $Opt(G)$ to be the maximum size of a planar subgraph of G. Given an algorithm A that takes (representations of) graphs G as input and outputs subgraphs of G, define $\mathrm{A}(G)$ to be the size of the planar graph A produces when G is the input. Now let us define A's *performance* or *approximation ratio* $r(\mathrm{A})$ to be the infimum, over all (representations of) graphs G, of $\mathrm{A}(G)/Opt(G)$ (if $Opt(G) > 0$, and 1 otherwise). In the literature, authors sometimes ensure that their performance ratio is at least one by defining it to be the reciprocal of ours.

Numerous approximation algorithms for MAXIMUM PLANAR SUBGRAPH appear in the literature, the simplest ones being Spanning Tree (output any spanning tree of G, assuming G is connected) and Maximal Planar Subgraph (output any planar subgraph to which the addition of any new edge would violate planarity). Spanning Tree is known to have performance ratio 1/3 (see below). Dyer, Foulds and Frieze [DFF85] proved that Maximal Planar Subgraph has performance ratio 1/3. Cimikowski [Cim95] proved that a path embedding heuristic of Chiba, Nishioka and Shirakawa [CNS79] and an edge embedding heuristic of Cai, Han and Tarjan [CHT93] have performance ratios not exceeding 1/3. In the same paper, Cimikowski studied two other polynomial-time heuristics: the "vertex-addition heuristic" and the "cycle-packing heuristic." The performance ratio of the former, to the authors' knowledge, is not known, whereas for the cycle-packing algorithm, it is 0. Dyer, Foulds and Frieze [DFF85] studied two other algorithms and proved that each has performance ratio at most 2/9. Also see [JM93].

In short, to the authors' knowledge, no previously proposed algorithm was known to have a performance ratio exceeding 1/3. What makes the problem more tantalizing is that achieving a performance ratio of 1/3 is trivial. In fact, Spanning Tree has performance ratio 1/3, since every spanning tree of a connected graph on n vertices has $n-1$ edges and every planar graph on n vertices has at most $3n-3 = 3(n-1)$ edges (and there are planar graphs on n vertices with $3n-6$ edges, for all $n \geq 3$). No previous algorithm could beat the bound achieved by a trivial algorithm.

In this paper, we present two new approximation algorithms for MAXIMUM PLANAR SUBGRAPH. Each achieves a performance ratio exceeding 1/3. The higher performance ratio is $2/5 = 0.4$ and is achieved by an algorithm which (surprisingly) invokes an algorithm for the graphic matroid parity problem as a subroutine and which runs in time $O(m^{3/2} n \log^6 n)$. A greedy variant still has performance ratio $7/18 = 0.3888...$, and runs in linear time on graphs of bounded degree.

Next, we provide an extension of the main algorithm. We provide a nontrivial approximation algorithm for MAXIMUM OUTERPLANAR SUBGRAPH, which is this problem: given G, find an outerplanar sub-

[*]College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280. Research supported in part by NSF grant CCR-9319106.

[†]College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280. Research supported in part by the CNPq (Brazil), under contract 200975/92-7.

[‡]Max-Planck-Institut für Informatik, D-66123 Saarbrücken, Germany. Research supported in part by the Graduiertenkolleg Effizienz und Komplexität von Algorithmen und Rechenanlagen, Universität Saarbrücken.

graph of G of maximum size. (An *outerplanar* graph is a graph which can be drawn in the plane without crossing edges, with all vertices on the boundary of the exterior face [H72].) This new algorithm has performance ratio at least $2/3$, which surpasses the bound of $1/2$ which is trivially obtained by producing a spanning tree.

Last, we show that MAXIMUM PLANAR SUBGRAPH is Max SNP-Hard, implying that there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at least $1 - \epsilon$ would imply that $P = NP$ [ALMSS92]. In addition, we show that the complementary problem, called NONPLANAR DELETION or NPD—given $G = (V, E)$, produce a smallest subset $L \subseteq E$ such that $(V, E - L)$ is planar—is also Max SNP-Hard.

2 The Approximation Algorithms

In this section we present the two new algorithms for MAXIMUM PLANAR SUBGRAPH. The higher performance ratio is at least $2/5$=0.4.

Let us give some motivation for our algorithm. As we said, given a (connected) graph G, an algorithm which outputs a spanning tree of G achieves a performance ratio of $1/3$. A graph whose cycles all have length three, i.e., are triangles, is planar, as it cannot contain a subdivision of K_5 or $K_{3,3}$. Moreover, note that a connected spanning subgraph of G whose cycles are triangles, besides being planar, has one more edge per triangle than a spanning tree of G.

Our better algorithm produces a subgraph of G whose cycles are triangles and, among these subgraphs, has the maximum number of edges. It can be implemented in time $O(m^{3/2} n \log^6 n)$, where m is the number of edges in G and n is the number of vertices in G, using a graphic matroid parity algorithm, as we will see later. We first present a greedy version of the algorithm.

2.1 A Greedy Version of the Algorithm.

Algorithm A, presented below, is a greedy version of our algorithm. It has a performance ratio of $7/18$=0.3888.... After presenting the algorithm and proving its performance ratio is $7/18$, we will show it can be implemented in linear time for graphs with bounded degree. We begin with some definitions.

A *triangular cactus* is a graph whose cycles (if any) are triangles and such that all edges appear in some cycle. A *triangular cactus in a graph G* is a subgraph of G which is a triangular cactus.

A *triangular structure* is a graph whose cycles (if any) are triangles. A *triangular structure in a graph G* is a subgraph of G which is a triangular structure. Note that every triangular cactus is a triangular structure, but not *vice versa*.

Algorithm A produces a triangular structure in the given graph G. The algorithm consists of two phases. First, A greedily constructs a maximal triangular cactus S_1 in G. Second, A extends S_1 to a triangular structure S_2 in G by adding as many edges as possible to S_1 without forming any new cycles.

Given a graph $G = (V, E)$ and $E' \subseteq E$, we denote by $G[E']$ the spanning subgraph of G induced by E', that is, the graph (V, E').

Algorithm A

```
Starting with $E_1 = \emptyset$, repeatedly (as long as possible) find a triangle $T$ whose vertices are in different components of $G[E_1]$, and add the edges of $T$ to $E_1$.
Let $S_1 := G[E_1]$.
Starting with $E_2 = E_1$, repeatedly (as long as possible) find an edge $e$ in $G$ whose endpoints are in different components of $G[E_2]$, and add $e$ to $E_2$.
Let $S_2 := G[E_2]$.
Output $S_2$.
```

Note that S_2 is indeed a triangular structure in G. As we mentioned before, S_2 is planar since it does not contain cycles of length greater than three.

THEOREM 2.1. *The performance ratio of algorithm A is $\frac{7}{18}$.*

Proof. First let us show that the performance ratio is at least $7/18$. Without loss of generality, we may assume G is connected, and has at least three vertices. Observe that the number of edges in S_2 is the number of edges in a spanning tree of G plus the number of triangles in S_1. So it suffices to count the number of triangles in S_1.

Let H be a maximum planar spanning subgraph of G. Let $n \geq 3$ be the number of vertices in G, and $t \geq 0$ be such that $3n - 6 - t$ is the number of edges in H. We can think of t as the number of edges missing for an embedding of H to be a triangulated plane graph. The number of triangular faces in H is at least $2n - 4 - 2t$. (This is a lower bound on the number of triangular faces of a plane embedding of H since if H were triangulated, it would have $2n - 4$ triangular faces, and each missing edge can destroy at most two of these triangular faces.)

Let k be the number of components of S_1 each with at least one triangle, and let $p_1, p_2, \ldots, p_k$ be the number of triangles in each of these components. Let $p = \sum_{i=1}^{k} p_i$. We will prove that p, the number of triangles in S_1, is at least a constant fraction of $n-2-t$. Note that if a triangle cannot be added to S_1, it is because two of its vertices are in the same component of S_1. Hence, one of its edges has its two endpoints in the same component of S_1. This means that at the end of the first phase, every triangle in G must have some two vertices in the same component of S_1. In particular, every triangular face in H must have some two vertices in the same component of S_1, and therefore one of its three edges must be in the subgraph of H induced by the vertices in a component of S_1. Thus we can associate with each triangular face F in H an edge e in F whose endpoints are in the same component of S_1. But any edge e in H lies in at most two triangular faces of H, so e could have been chosen by at most two triangular faces of H. It follows that the number of triangular faces in H is at most twice the number of edges in H whose endpoints are in the same component of S_1.

Let H' be the subgraph of H induced by the edges of H whose endpoints are in the same component of S_1. Note that $p_i \geq 1$, for all i, and that the number of vertices in the i^{th} component of S_1 is $2p_i + 1 \geq 3$. Since H' is planar, H' has at most $\sum_{i=1}^{k}(3(2p_i + 1) - 6) = 6p - 3k$ edges. By the observation at the end of the previous paragraph, $2(6p - 3k) \geq 2|E(H')| \geq$ (number of triangular faces in H) $\geq 2n - 4 - 2t$. From this, we have

$$p \geq \frac{n-2-t+3k}{6} \geq \frac{n-2-t}{6}.$$

Therefore the number of triangles in S_1 is at least $\frac{n-2-t}{6}$, and the ratio between the number of edges in S_2 and the number of edges in H is at least

$$\frac{n-1+\frac{n-2-t}{6}}{3n-6-t} = \frac{7n-8-t}{18n-36-6t} \geq \frac{7}{18},$$

since $t \geq 0$. This completes the proof that the performance ratio of algorithm A is at least 7/18.

Now, we will prove that the performance ratio is at most 7/18. This is done by presenting, for any $\epsilon > 0$, a *planar* graph G_ϵ such that algorithm A, with G_ϵ as input, can produce a subgraph S_2 of G_ϵ such that the number of edges in S_2 is at most $\frac{7}{18}+\epsilon$ times the number of edges in G_ϵ.

Given $\epsilon > 0$, let p be an integer such that $p > \frac{6\epsilon+1}{12\epsilon}$. Let S be any connected triangular cactus with p triangles. S has $2p+1 \geq 3$ vertices. Let S' be any triangulated plane supergraph of S on the same set of vertices (S' can be obtained from S by adding edges to S until it becomes triangulated). Since S' is triangulated, S' has $2(2p+1)-4 = 4p-2$ (triangular) faces. For each face of S', add a new vertex in the face and adjacent to all vertices in the boundary of that face. Let G_ϵ be the new graph. Observe that G_ϵ is a triangulated plane graph and has $(2p+1)+(4p-2) = 6p-1$ vertices. This means that G_ϵ has $3(6p-1)-6 = 18p-9$ edges. With G_ϵ as input for algorithm A, in the first phase it can produce $S_1 = S$, and S_2 can be S plus one edge for each of the new vertices (the vertices in G_ϵ not in S). The number of edges in S is $3p$. Hence, S_2 can have $3p+(4p-2) = 7p-2$ edges, while G_ϵ has $18p-9$ edges. Thus, the ratio between the number of edges in S_2 and the number of edges in G_ϵ is

$$\frac{7p-2}{18p-9} < \frac{7}{18} + \epsilon,$$

because $p > \frac{6\epsilon+1}{12\epsilon}$. ■

2.1.1 Linear Time for Bounded-Degree Graphs. In the case G has bounded degree d, we can implement algorithm A in linear time. We will only describe the implementation of the first phase, as the second one can clearly be implemented in linear time.

At any time, the vertices of the graph are partitioned in three sets: new, active and used. At the beginning, all the vertices are new. If there are no active vertices, choose a new vertex and make it active. Choose an active vertex x and "use" it; that is, include in the cactus S_1, one after the other, triangles formed by x and two new vertices, making these vertices active. Mark x "used" at the end of this process.

Using one vertex takes constant time as all degrees are bounded by d. We maintain the invariant that all triangles which contain a used vertex have been processed and all vertices which are active at a given time are in the same connected component of $G[E_1]$ at that time.

It is not hard to see that at the end, E_1 is maximal, in that no triangles can be added to it.

2.2 A Better Algorithm. The new algorithm, algorithm B below, finds a *maximum* triangular structure (one with the maximum number of edges) in a given graph G. Algorithm B has performance ratio at least 0.4, and can be implemented in time $O(m^{3/2} n \log^6 n)$. Now, let us present the algorithm and the lower bound of 0.4 on its performance ratio.

Algorithm B also has two phases. In the first one, B constructs a maximum triangular cactus S_1 in G. We will show later how to use a matroid parity algorithm to construct S_1. In the second phase, B extends S_1 to a triangular structure S_2 in G, as before, by adding to S_1 as many edges as possible which do not form new cycles.

Algorithm B

```
Let $S_1$ be a *maximum* triangular cactus in $G$.
Starting with $E_2 = E(S_1)$, repeatedly (as long as possible) find an edge $e$ in $G$ whose endpoints are in different components of $G[E_2]$, and add $e$ to $E_2$.
Let $S_2 := G[E_2]$.
Output $S_2$.
```

Observe that S_2 is a triangular structure in G, and therefore is planar. To analyze the algorithm, we need a definition. In any graph H, let $mts(H)$ denote the number of edges in a maximum triangular structure in H. Define $\rho(H) = mts(H)/|E(H)|$ if $E(H) \neq \emptyset$, and $\rho(H) = 1$ if $E(H) = \emptyset$.

We will prove that $\rho(H) \geq 0.4$ provided that H is planar. (And later we will prove that $\rho(H) \geq 2/3$ if H is outerplanar.) The key to understanding the analysis of algorithm B is the following. If G is any graph, let H be a maximum planar subgraph of G. Clearly $mts(G) \geq mts(H)$. Now $Opt(G) = |E(H)|$ implies that $\mathrm{B}(G)/Opt(G) = mts(G)/|E(H)| \geq mts(H)/|E(H)| = \rho(H)$. If we prove that $\rho(H) \geq 0.4$ for any planar H, we can infer that the performance ratio of B is at least 0.4.

THEOREM 2.2. *If H is a planar graph, then $\rho(H) \geq 0.4$.*

Proof. The theorem is easily verified if H has fewer than three vertices, so let us assume that H has $n \geq 3$ vertices. We may furthermore assume that H is connected. Embed H in the plane. Choose $t \geq 0$ so that $|E(H)| = 3n - 6 - t$.

Now let J be any triangular cactus obtained by choosing triangular faces of H until no more can be added; say the final J has k components. Let p be the number of triangles in J. As in the proof of Theorem 2.1, if we count twice every edge in H whose endpoints are in the same component of J, we will "cover" every triangular face of H; and, in fact, each triangular face of J will be covered three times, by the three edges bounding the face. Let s be the number of edges in H whose endpoints are in the same component of J. Let l be the number of triangular faces in H. Since the p triangles in J are covered three times, we have $(l - p) + 3p = l + 2p \leq 2s$. As in Theorem 2.1, we have $s \leq 6p - 3k$ and $l \geq 2n - 4 - 2t$.

It follows that $2n - 4 - 2t + 2p \leq l + 2p \leq 2s \leq 2(6p - 3k)$, so that

$$p \geq \frac{2n - 4 - 2t + 6k}{10} = \frac{n - 2 - t + 3k}{5} \geq \frac{n - 2 - t}{5}.$$

Since $\rho(H) = \frac{mts(H)}{|E(H)|}$, using $mts(H) \geq (n - 1) + p$, we have

$$\rho(H) \geq \frac{n - 1 + \frac{n-2-t}{5}}{3n - 6 - t} = \frac{6n - 7 - t}{15n - 30 - 5t} \geq \frac{2}{5},$$

for any $t \geq 0$. ∎

COROLLARY 2.1. *The performance ratio of algorithm B is at least* 0.4.

The next theorem gives an upper bound on the performance ratio of algorithm B.

THEOREM 2.3. *The performance ratio of algorithm B is at most $\frac{4}{9}$.*

Proof. We will prove this by presenting, for any $\epsilon > 0$, a *planar* graph G_ϵ such that algorithm B, with G_ϵ as input, can produce a subgraph S_2 of G_ϵ whose number of edges is at most $\frac{4}{9} + \epsilon$ times the number of edges of G_ϵ.

Given $\epsilon > 0$, let n' be an integer such that $n' > \frac{6\epsilon+1}{3\epsilon}$ and $n' \geq 3$. Let G'_ϵ be any triangulated plane graph on n' vertices. Call V' the vertex set of G'_ϵ. Since G'_ϵ is triangulated, G'_ϵ has $2n' - 4$ (triangular) faces. For each face of G'_ϵ, add a new vertex in the face and adjacent to all three vertices on the boundary of that face. Let G_ϵ be the new graph, and let V be the vertex set of G_ϵ. Observe that G_ϵ is a triangulated plane graph, and has $n' + (2n' - 4) = 3n' - 4$ vertices. Therefore, G_ϵ has $3(3n' - 4) - 6 = 9n' - 18$ edges. Let S be a maximum triangular structure in G_ϵ.

Any edge in G_ϵ has at least one endpoint in V'. Moreover, $|V'| = n'$. Therefore, a maximum matching in G_ϵ has at most n' edges (each with at least one distinct endpoint in V'). The following lemma is observed in [LP86, p. 440].

LEMMA 2.1. *If S is a triangular structure with t triangles in a given graph G, then there is a matching in G of size t.*

Using the lemma above, we conclude that S has at most n' triangles. Recall that S, being a triangular structure, is a spanning tree of G_ϵ plus one edge per triangle in S, which implies that S has at most $(3n' - 5) + n' = 4n' - 5$ edges. Furthermore, G_ϵ has $9n' - 18$ edges. Therefore the ratio between the number of edges in S and the number of edges in G_ϵ is

$$\frac{4n' - 5}{9n' - 18} < \frac{4}{9} + \epsilon,$$

because $n' > \frac{6\epsilon+1}{3\epsilon}$. ∎

How can one find a maximum triangular cactus quickly? A graphic matroid parity algorithm can be used to construct a maximum triangular cactus in a given graph [LP86]. The problem solved by a graphic matroid parity algorithm is GRAPHIC MATROID PARITY (GMP): given a multigraph $H = (V_H, E_H)$ and a partition of the edge set E_H into pairs of distinct edges $\{f, f'\}$, find a (simple) forest F with the maximum number of edges, such that $f \in F$ if and only if $f' \in F$, for all $f \in E_H$.

Let us show how to reduce the problem of finding a maximum triangular cactus in a given graph $G = (V, E)$ to GMP. This is done by describing a multigraph $G' = (V', E')$ and a partition $\mathcal{P}$ of E' into pairs of distinct edges of E', such that, from a solution to GMP for G'

and $\mathcal{P}$, we can construct a maximum triangular cactus in G.

First let $V' = V$. Now, let us describe E' and the partition $\mathcal{P}$. Initially, $E' = \emptyset$ and $\mathcal{P} = \emptyset$. For each triangle in G with edge set T, let $\{e, e'\}$ be any pair of distinct edges in T. Add two new edges f and f' to E', f with the same endpoints as e, and f' with the same endpoints as e'. We say that T *corresponds to* $\{f, f'\}$. Insert f and f' into $\mathcal{P}$.

We say a forest F in G' is *valid* if $f \in F$ if and only if $f' \in F$, for all f in E'. Observe that any valid forest has an even number of edges. The following lemma states a relation between valid forests in G' and triangular cacti in G. Let m and n be the number of edges and vertices, respectively, in G.

LEMMA 2.2. *There is a valid forest F in G' with $2p$ edges if and only if there is a triangular cactus S in G with p triangles. Moreover, S can be obtained from F (and* vice versa*) in time $O(n)$.*

For lack of space, we omit the proof of this lemma, which is used implicitly in [LP86].

As described by Chiba and Nishizeki [CN85], we can explicitly list all the triangles in a graph G with m edges in time $O(m^{3/2})$. So $|E'|$ is $O(m^{3/2})$.

Gabow and Stallmann [GS85] describe an algorithm for GMP, which runs in time $O(m'n' \log^6 n')$, where m' and n' are the number of edges and vertices, respectively, in the input graph. In our case, $n' = n$ and $m' = |E'|$, which is $O(m^{3/2})$. This gives a time bound of $O(m^{3/2} n \log^6 n)$ for this phase.

From the output of the Gabow-Stallmann algorithm, it is easy to find a maximum triangular cactus in time $O(n)$ (Lemma 2.2). Therefore the total time is $O(m^{3/2} n \log^6 n)$.

3 Outerplanar Subgraphs

Serendipitously, Algorithm B produces outerplanar graphs, so it is an approximation algorithm for MAXIMUM OUTERPLANAR SUBGRAPH, which is NP-Complete [GJ79, p. 197]. In fact, any algorithm which produces a spanning tree has performance ratio at least 1/2, because any outerplanar graph on $n \geq 2$ vertices has at most $2n-3$ edges (see below). A careful analysis shows that the performance ratio of B when used for MAXIMUM OUTERPLANAR SUBGRAPH is at least 2/3. This is an easy consequence of Theorem 3.1, in order to prove which we need some preliminaries.

An outerplanar graph G is a *maximal outerplanar graph* if no edge can be added without losing outerplanarity. As mentioned in [H72, p. 106], every maximal outerplanar graph G with at least three vertices is a triangulation of a polygon (i.e., the boundary of the exterior face is a Hamiltonian cycle and each interior face is triangular). By [H72, Cor. 11.9], G must have a vertex of degree two and $2|V(G)| - 3$ edges (this last statement is also true for $|V(G)| = 2$).

LEMMA 3.1. *Let H be a maximal outerplanar graph. If H has an odd number $n = 2p + 1$ of vertices, then there is a triangular cactus in H with p triangles. If H has an even number $n = 2p$ of vertices and xy is an edge on the boundary of the exterior face, then there is a triangular cactus S in H with $p - 1$ triangles such that x and y are not connected in S.*

Notice that we obtain the maximum number of triangles possible. In the former case all vertices are in the same component of the cactus, while in the latter, the cactus has two components.

Proof. We use a plane embedding of H.

The proof is by induction on n, the number of vertices of H. The case $n = 1$ is trivial. If $n = 2$ (in this case there is only one edge and $p = 1$), the theorem is true.

We inductively construct a triangular cactus of the given size.

Let $n = 2p + 1$. Let v be a vertex of degree two. Let x and y be its neighbors. They are adjacent, since interior faces are triangles. The graph $H - \{v\}$ is maximal outerplanar (since it has $(2n - 3) - 2 = 2(n - 1) - 3$ edges) and has an even number of vertices. It is easy to check that if a triangular cactus S' in this smaller graph has the property that x and y are not connected in S', we can add the triangle xyv to get a triangular cactus in H. The size of this cactus is $p - 1$, by induction, plus one, for a total of p.

Let $n = 2p$ and let the edge xy be on the boundary of the exterior face. This edge is on the boundary of a triangular face xyv on the inside. Walking along the Hamiltonian cycle which is the boundary of the exterior face, starting at v and in the direction that visits x just before y, let D_1 be the set of vertices visited between v and x, and let $n_1 = |D_1|$. Walking along the Hamiltonian cycle in the opposite direction again starting at v, let D_2 be the set of vertices visited between v and y, and let $n_2 = |D_2|$; $D_1 \cap D_2 = \{v\}$ and $D_1 \cup D_2 = V(H)$. The only edge in H between $D_1 - \{v\}$ and $D_2 - \{v\}$ is the edge xy.

Let H_1 be the subgraph of H induced by vertex set D_1, with, say, e_1 edges, and let H_2 be the subgraph of H induced by vertex set D_2, with, say, e_2 edges.

We have $n_1 + n_2$ is odd, since v is counted twice. Let us say without loss of generality that $n_1 = 2p_1 + 1$ is odd and $n_2 = 2p_2$ is even. Then $n = 2(p_1 + p_2)$. We have $e_1 + e_2 = (2n - 3) - 1$, as from H only the edge xy is not an edge of either H_1 or H_2. Since $e_1 \leq 2n_1 - 3$ and $e_2 \leq 2n_2 - 3$, we infer that $e_1 + e_2 \leq 2(n_1 + n_2) - 6 = 2(n + 1) - 6 = 2n - 4$. Since, in fact, $e_1 + e_2 = 2n - 4$, we infer that $e_1 = 2n_1 - 3$ and $e_2 = 2n_2 - 3$. Thus both H_1 and H_2 have to be maximal outerplanar, as they have the maximum number of edges.

Then by the inductive hypothesis we can construct in H_1 a cactus S_1 with p_1 triangles. If we apply the inductive hypothesis to H_2 with vy being the edge on the exterior face, we obtain a triangular cactus S_2 with $p_2 - 1$ triangles in which y and v are not connected. Then putting together the edges of S_1 and S_2 we get S, a cactus in H. In the new cactus S, any possible $x - y$-path must visit v, since neither S_1 nor S_2 has edge xy. But in S_2, y and v are not connected. It follows that x and y are not connected in S, so S is the desired cactus. S has $p_1 + (p_2 - 1)$ triangles, which is exactly the number we wanted. ■

In conclusion, for a maximal outerplanar graph with n vertices, we can find a triangular structure with $\lfloor \frac{n-1}{2} \rfloor$ triangles.

Now we prove a lower bound on $\rho(H)$.

THEOREM 3.1. *If H is outerplanar, then $\rho(H) \geq 2/3$.*

Proof. Let H be any 2-connected outerplanar graph. We add t edges to obtain a maximal outerplanar plane graph H'. Note that H' has $2n - 3$ edges and a triangular structure S with at least $\lfloor \frac{n-1}{2} \rfloor$ triangles.

However, the t missing edges can destroy at most t of these triangles in S, because S is a cactus. If $t \geq \frac{n}{2}$, we infer that

$$\rho(H) \geq \frac{n-1}{2n-3-n/2} \geq \frac{2}{3}.$$

Assume to the contrary that $t \leq \lfloor \frac{n-1}{2} \rfloor$. Then the number of edges in the triangular structure is at least $n - 1 + (\lfloor \frac{n-1}{2} \rfloor - t)$. Then

$$\rho(H) \geq \frac{n-1+\lfloor \frac{n-1}{2} \rfloor - t}{2n-3-t}.$$

The worst case is achieved when $t = \lfloor \frac{n-1}{2} \rfloor$ and is $\frac{2}{3}$.

If H is not 2-connected, we can do the above analysis for each of the 2-connected components (an edge appears in exactly one 2-connected component) and infer that a maximum triangular structure has $\frac{2}{3}$ of the edges in H. ■

The theorem above is tight, in the sense that there are outerplanar graphs H for which $\rho(H)$ is arbitrarily close to 2/3. In fact, there are outerplanar graphs H_i with $2i$ vertices and $3i - 2$ edges which do not have any triangle.

COROLLARY 3.1. *Algorithm B has performance ratio 2/3 for MAXIMUM OUTERPLANAR SUBGRAPH.*

4 The Complexity of the Problems

Papadimitriou and Yannakakis [PY91] defined a natural variant of NP for optimization problems: the complexity class Max SNP. This class, as they have shown, contains several well-known optimization problems, such as MAX 3-SAT and MAXIMUM CUT. In this section, we prove that MAXIMUM PLANAR SUBGRAPH (MPS) is Max SNP-hard, as is its complementary version: given a graph, find a smallest subset of its edges whose removal results in a planar graph. This means, by results of Arora et al. [ALMSS92], that there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm for MPS with performance ratio at least $1 - \epsilon$ implies that $P = NP$, and that an analogous statement can be made about the complementary problem.

As in [PY91], we use the concept of *L-reduction*, which is a special kind of reduction that preserves approximability. Let A and B be two optimization problems. We say *A L-reduces to B* if there are two polynomial-time algorithms f and g, and positive constants α and β, such that for each instance I of A,

1. Algorithm f produces an instance $I' = f(I)$ of B, such that the optima of I and I', of costs denoted $Opt_A(I)$ and $Opt_B(I')$ respectively, satisfy $Opt_B(I') \leq \alpha \cdot Opt_A(I)$, and

2. Given any feasible solution of I' with cost c', algorithm g produces a solution of I with cost c such that $|c - Opt_A(I)| \leq \beta \cdot |c' - Opt_B(I')|$.

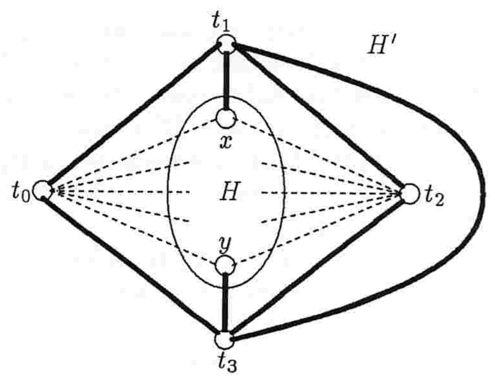
The main result of this section is

THEOREM 4.1. *MAXIMUM PLANAR SUBGRAPH is Max SNP-hard.*

Proof. Denote by $TSP_4(1,2)$ the following variant of the traveling salesman problem: given a complete graph, a pair of distinct vertices x, y, and costs one or two for each edge, such that the graph induced by the edges of cost one has maximum degree at most four, find a Hamiltonian path from x to y of minimum cost. Papadimitriou and Yannakakis [PY93] showed that $TSP_4(1,2)$ is Max SNP-hard.

We shall prove $TSP_4(1,2)$ L-reduces to MPS. The basic idea of the reduction comes from Liu and Geldmacher [LG77], where the decision version of MPS is proved to be NP-complete.

The first part of the L-reduction is the polynomial-time algorithm f and the constant α. Given any instance I of $TSP_4(1,2)$, f produces an instance G of MPS such that the cost of the optimum of G in MPS, denoted $Opt_{MPS}(G)$, is at most α times the cost of the optimum of I in $TSP_4(1,2)$, denoted by $Opt_{TSP_4(1,2)}(I)$, i.e., $Opt_{MPS}(G) \leq \alpha \cdot Opt_{TSP_4(1,2)}(I)$.



edges in F_1 ($2n$-edges)

---- edges in F_2 (8-edges)

Figure 1: Graph H' constructed from H.

Consider an instance I of $TSP_4(1,2)$. I is a complete graph $K = (V,E)$, a pair of distinct vertices x, y of V and a subset E_1 of E consisting of the edges of cost one. Let $H = (V, E_1)$ and $H' = (V \cup T, E_1 \cup F_1 \cup F_2)$, where $T = \{t_0, t_1, t_2, t_3\}$, $T \cap V = \emptyset$, $F_1 = \{t_0t_1, t_0t_3, t_1t_2, t_1t_3, t_1x, t_2t_3, t_3y\}$ and $F_2 = \cup_{z \in V}\{t_0z, t_2z\}$ (see figure 1).

Denoting by n the number of vertices of H, let G be the graph obtained from H' by (1) replacing each edge e in F_1 by $2n$ parallel internally-disjoint paths of length two (having new internal vertices) between the endpoints of e and (2) replacing each edge e in F_2 by eight parallel internally-disjoint paths of length two (having new internal vertices) between the endpoints of e.

Clearly G can be obtained from I in time polynomial in the size of I.

LEMMA 4.1. $Opt_{MPS}(G) \leq 124 \cdot Opt_{TSP_4(1,2)}(I)$.

Proof. Observe that $Opt_{TSP_4(1,2)}(I) \geq n-1$. A clear upper bound for $Opt_{MPS}(G)$ is the number of edges of G. To compute this, note first that H has maximum degree at most four by the definition of $TSP_4(1,2)$, and so H has at most $2n$ edges. Let us call the edges in F_1 *2n-edges* and the edges in F_2 *8-edges*. There are seven $2n$-edges: F_1 contains seven edges, each of them corresponding to $4n$ edges in G. There are $2n$ 8-edges: F_2 contains $2n$ edges, each of them corresponding to 16 edges in G. Hence the number of edges in G outside of H is $7 \cdot 4n + 16 \cdot 2n = 60n$. The total number of edges in G is therefore at most $2n + 60n = 62n \leq 124(n-1)$. Therefore, $Opt_{MPS}(G) \leq 124(n-1) \leq 124 \cdot Opt_{TSP_4(1,2)}(I)$. ■

This finishes the first part of the L-reduction, since we can take $\alpha = 124$.

The second and hard part of the L-reduction is the constant β and the algorithm g. Given a planar subgraph of G with m edges, g produces in polynomial time a Hamiltonian path from x to y of cost t in K such that $|t - Opt_{TSP_4(1,2)}(I)| \leq \beta|m - Opt_{MPS}(G)|$. We shall see that $\beta = 1$ suffices.

First, given a planar subgraph P of G, let us describe another planar subgraph P' of G with at least as many edges as P. Moreover, P' shall contain all edges of G not in H.

Let e be a $2n$-edge or an 8-edge of H'. We say e *appears in* P if P contains both edges of *all* the paths of length two corresponding to e. In this case, we also say that the endpoints of e *are adjacent in* P *by the* $2n$*-edge* or *8-edge* e. We say e *is missing in* P if P contains both edges of *none* of the paths (of length two) corresponding to e. (In this case, if e is an 8-edge, then P is missing at least eight of its 16 edges in G corresponding to e.) It is possible that a $2n$-edge or an 8-edge of H' neither appears in P nor is missing in P.

Let us modify P so that any $2n$-edge or 8-edge of H' either appears in P or is missing in P. This is done as follows. If a $2n$-edge or an 8-edge e of H' neither appears nor is missing in P, then we insert in P all edges of G in the paths (of length two) corresponding to e. Note that P remains planar. Clearly, the number of edges in P cannot decrease by this operation. The new graph is also called P.

Now we can describe P'. We have three cases: (1) if some $2n$-edge e does not appear in P, then define P' to be the graph induced by all edges of G not in H; (2) if all the $2n$-edges and the 8-edges of H' appear in P, then let P' be the same as P; and (3) if all the $2n$-edges of H' appear in P, but not all the 8-edges, then we modify P to obtain P', as described in the next two paragraphs.

The idea is to remove from P some edges of H and add to P edges of H' not in H so that all the 8-edges of H' appear in the modified graph, and it remains planar and has at least as many edges as the original P.

Let U be the set of vertices v of H such that at least one of the two 8-edges incident to v in H' is missing in P. Observe that $|U| \geq 1$, as case (2) considered $|U| = 0$. For each vertex v in U, remove from P all edges of H incident to v in P (at most four edges are removed per vertex) and add to P all the edges outside of H so that the two 8-edges incident to v appear in P (at least eight edges are added, corresponding to the 8-edges incident to v missing in P). To guarantee that the graph obtained this way is planar, we must make room to embed the modified 8-edges. This is done by also removing from P all edges of H incident to y (if they were not already removed). Let P' be the graph

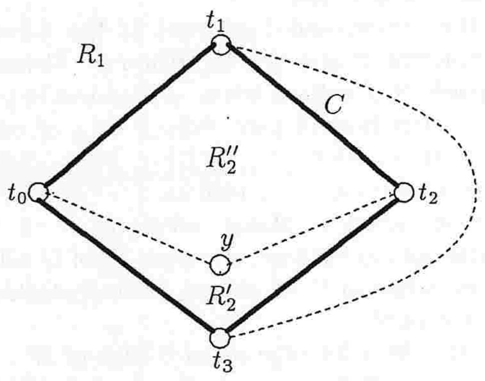


Figure 2: Cycle C, regions R_1 and $R_2 = R_2' \cup R_2''$.

obtained after all these modifications.

LEMMA 4.2. *P' is planar and has at least as many edges as P.*

Proof. In case (1), we include in P' at least $2n$ edges that do not appear in P (at least one in each of the $2n$ paths corresponding to e), and we remove at most $2n$ edges, the maximum number of edges in H. So P' has at least as many edges as P. Moreover P' is planar.

There is nothing to be proved in case (2).

Case (3) is the complicated one. First note that P' has at least as many edges as G, since, for each vertex in U, we remove at most four edges and add at least eight. Furthermore, we remove at most four edges incident to y. Hence, we gain at least $(8-4)|U|-4 = 4|U|-4 \geq 0$ edges, since $|U| \geq 1$.

Now, let us show that P' is planar. We can think of the $2n$-edges and 8-edges as single edges, as they are in H' (since if we can embed a single edge, we can embed a $2n$-edge or an 8-edge as well). We will modify a given embedding of P into an embedding for P'.

Let C be the cycle (using four $2n$-edges) t_0, t_1, t_2, t_3, t_0. Observe that the $2n$-edges in C appear in P, since we are in case (3). Given an embedding for P, cycle C divides the plane into two regions, R_1, containing the $2n$-edge t_1t_3, and R_2 (see figure 2). The $2n$-edge t_1t_3 separates t_0 from t_2 in R_1. Moreover, each vertex in $V-U$ (the vertices of H not in U) is adjacent in P by 8-edges to t_0 and t_2. Because t_0 and t_2 are separated in R_1, none of these vertices can be embedded in R_1, which implies they must be embedded in R_2. Keep these vertices (t_0, t_1, t_2, t_3 and the vertices in $V-U$) embedded as they are.

Now, observe that y is adjacent in P' only to t_0, t_2 (by 8-edges) and t_3 (by a $2n$-edge). Furthermore, y is the only vertex in H which is adjacent to t_3. This means, before we embed y, the vertices t_0, t_2 and t_3 are not separated in R_2. Therefore, y can be embedded in R_2 with the $2n$-edge t_3y and the 8-edges t_0y and yt_2 "next to" the $2n$-edges t_0t_3 and t_3t_2. The edges t_0y and yt_2 together split region R_2 into two regions R_2' containing edge t_3y, and R_2'' containing vertex t_1. Observe that t_0 and t_2 are not separated in R_2'', since t_3 is the only vertex, besides t_0 and t_2, which is adjacent to y (by a $2n$-edge).

All the vertices in $U - \{x\}$ are adjacent in P' only to t_0 and t_2 (by 8-edges), and therefore they all can be embedded in R_2'' with their two 8-edges "next to" the 8-edges t_0y and yt_2. If $x \in U$ then observe that x is the only vertex in H which is adjacent in P' to t_1 (by a $2n$-edge). This means, before we embed x, the vertices t_0, t_1 and t_2 are not separated in R_2''. Therefore, x can be embedded in R_2'' with the $2n$-edge xt_1, and the 8-edges t_0x and xt_2 "next to" the $2n$-edges t_0t_1 and t_1t_2. If $x \notin U$ then it does not need to be moved in the embedding. The embedding obtained this way is a plane embedding of P', completing the proof that P' is planar. ∎

Observe that P' contains all the edges of G not in H. Let F be the set of edges of H appearing in P'.

LEMMA 4.3. *The graph $G_F = (V, F)$ is a collection of vertex-disjoint paths which can be extended in K (the complete graph on V) to a Hamiltonian path from x to y, in polynomial time.*

Proof. Let us prove that G_F satisfies the following four conditions: (1) There is no vertex of degree greater than two in G_F. (2) Vertices x and y have degree at most one in G_F. (3) There is no cycle in G_F. (4) If x and y are in the same component of G_F, then this component spans all vertices in V. We will prove each of these conditions holds by contradiction.

Suppose (1) does not hold. Let z_0 be a vertex in V of degree at least 3 in G_F. Let z_1, z_2, z_3 be three of its neighbors in G_F. (Notice that z_0, z_1, z_2, z_3 are distinct vertices of H, so they are distinct of t_0, t_2.) Then each one of t_0, t_2, z_0 is adjacent in P' to each one of z_1, z_2, z_3 (some of them are adjacent in P' by 8-edges). Therefore, $t_0, t_2, z_0; z_1, z_2, z_3$ define a subdivision of $K_{3,3}$ in P', a contradiction, because P' is planar. Thus, (1) holds.

Suppose (2) does not hold. If x has degree more than one in G_F, let z_1 and z_2 be two of its neighbors in G_F. (Notice that z_1 and z_2 are distinct vertices of H, distinct of x, so they are distinct of t_0, t_2.) Then each one of t_0, t_2, x is adjacent in P' to each one of t_1, z_1, z_2 (some of them are adjacent in P' by $2n$-edges or 8-edges). Therefore, $t_0, t_2, x; t_1, z_1, z_2$ define a subdivision of $K_{3,3}$ in P', a contradiction, because P' is planar. Analogously, we have a contradiction if y has degree more than one in G_F. Thus, (2) holds.

Suppose (3) does not hold. Let z_1, z_2, z_3 be three vertices in a cycle of G_F. (Since z_1, z_2, z_3 must have degree at least two, they are not x or y by condition (2), and they are not t_0 or t_2 since they are vertices of H.) Then z_1, z_2, z_3, t_0, t_2 are pairwise linked by internally vertex-disjoint paths (the path between t_0 and t_2 uses the two 8-edges incident to x, while the others use one $2n$-edge or 8-edge). Therefore, t_0, t_2, z_1, z_2, z_3 define a subdivision of K_5 in P', a contradiction, because P' is planar. Hence, (3) holds.

Suppose (4) does not hold. Let z_0 be a vertex in V which is not in the component having x and y in G_F. In this case, t_0, t_1, t_2, x, y are pairwise linked by internally vertex-disjoint paths (the path between t_0 and t_2 uses z_0, the path between t_1 and y uses t_3, the path between x and y is in G_F, and the others use one $2n$-edge or 8-edge). Therefore, t_0, t_1, t_2, x, y define a subdivision of K_5 in P', a contradiction, because P' is planar. Hence, (4) holds.

Therefore, the conditions hold. From (1) and (2), we conclude that G_F is a collection of paths. From (2) and (3), these paths can be extended in K (the complete graph on V) to a Hamiltonian path from x to y. Furthermore, note that this can be done in polynomial time. ■

Let HP be a Hamiltonian path from x to y containing all edges in F and some edges (in K) of cost two. HP exists by Lemma 4.3. Denote by m' the number of edges of P' and by t the cost of HP.

Now the following Lemma states that β exists, and specifically, $\beta = 1$. This will complete the proof of the theorem.

LEMMA 4.4.

$$t - Opt_{TSP_4(1,2)}(I) = Opt_{MPS}(G) - m',$$

and hence

$$|t - Opt_{TSP_4(1,2)}(I)| = |m' - Opt_{MPS}(G)|.$$

Proof. As in the proof of Lemma 4.1, the number of edges in G outside of H is $60n$. All these edges are in P'. Therefore, the number of edges in F is $m' - 60n$. And the cost of HP is

$$t = 2(n-1) - (m' - 60n). \tag{4.1}$$

Let Q be an optimal solution of MPS for G. Using the same argument for Q that we used for P, Lemma 4.3 and the argument above imply the existence of a Hamiltonian path of cost $2(n-1) - (Opt_{MPS}(G) - 60n)$. Therefore

$$Opt_{TSP_4(1,2)}(I) \le 2(n-1) - (Opt_{MPS}(G) - 60n). \tag{4.2}$$

Given an optimum solution HP^* of $TSP_4(1,2)$ for I, we can construct a solution of MPS for G by selecting all the edges outside of H plus the edges of cost one in HP^*. Observe that these edges really determine a planar subgraph of G. Let z be the number of these edges. Since we have $60n$ edges in G outside of H, HP^* has $z - 60n$ edges of cost one and the remaining ones (of its $n-1$ edges) have cost two. This means that

$$\begin{aligned} Opt_{TSP_4(1,2)}(I) &= 2(n-1) - (z - 60n) \\ &\ge 2(n-1) - (Opt_{MPS}(G) - 60n), \end{aligned}$$

since $Opt_{MPS}(G) \ge z$.

Therefore, from (4.2) and (4.3), we have

$$Opt_{TSP_4(1,2)}(I) = 2(n-1) - (Opt_{MPS}(G) - 60n).$$

And this together with (4.1) means

$$t - Opt_{TSP_4(1,2)}(I) = Opt_{MPS}(G) - m'.$$

Hence $|t - Opt_{TSP_4(1,2)}(I)| = |m' - Opt_{MPS}(G)|$. ■

From $m \le m'$, it follows that

$$t - Opt_{TSP_4(1,2)}(I) \le Opt_{MPS}(G) - m$$

and

$$|t - Opt_{TSP_4(1,2)}(I)| \le 1 \cdot |m - Opt_{MPS}(G)|.$$

This completes the proof of Theorem 4.1. ■

Let us denote the complementary version of MPS by NPD: given a graph G, find a smallest set of edges of G whose removal results in a planar graph.

A slight modification of the L-reduction presented above proves the following.

THEOREM 4.2. *NPD is Max SNP-hard.*

Proof. The first part of the L-reduction is almost the same. From an instance I of $TSP_4(1,2)$, we construct G in exactly the same way. As before, $Opt_{TSP_4(1,2)}(I) \ge n-1$. As in the proof of Lemma 4.1, the maximum number of edges of G is $62n$. Thus, the optimum of G in NPD, denoted as $Opt_{NPD}(G)$, is at most $62n$. And then $Opt_{NPD}(G) \le 62n \le 124(n-1) \le 124 \cdot Opt_{TSP_4(1,2)}(I)$. We can take $\alpha = 124$, as before.

In the second part, given an instance I of $TSP_4(1,2)$, let G be constructed from I as in the previous reduction. Let D be a subset of the edges of G whose removal results in a planar subgraph of G. We shall find in polynomial time a Hamiltonian path HP such that $t - Opt_{TSP_4(1,2)}(I) \le d - Opt_{NPD}(G)$, where t is the number of edges in HP and $d = |D|$. Just as we took a planar subgraph P of G and found a planar subgraph P' which contains all edges of G not in H, and is at least as large as P, from D we can find a set D' of edges of G containing *none* of the edges of G not in H, which is at least as *small* as D. Applying Lemma 4.3, we can obtain a Hamiltonian path HP, as before, in polynomial time.

Now, let us prove that β exists. Let m' be the number of edges in P'. Note that $d' + m' = |E(G)|$. Moreover, $Opt_{MPS}(G) + Opt_{NPD}(G) = |E(G)|$. Therefore, $d' - Opt_{NPD}(G) = Opt_{MPS}(G) - m'$. Applying Lemma 4.4, we conclude that $t - Opt_{TSP_4(1,2)}(I) = d' - Opt_{NPD}(G)$, which, together with $d' \leq d$, implies that we can take $\beta = 1$. ■

5 Open Problems

Many open problems are suggested by this research. How large a performance ratio can one achieve is an obvious one. Is there a linear-time approximation algorithm for MAXIMUM PLANAR SUBGRAPH with performance ratio $1/3 + \epsilon$? (A *maximal* planar subgraph can be found in linear time [H95, D95].) Is there any approximation algorithm with a constant performance ratio for NPD? Can one achieve a performance ratio of $1/3 + \epsilon$ for MAXIMUM WEIGHT PLANAR SUBGRAPH, which is this problem: given a weighted graph, find a planar subgraph of maximum weight. For this problem, any maximum weight spanning tree can be shown to have weight at least one third of the optimum. What performance ratios are achievable for finding heavy outerplanar subgraphs? What performance ratio can be achieved for THICKNESS (given G, partition the edges of G into as few planar subgraphs as possible)? A factor of 3 here is trivial, via arboricity.

6 Acknowledgments

The authors are very grateful to David Williamson for his help. We thank Laci Lovász for informing us that a maximum triangular structure can be found in polynomial time. We thank Robert Cimikowski for kindly sending us drafts of his papers. The third author would like to thank Kurt Mehlhorn, for suggesting the problem and supporting his work. The first author would like to thank Robin Thomas for helpful discussions. We also would like to thank the program committee for informing us about a faster way to find all triangles in a graph.

References

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof Verification and Hardness of Approximation Problems," *Proc. 33rd IEEE Symposium on Foundations of Computer Science*, 14–23, 1992.

[CHT93] J. Cai, X. Han and R. E. Tarjan, "An $O(m \log n)$-time Algorithm for the Maximal Planar Subgraph Problem," *SIAM Journal on Computing*, 22:1142–1162, 1993.

[Cim95] R. Cimikowski. "An Analysis of Some Heuristics for the Maximum Planar Subgraph Problem," *Proc. 6th Annual ACM-SIAM Symp. on Discrete Algorithms*, 322–331, 1995.

[CNS79] T. Chiba, I. Nishioka and I. Shirakawa. "An Algorithm of Maximal Planarization of Graphs," *Proc. IEEE Symp. on Circuits and Systems*, 649–652, 1979.

[CN85] N. Chiba and T. Nishizeki, "Arboricity and Subgraph Listing Algorithms," *SIAM Journal of Computing*, 14:210-223, 1985.

[D95] H. N. Djidjev, "A Linear Algorithm for the Maximal Planar Subgraph Problem," *Proc. 4th International Workshop on Algorithms and Data Structures* (WADS '95), 369-380, 1995.

[DFF85] M. E. Dyer, L. R. Foulds and A. M. Frieze, "Analysis of Heuristics for Finding a Maximum Weight Planar Subgraph," *European Journal of Operational Research*, 20:102–114, 1985.

[F92] L. R. Foulds, *Graph Theory Applications*, Springer-Verlag, New York, 1992.

[GS85] H. N. Gabow and M. Stallmann, "Efficient Algorithms for Graphic Matroid Intersection and Parity," *Automata, Language and Programming: 12th Colloq., Lecture Notes in Computer Science*, vol. 194, 210–220, 1985.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Co., 1979.

[H72] F. Harary, *Graph Theory*, Addison-Wesley Publishing Company, 1972.

[H95] W. Hsu, "A Linear Time Algorithm for Finding Maximal Planar Subgraphs," *Proc. 6th Annual International Symposium on Algorithms and Computation* (ISAAC95), 1995.

[JM93] M. Jünger and P. Mutzel, "Maximum Planar Subgraph and Nice Embeddings: Practical Layout Tools," *Universität zu Köln, Report No. 93.145*, 1993, to appear in *Special Issue of Algorithmica on Graph Drawing.*

[LG77] P. C. Liu and R. C. Geldmacher, "On the Deletion of Nonplanar Edges of a Graph," *Proc. 10th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, 727–738, 1977.

[LP86] L. Lovász and M. D. Plummer, *Matching Theory*, Elsevier Science, Amsterdam, 1986.

[PY91] C. H. Papadimitriou and M. Yannakakis. "Optimization, Approximation, and Complexity Classes," *Journal of Computer and System Sciences*, 43:425–440, 1991.

[PY93] C. H. Papadimitriou and M. Yannakakis, "The Traveling Salesman Problem with Distances One and Two," *Mathematics of Operations Research*, 18(1):1–11, 1993.

[TDB88] R. Tamassia, G. Di Battista and C. Batini, "Automatic Graph Drawing and Readability of Diagrams," *IEEE Transactions on Systems, Man and Cybernetics*, 18:61-79, 1988.

Algorithmica (2003) 36: 179–205
DOI: 10.1007/s00453-002-1020-3

Algorithmica
© 2003 Springer-Verlag New York Inc.

A New Approximation Algorithm for Finding Heavy Planar Subgraphs[1]

Gruia Călinescu,[2] Cristina G. Fernandes,[3] Howard Karloff,[4] and Alexander Zelikovsky[5]

Abstract. We provide the first nontrivial approximation algorithm for MAXIMUM WEIGHT PLANAR SUBGRAPH, the NP-hard problem of finding a heaviest planar subgraph in an edge-weighted graph G. This problem has applications in circuit layout, facility layout, and graph drawing. No previous algorithm for MAXIMUM WEIGHT PLANAR SUBGRAPH had a performance ratio exceeding $1/3$, which is obtained by any algorithm that produces a maximum weight spanning tree in G. Based on the Berman–Ramaiyer Steiner tree algorithm, the new algorithm has performance ratio at least $1/3 + 1/72$ and at most $5/12$. We also show that if G is complete and its edge weights satisfy the triangle inequality, then the performance ratio is at least $3/8$. Furthermore, we derive the first nontrivial performance ratio ($7/12$ instead of $1/2$) for the NP-hard MAXIMUM WEIGHT OUTERPLANAR SUBGRAPH problem.

Key Words. Weighted planar graph, Approximation algorithm, Performance ratio.

1. Introduction.

MAXIMUM WEIGHT PLANAR SUBGRAPH is this problem: given a graph G with a nonnegative weight defined for each edge, find a planar subgraph of G of maximum weight. This problem—especially this weighted version—has applications in graph drawing, circuit layout, and facility layout.

Graph drawing is the process of drawing a graph, usually on a two-dimensional medium, as "nicely" as possible, where "nicely" is defined by the application. A long annotated bibliography on the large class of graph-drawing algorithms is found in [BETT]. A plane embedding of a planar graph G could naturally be viewed as "nice." Graph-drawing experts have studied many different ways of drawing planar graphs in the plane, so as to maximize some optimization criterion [CON], [CYN], [LNS], [T]. For a nonplanar graph G, one can start by drawing a planar subgraph of G, using some preferred method for drawing planar graphs [TBB], [BNTT], and then drawing the remaining edges. *Planarization* is the process of obtaining a planar graph from a nonplanar one

[1] The research of the first three authors was performed while at the Georgia Institute of Technology, and that by the fourth author while at the University of Virginia. The first and third authors were supported in part by NSF Grant CCR-9319106. The second author's research was supported in part by CNP$_q$ (Brazil) under Contract 200975/92-7, FAPESP (Proc. 96/04505-2), and ProNEx 107/97 MCT/FINEP. The fourth author was supported in part by the Packard Foundation.

[2] Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA. calinesc@iit.edu.

[3] Department of Computer Science, University of São Paulo, Rua do Matao 1010, 05508-900 Sao Paulo, Brazil. cris@ime.usp.br.

[4] AT&T Laboratories—Research, Room C231, 180 Park Ave., Florham Park, NJ 07932, USA. howard@research.att.com.

[5] Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA. alexz@cs.gsu.edu.

Received December 9, 2001; revised October 9, 2002. Communicated by S. Albers.
Online publication March 3, 2003.

[CNS], [OT], [JTS]. If different edges are not equally important, each edge might come with a weight reflecting the importance of drawing that edge; the edges in a maximum weight planar subgraph would be drawn first. Algorithms designed specifically for this weighted planar subgraph problem have been studied by Eades et al. [EFG], Jünger and Mutzel [JM1], Dyer et al. [DFF], and Foulds and Robinson [FR]. Two other references dealing with finding planar subgraphs are [JM2] and [L].

Another application of MAXIMUM WEIGHT PLANAR SUBGRAPH is *architectural floor planning*: the laying out, in some optimal or near-optimal way, of a set of facilities [DFF], [F], [FR], [M2]. Krejcirik [K] presented the first graph-theoretic approach to architectural floor planning, which sometimes yielded improved solutions [HW]. For each pair of facilities (i, j), m_{ij} denotes the desirability of placing i adjacent to j. The "desirability" of the layout is defined to be the sum of the m_{ij}'s of adjacent facilities i, j. The goal is to produce a layout with sum as high as possible. This problem can be modeled as an instance of MAXIMUM WEIGHT PLANAR SUBGRAPH: the vertices are the facilities and the weight of the edge between facilities i and j is m_{ij}.

Architectural floor planning—graph-theoretic and not—has traditionally been used in the design of industrial plants [DFF], [F], [FR], [M2], but also in the design of airports [BFP], hospitals [E], universities [DH], and libraries [FT]. Often m_{ij} is the number of people expected to travel from facility i to facility j.

Another important application of MAXIMUM WEIGHT PLANAR SUBGRAPH is *circuit layout* [M1], [M2]. The goal is to lay out a circuit on a one-layer board to maximize the number of connections between components. Now m_{ij} represents the number of desired connections between components i and j. The maximum weight planar subgraph gives a largest subset of connections that can be laid out without wire crossings on a board.

These applications justify the interest in solving MAXIMUM WEIGHT PLANAR SUBGRAPH. Unfortunately, because MAXIMUM WEIGHT PLANAR SUBGRAPH is NP-hard [LG], a polynomial-time exact algorithm is unlikely to exist. For this reason, we study approximation algorithms for MAXIMUM WEIGHT PLANAR SUBGRAPH.

We define the *performance ratio* of an algorithm A that takes weighted graphs G and produces planar subgraphs H of G as the infimum, over all weighted graphs G, of the ratio between the weight of H and the weight $Opt(G)$ of a heaviest planar subgraph of G; the performance ratio is at most one. In fact, MAXIMUM WEIGHT PLANAR SUBGRAPH is known to be Max SNP-hard even when all the edge weights are one [CFFK]. This means there is an $\varepsilon > 0$ such that no approximation algorithm achieves a ratio of $1 - \varepsilon$ for the problem, unless P = NP. The best (largest) known ε, however, is tiny, making $1 - \varepsilon$ very far from the best previously known performance ratio for MAXIMUM WEIGHT PLANAR SUBGRAPH, which was $1/3$ [C]. Any algorithm that produces a maximum weight spanning tree (see Section 3.4) achieves the $1/3$ ratio. See [DFF] for three algorithms for the weighted case and [CFFK] for a list of three others for the unweighted case, all having performance ratio at most $1/3$. In this paper we present the first algorithm for MAXIMUM WEIGHT PLANAR SUBGRAPH to cross the $1/3$ threshold: its performance ratio is at least $1/3 + 1/72$. We also prove that its performance ratio is at most $5/12$.

In the unweighted case, any algorithm producing any spanning tree of G has performance ratio $1/3$ (a spanning tree has $n - 1$ edges and a planar subgraph has at most $3n - 3$), and a recent paper exhibited a polynomial-time algorithm with a performance ratio of $4/9$ [CFFK]. However, the weighted case seems much more difficult. The key idea in the algorithms of [CFFK] is to produce a graph all of whose cycles are triangles (a *triangular structure*, which is necessarily planar), having many triangles. Each triangle gives one more edge than a spanning tree would have—the more triangles, the better. A simple algorithm greedily choosing triangles and then connecting them achieves $1/3+\varepsilon$ for a fixed positive ε [CFFK]. To get $4/9$, one finds a *largest* triangular structure in the input graph, which, surprisingly, can be done in polynomial time.

Unfortunately, the algorithms for the unweighted case do not work in the weighted case. First, one clearly cannot take arbitrary triangles, regardless of their weight. One must focus, of course, on heavy triangles. How? Recall that in the unweighted case, one can choose triangles (never creating any cycles of length exceeding three) and then arbitrarily tie together the components into a connected graph. In the weighted case, focusing on heavy triangles early on, without regard for the later step of *connecting* the triangles, might force the algorithm to use overly light edges to connect the triangles. Second, it is not known whether a *heaviest* triangular structure in G can be found in polynomial time. Graphical matroid parity is used to find a largest triangular structure in an unweighted graph [CFFK], graphical matroid parity being solvable in polynomial time [GS]; *weighted* graphical matroid parity is not known to be NP-hard or in P. Third, although in the unweighted case, proving that a heaviest (i.e., largest) triangular structure in G has weight at least $(\frac{1}{3}+\varepsilon)Opt(G)$ is not hard (indeed, a greedily constructed triangular structure suffices), whether the same property held for weighted graphs was not known. In fact, we can prove that this is so only indirectly by noting that our algorithm has performance ratio at least $1/3 + 1/72$, and that its output is a triangular structure.

For most approximation algorithms proposed recently, the performance ratios for the weighted cases are the same as those for the unweighted cases [CST], [S], [GW], [FG], [H2], [H3]. We, however, can prove only that the performance ratio for MAXIMUM WEIGHT PLANAR SUBGRAPH is at least $1/3 + 1/72$, whereas the performance ratio for MAXIMUM WEIGHT PLANAR SUBGRAPH in the unit-weight case is at least $4/9$. Of course, these are only the best bounds *known*. We leave to the reader the (open) problem of proving that the best *possible* performance ratios are the same.

Our algorithm provides two other results. First, in the important special case in which G is a complete graph whose edge weights satisfy the triangle inequality, we can prove a better performance ratio of $3/8 = 0.375$. Second, since our algorithm produces triangular structures and any triangular structure is outerplanar (i.e., can be drawn in the plane with all vertices on the boundary of the outer face), we have an approximation algorithm for MAXIMUM WEIGHT OUTERPLANAR SUBGRAPH. We can prove a performance ratio of at least $7/12$, whereas producing a spanning tree gives a performance ratio of $1/2$. (The best ratio known in the unweighted case is $2/3$ [CFFK].)

Our algorithm exploits an analogy between Steiner trees and triangular structures. Modeled on the two-phase Berman–Ramaiyer Steiner tree algorithm [BR], the algorithm is clearly polynomial time. However, its analysis is difficult. A "road map" guiding the reader through the analysis appears in Section 3.1.

2. The Algorithm.

MAXIMUM WEIGHT PLANAR SUBGRAPH is this problem: given a graph G with a nonnegative weight defined for each edge, find a planar subgraph of G of maximum weight.

Let $G = (V, E)$ be a connected graph with a nonnegative weight function w on the edges. Recall that a triangular structure is a graph all of whose cycles (if any) are triangles. A *triangular structure in a graph* G is a subgraph of G which is a triangular structure. We will describe an algorithm which produces a heavy triangular structure in G.

Before precisely describing the algorithm, we need some notation. The vertex sets in all graphs will be obvious, so we will identify a graph with its edge set.

The algorithm has two phases, the *evaluation phase* and the *construction phase*. The evaluation phase (executed first) maintains a tree whose edges are not only from the original graph, but also some *artificial* edges. The artificial edges have weight, also denoted by w. We denote by N_i the set of two artificial edges added in the ith augmentation (which will be defined soon).

We start with T_0, a maximum spanning tree in $G = G_0$. We will maintain the invariant that T_i is a maximum spanning tree in $G_i = (V, E \cup (\bigcup_{j=1}^{i} N_j))$. G_i is, in fact, a multigraph.

Let Δ be a triangle of G, the original graph. In the spanning tree T_{i-1}, there is a unique vertex s such that the paths in T_{i-1} from s to each of the three vertices of Δ are edge-disjoint. For each of these three paths (at most one of which may be trivial), choose an edge of minimum weight on the path, if possible. Now label the vertices of Δ as x, y, z such that the weight of the minimum weight edge e_1 on the s–x path is at most the weight of the minimum weight edge e_2 on the s–y path, which is at most the weight of the minimum weight edge e_3 on the s–z path. (If this last path is trivial, leave e_3 undefined and, by an abuse of notation, say that $w(e_3) = \infty$.) The edges of the triangle Δ are unnamed.

One can easily check that e_1 and e_2 are two edges of T_{i-1} such that each of the three connected components of $T_{i-1} \backslash \{e_1, e_2\}$ contains one vertex of Δ. Moreover, among all the choices of pairs (e, f) of edges of T_{i-1} whose removal pairwise disconnects x, y, and z, the pair (e_1, e_2) has the least total weight. We define the *gain* $g_{i-1}(\Delta)$ *of the triangle* Δ *with respect to* T_{i-1} to be $g_{i-1}(\Delta) = w(\Delta) - w(e_1) - w(e_2)$, where by $w(\Delta)$ we mean the sum of the weights of the three (original) edges of the triangle Δ.

We call the following operation *augmenting the tree* T_{i-1} *by the triangle* Δ. This is done only if $g_{i-1}(\Delta) > 0$, so we assume now that $g_{i-1}(\Delta) > 0$. An augmentation produces a tree T_i of greater weight.

Let e_1' be a new artificial edge with endpoints x and z and $w(e_1') = w(e_1) + g_{i-1}(\Delta)$. Let e_2' be a new artificial edge with endpoints y and z and $w(e_2') = w(e_2) + g_{i-1}(\Delta)$. Note that both e_1' and e_2' are parallel to existing (unnamed) edges of the triangle Δ. Let $N_i = \{e_1', e_2'\}$. Then replace e_1 by e_1' and replace e_2 by e_2' to get T_i: $T_i = (T_{i-1} \backslash \{e_1, e_2\}) \cup N_i$.

Note that e_1 and e_2 might themselves be artificial edges. Because vertices x, y, z are in separate components of $T_{i-1} \backslash \{e_1, e_2\}$, T_i is a tree. We will see later that T_i is a maximum spanning tree of G_i.

2.1. *The Evaluation Phase.*

The evaluation phase starts with a maximum spanning tree T_0 in $G_0 = G$, and sets $\mathcal{T}$ to be the set of all triangles in G. Repeatedly, the algorithm removes any one triangle Δ from $\mathcal{T}$, and calculates its gain $g_{i-1}(\Delta)$ relative to the current tree T_{i-1}. If nonpositive, it does nothing, while if positive, it augments T_{i-1} by Δ to get T_i (and N_i), sets $\Delta_i \leftarrow \Delta$, and increments i. This process continues until $\mathcal{T}$ is exhausted.

Here is the evaluation phase.

Evaluation Phase

```
1   Let $T_0$ be a maximum spanning tree in $G$; $i \leftarrow 1$
2   Let $\mathcal{T}$ be the set of all triangles in $G$
3   while $\mathcal{T} \neq \emptyset$ do
4      Choose any $\Delta \in \mathcal{T}$
5      if $g_{i-1}(\Delta) > 0$ then
6         Augment $T_{i-1}$ by $\Delta$ to get $T_i$ and $N_i$
7         $\Delta_i \leftarrow \Delta$
8         $i \leftarrow i + 1$
9      endif
10     $\mathcal{T} \leftarrow \mathcal{T} \backslash \{\Delta\}$
11  endwhile
12  $f \leftarrow i - 1$.
```

2.2. *The Construction Phase.* The goal of the construction phase is to use the spanning tree T_f, the sets $N_f, N_{f-1}, N_{f-2}, \ldots, N_1$, and the triangles $\Delta_f, \Delta_{f-1}, \Delta_{f-2}, \ldots, \Delta_1$ from the evaluation phase to build a connected triangular structure in $G_0 = G$. This is done iteratively by maintaining a connected triangular structure S_i in G_i, for $i = f, f-1, f-2, \ldots, 0$, where $G_i = (V, E \cup (\bigcup_{j=1}^{i} N_j))$. The spanning tree T_f in G_f serves as the connected triangular structure S_f in G_f to get us started. From that point on, in the ith iteration ($i = f, f-1, \ldots, 1$), consider the two-element set N_i. We must ensure that $N_i \cap S_{i-1} = \emptyset$, since S_{i-1} must be a subgraph of G_{i-1} and the edges of N_i do not appear in G_{i-1}. What happens next depends on $|N_i \cap S_i|$. If $|N_i \cap S_i| = 0$, we simply set $S_{i-1} \leftarrow S_i$; $|S_{i-1}| = |S_i|$. If $|N_i \cap S_i| = 1$, then we set $S_{i-1} \leftarrow (S_i \backslash N_i) \cup \{h\}$, where h is a heavy edge chosen to guarantee that S_{i-1} is connected; again, $|S_{i-1}| = |S_i|$. The most interesting case is the one in which $|N_i \cap S_i| = 2$, that is, $N_i \subseteq S_i$. Then we set $S_{i-1} \leftarrow (S_i \backslash N_i) \cup \Delta_i$. Since $|\Delta_i| = 3$, we have $|S_{i-1}| = |S_i| + 1$. It is through this step that we add triangles to the final S_0 and improve on the maximum spanning tree.

Here is the construction phase.

Construction Phase

```
1   $S_f \leftarrow T_f$
2   for $i = f$ downto 1 do
3      if $|N_i \cap S_i| = 0$ then $S_{i-1} \leftarrow S_i$        /* $|S_{i-1}| = |S_i|$ */
4      if $|N_i \cap S_i| = 1$ then
5         $S_i' \leftarrow S_i \backslash N_i$                       /* $S_i'$ has two connected components */
6         Find edge $h$ of $G_{i-1}$ of maximum
          weight with endpoints in different
          components of $S_i'$
7         $S_{i-1} \leftarrow S_i' \cup \{h\}$                     /* $|S_{i-1}| = |S_i|$ */
8      endif
9      if $|N_i \cap S_i| = 2$
       then $S_{i-1} \leftarrow (S_i \backslash N_i) \cup \Delta_i$     /* $|S_{i-1}| = |S_i| + 1$ */
10  endfor
11  Output $S_0$
```

3. Analysis

3.1. *Road Map.* The output of our algorithm, when the input is a weighted connected graph G, is S_0. If S is a subgraph of G or a set of edges of G, then we denote by $w(S)$ the sum of $w(e)$ over all e in S. Our analysis proceeds as follows:

1. First, we prove that S_0 is a (connected) triangular structure in G (Section 3.2). This ensures that the output is a planar subgraph of G.
2. We prove in Section 3.3 that the weight of S_0 is at least the average of the weight of T_0, a maximum spanning tree in G, and the weight of T_f, a spanning tree (indeed, *maximum* spanning tree) in the supergraph G_f of G obtained during the evaluation phase.

   We will see shortly that $w(T_0) \geq \frac{1}{3}Opt(G)$, where $Opt(G)$ is the weight of a heaviest planar subgraph of G. We circuitously prove that $w(S_0) \geq (\frac{1}{3} + \frac{1}{72})Opt(G)$ via two lower bounds on $w(T_0)$ (the trivial bound that $w(T_0) \geq \frac{1}{3}Opt(G)$ and a more difficult one), and an easy lower bound on $w(T_f)$.
3. In Section 3.4 we prove the trivial fact that $w(T_0) \geq \frac{1}{3}w(P)$ for any planar subgraph P of G.
4. Section 3.5 contains a plethora of lemmas needed to prove the easy lower bound on $w(T_f)$.
5. In Section 3.6 we introduce the concept of a "good" set $\mathcal{S}$ of triangles in a plane graph P. Let $b(\Delta)$ denote the median weight of the edges in triangle Δ. Section 3.6.1 provides a proof of the easy lower bound on $w(T_f)$:

$$w(T_f) \geq \tfrac{1}{3}w(P) + \tfrac{1}{12}\left[\sum_{\Delta\in\mathcal{S}} b(\Delta)\right]$$

   provided only that $\mathcal{S}$ is a good set of triangles in a plane subgraph P of G.
6. In Section 3.6.2 the reader will find the technically most important fact of the analysis, Corollary 20, which has Lemma 19 as its crux. Corollary 20 states that if P is a weighted plane subgraph of G, then there is a good family $\mathcal{S}$ of triangles of P such that

$$w(T_0) \geq \tfrac{1}{2}w(P) - \tfrac{1}{2}\left[\sum_{\Delta\in\mathcal{S}} b(\Delta)\right].$$

   This is the difficult lower bound.
7. Letting P be any plane subgraph of G and letting $\beta = \sum_{\Delta\in\mathcal{S}} b(\Delta)$, for the good family $\mathcal{S}$ of triangles of P given by Corollary 20, we have
   (a) $w(T_0) \geq \frac{1}{3}w(P)$,
   (b) $w(T_0) \geq \frac{1}{2}w(P) - \frac{1}{2}\beta$, and
   (c) $w(T_f) \geq \frac{1}{3}w(P) + \frac{1}{12}\beta$.

   The last two conditions and $w(S_0) \geq \frac{1}{2}[w(T_0) + w(T_f)]$ say that $w(S_0)$ is at least $(\frac{1}{3}+\varepsilon)w(P)$ unless β is large. However, when β is large, the first and third conditions, taken with $w(S_0) \geq \frac{1}{2}[w(T_0) + w(T_f)]$, say that $w(S_0) \geq (\frac{1}{3} + \varepsilon)w(P)$. The formal statement that

$$\tfrac{1}{2}[w(T_0) + w(T_f)] \geq (\tfrac{1}{3} + \tfrac{1}{72})w(P),$$

whatever the value of β, appears in Section 3.7. Taking P to be a heaviest planar subgraph of G, we are done.

8. Section 3.9 proves that the performance ratio is at least $3/8$ if the edge weights in the input graph G satisfy the triangle inequality.
9. Section 3.10 gives a $7/12$ bound for the problem of finding a heaviest outerplanar subgraph in G. A ratio of $1/2$ is obtained by any maximum spanning tree.

3.2. *S_0 is a Connected Triangular Structure in G.* We concentrate on the construction phase. When going from S_i to S_{i-1}, we make sure that $S_{i-1} \cap N_i = \emptyset$; thus S_{i-1} is a subgraph of G_{i-1}. In particular, S_0 is a subgraph of $G = G_0$. The fact that S_i is a connected triangular structure for all i follows from the following lemma.

LEMMA 1. *For all $i \in \{f, f-1, f-2, \ldots, 0\}$, S_i is a connected graph, all of whose cycles are triangles in the original graph G.*

PROOF. S_f is a spanning tree, so for $i = f$, the lemma is true. We assume that the lemma holds for some i, $f \geq i \geq 1$, and prove it holds for $i - 1$. We follow the cases of each iteration of the construction phase.

If $|N_i \cap S_i| = 0$, then $S_{i-1} = S_i$, which has the desired properties.

Suppose now that $|N_i \cap S_i| = 1$. Then S_i' is obtained from S_i by removing an artificial edge. This edge, being artificial, by the inductive hypothesis does not belong to any cycle. It follows that S_i' has two components. Then in line 7 one edge is added between the two components of S_i'. Hence this new edge does not belong to any cycle of S_{i-1} and therefore all cycles of S_{i-1} are cycles of S_i. Also, S_{i-1} is connected.

Suppose for the remainder of the proof that $|N_i \cap S_i| = 2$. We use the notation from the definition of augmentation. $N_i = \{e_1', e_2'\}$, where e_1' has endpoints x and z and e_2' has endpoints y and z. By the inductive hypothesis, since both e_1' and e_2' are artificial edges, neither appears in any cycle of S_i. Let f_1, f_2, and e be the edges of the original (simple) graph with endpoints x and z, y and z, and x and y, respectively. Consider $S' = (S_i \backslash N_i) \cup \{f_1, f_2\}$ (the two edges of N_i are replaced by two edges parallel to them). Then in S' all cycles are triangles of the original graph G and the edges f_1 and f_2 do not appear in any cycle. We have $S_{i-1} = S' \cup \{e\}$.

We know that before adding e, f_1 and f_2 did not appear in any cycle. We will show that adding e only creates the cycle $\langle f_1, f_2, e\rangle$. Any cycle we might create by adding e has to contain e, of course. Suppose, for a contradiction, that adding e creates a cycle C other than $\langle f_1, f_2, e\rangle$. Assume first that C contains neither f_1 nor f_2. By replacing e in C by f_1 and f_2, we either get a cycle containing f_1 and f_2 but not e, a contradiction, or a union of two cycles (this is the case in which C goes through the common endpoint of f_1 and f_2). However, one of these cycles contains f_1 and does not contain e, a contradiction. If C contains f_1 and e, but not f_2, then by replacing f_1 and e in C by f_2, we get a cycle containing f_2 and not e, contradicting the fact that before adding e, f_1 and f_2 did not appear in any cycle. The case in which C contains f_2 and e, but not f_1, is similar.

It follows that in S_{i-1} all cycles are triangles of the original graph. Also, since S_i is connected, S' is connected and therefore S_{i-1} is connected. □

COROLLARY 2. *S_0 is a connected triangular structure in G and hence planar.*

3.3. *A Lower Bound on the Weight of S_0.* We need the following lemma and its corollary, analogues of Steiner tree Lemmas 3.5 and 3.7 in [BR].

LEMMA 3. *For $i = 1, \dots, f$, $w(S_{i-1}) \geq w(S_i) - g_{i-1}(\Delta_i)$.*

PROOF. The proof follows the three cases from each iteration of the construction phase. As above, let $\Delta_i = xyz$ and let e_1 and e_2 be the two edges removed from T_{i-1} during the ith augmentation. $N_i = \{e'_1, e'_2\}$ is the set of artificial edges added during the ith augmentation. Also $w(e'_1) = w(e_1) + g_{i-1}(\Delta_i)$ and $w(e'_2) = w(e_2) + g_{i-1}(\Delta_i)$.

If $|N_i \cap S_i| = 0$, then $S_{i-1} = S_i$ and the lemma follows from the fact that $g_{i-1}(\Delta_i) > 0$.

If $|N_i \cap S_i| = 2$, then $w(S_{i-1}) = w(S_i) - w(e'_1) - w(e'_2) + w(\Delta_i) = w(S_i) - w(e_1) - g_{i-1}(\Delta_i) - w(e_2) - g_{i-1}(\Delta_i) + w(\Delta_i) = w(S_i) - g_{i-1}(\Delta_i)$, since $g_{i-1}(\Delta_i) = w(\Delta_i) - w(e_1) - w(e_2)$.

Now suppose that $|N_i \cap S_i| = 1$. We only consider the case $N_i \cap S_i = \{e'_1\}$; in the case $N_i \cap S_i = \{e'_2\}$, a similar argument applies. The endpoints of e'_1 are x and z and therefore in $S'_i = S_i \backslash \{e'_1\}$, x and z are in two different components (by Lemma 1). Consider the (unique) path in T_{i-1} from x to z. There is at least one edge on this path which has endpoints in two different components of S'_i; choose such an edge and call it e. We conclude that the edge h selected by the algorithm in the construction phase satisfies $w(h) \geq w(e)$. Note also that e_1 is the minimum weight edge on the path in T_{i-1} from x to z, so $w(e_1) \leq w(e)$. Then we have $w(h) \geq w(e_1) = w(e'_1) - g_{i-1}(\Delta_i)$. We conclude that $w(S_{i-1}) = w(S_i) - w(e'_1) + w(h) \geq w(S_i) - g_{i-1}(\Delta_i)$. □

COROLLARY 4. $w(S_0) \geq \frac{1}{2}(w(T_0) + w(T_f))$.

PROOF. It is immediate that for $i = 1, \dots, f$, $w(T_i) = w(T_{i-1}) + 2g_{i-1}(\Delta_i)$, and thus $w(T_f) = w(T_0) + 2\sum_{i=1}^{f} g_{i-1}(\Delta_i)$. It follows, by Lemma 3, that $w(S_0) \geq w(S_f) - \sum_{i=1}^{f} g_{i-1}(\Delta_i) = w(T_f) - \sum_{i=1}^{f} g_{i-1}(\Delta_i) = w(T_0) + \sum_{i=1}^{f} g_{i-1}(\Delta_i) = \frac{1}{2}(w(T_0) + w(T_f))$. □

3.4. *A Trivial Lower Bound on the Weight of a Maximum Spanning Forest.* The proof of the next lemma uses the concept of *arboricity*, defined below, and a theorem by Nash-Williams.

DEFINITION 1. The *arboricity* of a graph is the minimum number of spanning forests into which its edge set can be partitioned.

We have this classic theorem:

THEOREM 5 (Nash-Williams; see [N] and [CMW$^+$]). *The arboricity of a graph G is the maximum, over all subgraphs H of G with at least two vertices, of*

$$\left\lceil \frac{|E(H)|}{|V(H)|-1} \right\rceil .$$

LEMMA 6. *Let $\mathcal{F}$ be a family of graphs closed under taking subgraphs, such that, for some positive integer c, $|E(G)| \leq c(|V(G)|-1)$ for all G in $\mathcal{F}$. Let w be a nonnegative weight function on the edges of some G in $\mathcal{F}$, and let F be a maximum spanning forest in G. Then $w(F) \geq (1/c)w(G)$.*

PROOF. Let G be a graph in $\mathcal{F}$. We have $|E(H)| \leq c(|V(H)|-1)$ for any subgraph H of G, because $\mathcal{F}$ is closed under taking subgraphs. Thus, by Theorem 5, the arboricity of G is at most c.

This means that the edge set of G can be partitioned into c forests $F_1, \ldots, F_c$. Clearly we have $w(F_1) + \cdots + w(F_c) = w(G)$. However, $w(F) \geq w(F_i)$ for all i, and this implies that $w(G) = w(F_1) + \cdots + w(F_c) \leq c \cdot w(F)$. □

The proof method of Lemma 17 gives an alternative way to prove Lemma 6 without relying on arboricity.

COROLLARY 7 (see Theorem 3.3 of [DFF]). *Let P be a simple planar graph with a nonnegative weight function w defined on its edges. If F is a maximum spanning forest in P, then $w(F) \geq \frac{1}{3}w(P)$.*

PROOF. Apply Euler's formula ($|E(P)| \leq 3(|V(P)|-1)$) and Lemma 6 with $c = 3$ to the family of simple planar graphs. □

Recall that T_0 is a maximum spanning tree in G. As a consequence of Corollary 7, we have $w(T_0) \geq \frac{1}{3}w(P)$, for any planar subgraph P of G.

3.5. *Some Lemmas*

DEFINITION 2. If T is a spanning tree in a connected multigraph H, and c and d are vertices of H, let $index_T(c, d)$ denote the weight of the lightest edge on the c–d path in T.

Lemmas 8–12 are used solely in the proof of Lemma 13.

LEMMA 8. *Let H be a connected multigraph and let T be a maximum spanning tree in H. Let $a \neq b$ be two vertices. Let e be a minimum weight edge on the a–b path in T. Let e' be a new edge with endpoints a, b and of weight $w(e') \geq w(e) = index_T(a, b)$.*

Define $H' = H \cup \{e'\}$ *and* $T' = (T \cup \{e'\})\backslash\{e\}$. *Then* T' *is a spanning tree in* H',

1. *for all vertices* u, v, $index_{T'}(u, v) \geq index_T(u, v)$, *and*
2. T' *is a maximum spanning tree in* H'.

PROOF. That T' is a spanning tree in H' is obvious.

Let C be the unique cycle in $T \cup \{e'\} = T' \cup \{e\}$. Let u and v be two vertices and let P and P' be the unique paths in T and T', respectively, between u and v. Let us prove part 1.

If P does not contain e, then $P = P'$ and therefore $index_{T'}(u, v) = index_T(u, v)$.

Assume now that P contains e. Then P is composed of three paths: P_1, from u to the first vertex of C on P; P_2, a segment of C containing e; and P_3, from the last vertex of C on P to v. P_1 or P_3 may be trivial. Similarly, P' is composed of three paths: P_1, P_2' and P_3, where P_2' is a segment of C containing e'. Let f be an edge of minimum weight on P', so that $w(f) = index_{T'}(u, v)$.

If f is on P_1 or P_3, then f is on P and therefore $index_{T'}(u, v) = w(f) \geq index_T(u, v)$.

If f is on P_2', then $w(f) \geq w(e)$ (e is the minimum weight edge on C since $w(e') \geq w(e)$). Since e is on P, $w(e) \geq index_T(u, v)$ and therefore $w(f) \geq index_T(u, v)$. Hence $index_{T'}(u, v) \geq index_T(u, v)$. The proof of part 1 is complete.

The statement that T' is a maximum spanning tree in H' is equivalent to the statement that T' is a spanning tree in H' and for any edge f of H' with, say, endpoints u and v, $w(f) \leq index_{T'}(u, v)$; an analogous statement holds for T and H [AMO, Theorem 3.3].

Because T is a maximum spanning tree in H, for all edges f in H, with, say, endpoints u and v, $w(f) \leq index_T(u, v) \leq index_{T'}(u, v)$. Also, $w(e') = index_{T'}(a, b)$, because e' has endpoints a and b, and $e' \in T'$. It follows that for all $f \in H'$, with endpoints u, v, $w(f) \leq index_{T'}(u, v)$. This makes T' a maximum spanning tree in H'. □

LEMMA 9. *For* $i = 0, 1, 2, \ldots, f$, T_i *is a maximum spanning tree in* G_i, *and for all* $i \geq 1$ *and all vertices* $u \neq v$, $index_{T_{i-1}}(u, v) \leq index_{T_i}(u, v)$.

PROOF. By induction on i, the case of $i = 0$ being trivial. Choose $i \geq 1$ and assume that T_{i-1} is a maximum spanning tree in G_{i-1}.

We can think of the augmentation of the tree T_{i-1} by a triangle Δ_i of positive gain $g_{i-1}(\Delta_i)$ as taking place in two steps. In the first step, a new artificial edge e_1' with endpoints x, z and weight $w(e_1') = w(e_1) + g_{i-1}(\Delta_i) > w(e_1)$ is added to the tree T_{i-1} and to G_{i-1}, and then e_1 is removed from the tree. Let $T_{i-1}' = (T_{i-1} \cup \{e_1'\})\backslash\{e_1\}$ be the resulting tree and let $G_{i-1}' = G_{i-1} \cup \{e_1'\}$ be the resulting graph. In the second step, an artificial edge e_2' with endpoints y, z and weight $w(e_2') = w(e_2) + g_{i-1}(\Delta_i) > w(e_2)$ is added to T_{i-1}' and to G_{i-1}'; next, e_2 is removed from the tree. $T_i = (T_{i-1}' \cup \{e_2'\})\backslash\{e_2\}$ is the resulting tree and $G_i = G_{i-1}' \cup \{e_2'\}$ is the resulting graph.

We prove the lemma by applying Lemma 8 twice. The first time, we set $H = G_{i-1}$, $T = T_{i-1}$, $a = x$, $b = z$, $e = e_1$, $e' = e_1'$. We have $w(e') > w(e)$, as needed. $T' = (T \cup \{e_1'\})\backslash\{e_1\} = T_{i-1}'$ and $H' = G_{i-1}'$. We conclude straightforwardly that

$$index_{T_{i-1}'}(u, v) = index_{T'}(u, v) \geq index_T(u, v) = index_{T_{i-1}}(u, v)$$

and that T_{i-1}' is a maximum spanning tree in G_{i-1}'.

The second time, we let $H = G'_{i-1}$, $T = T'_{i-1}$, $a = y$, $b = z$, and $e = e_2$. We need e_2 to be a minimum weight edge on the y–z path in T'_{i-1}, but this holds since the y–z paths in T_{i-1} and T'_{i-1} are the same. We set $e' = e'_2$, with endpoints y, z. We have $w(e'_2) > w(e_2)$, as needed. Conveniently, $T' = (T'_{i-1} \cup \{e'_2\}) \backslash \{e_2\} = T_i$ and $H' = G'_{i-1} \cup \{e'_2\} = G_i$. Now $index_{T_i}(u, v) \geq index_{T'_{i-1}}(u, v) \geq index_{T_{i-1}}(u, v)$ and T_i is a maximum spanning tree in G_i. □

In what follows we use $index_i(u, v)$ to denote $index_{T_i}(u, v)$.

LEMMA 10. *Fix any triangle* Δ *of* G. *For all* $i = 1, \ldots, f$, $g_i(\Delta) \leq g_{i-1}(\Delta)$.

PROOF. We use notation from the definition of the gain of a triangle $\Delta = xyz$. We have $index_{i-1}(x, y) = index_{i-1}(x, z) = w(e_1)$ and $index_{i-1}(y, z) = w(e_2) \geq w(e_1)$. Then

$$\begin{aligned} g_{i-1}(\Delta) &= w(\Delta) - index_{i-1}(x, y) - index_{i-1}(y, z) \\ &= w(\Delta) - \min\{index_{i-1}(x, y), index_{i-1}(x, z), index_{i-1}(y, z)\} \\ &\quad - \max\{index_{i-1}(x, y), index_{i-1}(x, z), index_{i-1}(y, z)\}. \end{aligned}$$

In other words, to obtain the gain of a triangle we subtract from the weight of the triangle the highest and lowest indices between two vertices of the triangle (the two lowest indices being equal). Since the index function cannot decrease as i increases (Lemma 9), the gain of a triangle cannot increase as i increases. □

LEMMA 11. *For any triangle* Δ, *there is some* $i \in \{0, \ldots, f\}$ *such that* $g_i(\Delta) \leq 0$.

PROOF. This is certainly true if at the moment Δ was considered during the evaluation phase, $g_i(\Delta) \leq 0$, or in other words, Δ was ignored and not used for an augmentation. So in the following, assume $\Delta = xyz$ was used during the ith augmentation. In other words, $\Delta = \Delta_i$. Therefore $g_{i-1}(\Delta) > 0$; and in this case, we prove that $g_i(\Delta) < 0$. One can easily verify that $N_i = \{e'_1, e'_2\}$, the set of artificial edges added during the ith augmentation, is the only set of two edges of T_i with the property that their deletion pairwise disconnects x, y, and z. Then, because in calculating the gain we subtract the weight of two edges whose deletion pairwise disconnects x, y, and z, and because only the pair $\{e'_1, e'_2\}$ has this property, $g_i(\Delta) = w(\Delta) - w(e'_1) - w(e'_2) = w(\Delta) - (w(e_1) + g_{i-1}(\Delta)) - (w(e_2) + g_{i-1}(\Delta)) = -g_{i-1}(\Delta) < 0$, since $g_{i-1}(\Delta) = w(\Delta) - w(e_1) - w(e_2)$. □

The following lemma is an analogue of Lemma 3.10 of [BR].

LEMMA 12. *No triangle of the original graph* G *has positive gain with respect to* T_f.

PROOF. The previous two lemmas prove that for all Δ, $g_f(\Delta) \leq 0$. □

Let $P = (V, E_P)$ be any planar subgraph of $G = (V, E)$. Define a weight function w' on the edges of P by $w'(xy) = \mathit{index}_f(x, y)$ for all edges $xy \in E_P$. (Recall that $\mathit{index}_f(x, y)$ is the weight of the lightest edge in the unique x–y path in T_f.)

If we refer to a specific triangle Δ of P, we denote by $a(\Delta) \geq b(\Delta) \geq c(\Delta)$ the weights, with respect to w, of the edges of the triangle. Similarly, $a'(\Delta) \geq b'(\Delta) \geq c'(\Delta)$ are the weights of the edges with respect to w' (where the edge of weight $a(\Delta)$ need not be the one of weight $a'(\Delta)$, etc.).

LEMMA 13. *The weight function w' on the edges of P has the following three properties*:

1. *For any edge $e \in E_P$, $w'(e) \geq w(e)$.*
2. *For any triangle Δ of P, $a'(\Delta)+b'(\Delta) \geq a(\Delta)+b(\Delta)+c(\Delta)$. Also $c'(\Delta) = b'(\Delta)$.*
3. $w'(\Delta) - w(\Delta) = [a'(\Delta) + b'(\Delta) + c'(\Delta)] - [a(\Delta) + b(\Delta) + c(\Delta)] \geq b(\Delta)$.

PROOF. Let $e = xy \in E_P \subseteq E \subseteq E(G_f)$. Since T_f is a maximum spanning tree in G_f, we have $w(e) \leq \mathit{index}_f(x, y) = w'(e)$.

Fix a triangle $\Delta = xyz$ of P. That $c'(\Delta) = b'(\Delta)$ follows immediately from the properties of the index function (see the proof of Lemma 10). Also, we know (by Lemma 12) that Δ has nonpositive gain with respect to T_f. By the method of computing the gain using the index function (again see the proof of Lemma 10), we have $0 \geq g_f(\Delta) = w(\Delta) - \max\{\mathit{index}_f(x, y), \mathit{index}_f(x, z), \mathit{index}_f(y, z)\} - \min\{\mathit{index}_f(x, y), \mathit{index}_f(x, z), \mathit{index}_f(y, z)\} = w(\Delta) - \max\{w'(xy), w'(xz), w'(yz)\} - \min\{w'(xy), w'(xz), w'(yz)\} = w(\Delta) - a'(\Delta) - c'(\Delta) = w(\Delta) - a'(\Delta) - b'(\Delta)$, and it follows that $a'(\Delta) + b'(\Delta) \geq w(\Delta) = a(\Delta) + b(\Delta) + c(\Delta)$.

The third statement is obvious. □

LEMMA 14. *The weight $w(T_f) \geq \frac{1}{3}w'(P)$.*

PROOF. Consider the multigraph $H = (V, E_P \cup T_f)$, where we assign weight $w'(e)$ if $e \in E_P$ and $w(e)$ if $e \in T_f$; note that if $e \in E_P \cap T_f$ and e has endpoints x and y, then $w'(e) = \mathit{index}_f(x, y)$ because $e \in E_P$, and $\mathit{index}_f(x, y) = w(e)$, because $e \in T_f$.

We prove that T_f is a maximum spanning tree in multigraph H. Consider any edge e in H with endpoints, say, x and y. If $e \in E_P$, we have $w'(e) = \mathit{index}_f(x, y)$. If $e \in T_f$, we clearly have $w(e) = \mathit{index}_f(x, y)$. So in either case, the weight of an edge with endpoints x, y is at most $\mathit{index}_f(x, y)$; this makes T_f a maximum spanning tree in H.

Since P is a subgraph of H and T_f is a maximum spanning tree of H, the weight of T_f is at least the weight of a maximum spanning tree in P (with weights given by w'). Since Corollary 7 implies that any maximum spanning tree in P has weight at least $\frac{1}{3}w'(P)$, we have $w(T_f) \geq \frac{1}{3}w'(P)$. □

3.6. *Two Lower Bounds.* All graphs considered in this section are simple. We prove two bounds in this section. Both rely on the concept of a "good" set of triangles in the original graph G. The first bound, $w(T_f) \geq \frac{1}{3}w(P) + \frac{1}{12}[\sum_{\Delta \in S} b(\Delta)]$ for any good set S of triangles in P (Corollary 16), is easy and follows directly from the definition of a good set of triangles. The second, $w(T_0) \geq \frac{1}{2}w(P) - \frac{1}{2}[\sum_{\Delta \in S} b(\Delta)]$ (Corollary 20), is more difficult to prove and only holds for a carefully chosen good set $\mathcal{S}$ of triangles in P.

DEFINITION 3. A *straight-line plane graph* is a plane graph whose edges are drawn as line segments.

DEFINITION 4. We call a set $\mathcal{S}$ of triangles of a straight-line plane graph P a *good family* if every edge of P participates in at most two triangles of $\mathcal{S}$ on each side of the line containing the edge.

3.6.1. *The (Easy) Lower Bound on* $w(T_f)$. We need the following lemma. Recall that P is a planar subgraph of G, and $w'(xy) = index_f(x, y)$ for all edges $xy \in E(P)$.

LEMMA 15. *Let $\mathcal{S}$ be any good family of triangles in the straight-line plane graph P. Then $w'(P) \geq w(P) + \frac{1}{4}[\sum_{\Delta\in\mathcal{S}} b(\Delta)]$.*

PROOF. Let $m(e)$ be the number of triangles of $\mathcal{S}$ in which e participates. Because $\mathcal{S}$ is good, we have $m(e) \leq 2 \cdot 2 = 4$ for all e in G. Using Lemma 13, we have

$$\begin{aligned}\sum_{\Delta\in\mathcal{S}} b(\Delta) &\leq \sum_{\Delta\in\mathcal{S}}(w'(\Delta) - w(\Delta)) = \sum_{\Delta\in\mathcal{S}}\sum_{e\in\Delta}(w'(e) - w(e))\\ &= \sum_{e\in E(P)} (w'(e) - w(e))m(e) \leq \sum_{e\in E(P)} 4(w'(e) - w(e))\\ &= 4(w'(P) - w(P)).\end{aligned}$$ □

COROLLARY 16. *Let $\mathcal{S}$ be any good family of triangles in a straight-line plane graph P which is a subgraph of $G = G_0$. Then*

$$w(T_f) \geq \tfrac{1}{3}w(P) + \tfrac{1}{12}\left[\sum_{\Delta\in\mathcal{S}} b(\Delta)\right].$$

PROOF. Combine Lemmas 14 and 15. □

3.6.2. *The (Difficult) Lower Bound on* $w(T_0)$. We will need the following lemma, whose proof uses an idea from [KH].

LEMMA 17. *Let P be a triangle-free planar graph with a nonnegative weight function c defined on its edges, such that edges xz and yz in P have weight zero. Then there is a spanning forest F in P such that $c(F) \geq \frac{1}{2}c(P)$ and such that x, y, and z are in three different components of F.*

PROOF. We use the greedy algorithm to construct F, first sorting the edges of P into nonincreasing order by weight, with edges xz and yz at the end. Let E_i be the set of the first i edges in this ordering, $1 \leq i \leq m$, $m = |E(P)|$. By c_i we denote the weight of the ith edge in this ordering. Starting with $F = \emptyset$, the greedy algorithm scans the edges in the given order and adds an edge to F as long as it satisfies the following conditions:

1. It does not create any cycles.
2. It does not create a path connecting any two of x, y, and z.

Let F be the set of edges chosen by the greedy algorithm and let $F_i = E_i \cap F$. Then

$$c(F) = \sum_{i=1}^{m-2} |F_i|(c_i - c_{i+1}),$$

$$c(P) = \sum_{i=1}^{m-2} |E_i|(c_i - c_{i+1}).$$

Note that the index only goes to $m-2$. It is therefore enough to show that $|E_i| \le 2|F_i|$ for $1 \le i \le m-2$.

Choose an $i \le m-2$. We assume that the components of F_i have $p_1, p_2, \ldots, p_k$ vertices, where each of x, y, and z belongs to one of the first three components. Of course, $|F_i| = \sum_{j=1}^{k}(p_j - 1)$.

We call the set of vertices in the first three components *supercomponent* 123. Note that no edge of E_i can have one endpoint in a component $j > 3$ and the other endpoint in another component $j' > 3$, since some such edge would have been selected by the greedy algorithm, thus merging two components. Similarly, no edge of E_i can have exactly one endpoint in the supercomponent 123. We associate each edge of E_i with the component $j > 3$ or with the supercomponent 123 containing both endpoints of the edge.

The edges of E_i associated with a component $j > 3$ are a subset of the edges of the graph induced in P by the vertices of the component. Thus, for $j > 3$, the number of edges associated with the component is at most $2(p_j - 1)$, because the induced graph is a triangle-free planar graph. (For all $l \ge 2$, the number of edges in a triangle-free planar graph with l vertices is at most $2(l-1)$; the number of edges is at most $2l - 4$ if $l \ge 3$ [B, Theorem 12, p. 18].

The number of edges in E_i associated with supercomponent 123 (which has at least three vertices) is bounded by $[2(p_1 + p_2 + p_3) - 4] - 2$. The "$-2$" appears because the edges xz and yz are not in E_i since $i \le m-2$. Indeed, the graph induced in P by the vertices of supercomponent 123 is planar, triangle-free, and includes the edges xz and yz. It follows that the number of edges in E_i associated with supercomponent 123 is at most $2(p_1 + p_2 + p_3) - 6 = 2[(p_1 - 1) + (p_2 - 1) + (p_3 - 1)]$. Therefore, $|E_i| \le \sum_{j=1}^{3} 2(p_j - 1) + \sum_{j=4}^{k} 2(p_j - 1) = 2|F_i|$. □

We continue with a straight-line plane subgraph P of a weighted graph G. We say a triangle xyz is *inside triangle* xyw (or xyw *contains* xyz) if z is embedded in the bounded plane face defined by xyw. If xyz and xyw are two triangles with z and w on the same side of the line containing xy, $w \ne z$, then one of the triangles has to be inside the other.

We will use the following topological fact, given without proof.

FACT 18. *Let xsz be a triangle containing a triangle xyz (with $s \ne y$). Then there is no triangle xyw (with $w \ne z$) which contains xyz.*

For a triangle Δ, we denote the median weight of the edges of the triangle by $b(\Delta)$, as above.

DEFINITION 5. In a straight-line plane graph, a *separating triangle* is a triangle $\Delta = xyz$ such that some neighbor of x, y, or z is embedded inside Δ, and some neighbor of x, y, or z is embedded outside Δ. (This is slightly different from the standard definition of a separating triangle.)

We now prove the key result needed for the lower bound.

LEMMA 19. *Let P be a straight-line weighted plane graph. There is a spanning forest $\bar{T}$ in P and a good family $\mathcal{S}$ such that*

$$w(\bar{T}) + \tfrac{1}{2} \sum_{\Delta \in \mathcal{S}} b(\Delta) \geq \tfrac{1}{2} w(P).$$

PROOF. By induction on the number of separating triangles in P.

If P has no separating triangle, we choose $\mathcal{S}$ to be the set of all triangles in P. Now $\mathcal{S}$ is good, for if it were not, choose an edge xy with three triangles xya, xyb, and xyc of $\mathcal{S}$ on the same side of the line containing xy. Relabel a, b, and c, if necessary, so that a is inside $\Delta = xyb$ and c is outside. Then a is a neighbor of x embedded inside Δ and c is a neighbor of x embedded outside Δ, so that Δ is a separating triangle, a contradiction.

We set $\beta = \sum_{\Delta \in \mathcal{S}} b(\Delta)$. Removing the median weight edge in each triangle of P leaves us with a triangle-free graph, whose weight is at least $w(P) - \beta$. Let $\bar{T}$ be a maximum spanning forest in the resulting triangle-free graph. Then

$$w(\bar{T}) \geq \tfrac{1}{2}(w(P) - \beta),$$

by a result similar to Corollary 7 for triangle-free planar graphs, using the same upper bound on the number of edges in triangle-free simple planar graphs as in the proof of Lemma 17.

For the inductive step, assume that P has a separating triangle. Choose a separating triangle, say xyz, of minimum area; there is no separating triangle inside it. Let P_{in} be the subgraph of P induced by the vertices x, y, z and the vertices inside the triangle xyz. P_{in} has no separating triangle. Each edge of xyz is in at most one triangle in P_{in} other than xyz. Consider the weight function w_{in} on the edges of P_{in} defined as follows: $w_{\text{in}}(e) = w(e)$ for all $e \in P_{\text{in}}$, except that $w_{\text{in}}(xy) = w_{\text{in}}(yz) = w_{\text{in}}(zx) = 0$. Let P_{out} be the subgraph of P induced by the vertices x, y, z and the vertices outside the triangle xyz. Now

$$w(P) = w(P_{\text{out}}) + w_{\text{in}}(P_{\text{in}}).$$

Our goal is to find a spanning forest F in P_{in} which does not connect any two of x, y, and z, and a spanning forest in P_{out}. Together, they will form the desired spanning forest $\bar{T}$ in P.

Since every separating triangle in P_{out} is also a separating triangle in P, and since the separating triangle xyz of P is not separating in P_{out}, P_{out} has fewer separating triangles

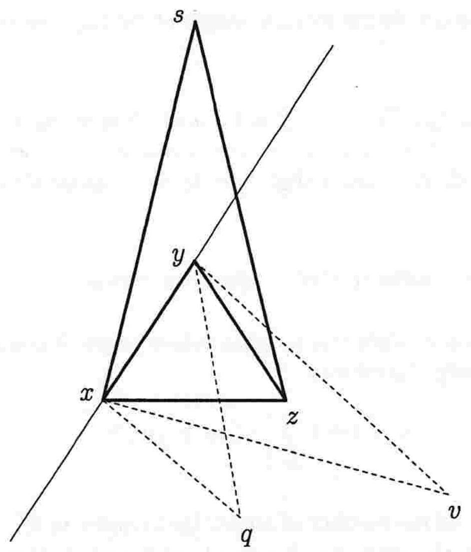


Fig. 1. xyz is a triangular face (nothing embedded inside) of the straight-line plane graph P_{out}. If y is in the interior of triangle xzs, z cannot be in the interior of some triangle xyv of P_{out}. Also, a triangle like xyq cannot exist. In other words, on the side containing z of the line containing edge xy, there can be only one triangle xyw for some w, namely, xyz.

than P; by induction, we have a spanning forest T_{out} in P_{out} and a good family $\mathcal{S}_{\text{out}}$ of triangles of P_{out} such that

$$w(T_{\text{out}}) + \tfrac{1}{2} \sum_{\Delta \in \mathcal{S}_{\text{out}}} b(\Delta) \geq \tfrac{1}{2} w(P_{\text{out}}).$$

Since P_{in} has no separating triangles, we know, where $\mathcal{S}'_{\text{in}}$ is the set of *all* triangles of P_{in}, that $\mathcal{S}'_{\text{in}}$ is good for P_{in}. Unfortunately, however, we cannot guarantee that $\mathcal{S}'_{\text{in}} \cup \mathcal{S}_{\text{out}}$ is a good family for P. We will construct a slightly different good $\mathcal{S}$ for P.

In the following we assume that, in P_{out}, on the side containing y of the line containing edge xz, there is only one triangle of the form xzw, namely, xzy, and on the side containing x of the line containing edge yz, there is only one triangle of the form yzw, namely, yzx.

We now justify this assumption. If the assumption is not true, then, in P_{out}, either (1) on the side containing y of the line containing edge xz, there are at least two triangles containing edge xz, or (2) on the side containing x of the line containing edge yz, there are at least two triangles containing yz. The two cases are symmetric, for we can interchange x and y. So we only argue about the former, in which, in P_{out}, on the side containing y of the line containing edge xz, there are at least two triangles. One of them is triangle xyz and the other is triangle xzs (containing y in its interior). For intuition, we refer to Figure 1. However, in this case Fact 18 ensures that in P_{out}, on the side containing z of the line containing edge xy, there is only one triangle of the form xyw for some w, namely, the facial triangle xyz of P_{out}. (There is no triangle such as xyv or xyq in Figure 1.) Similarly, on the side containing x of the line containing edge yz, the only triangle wyz for some

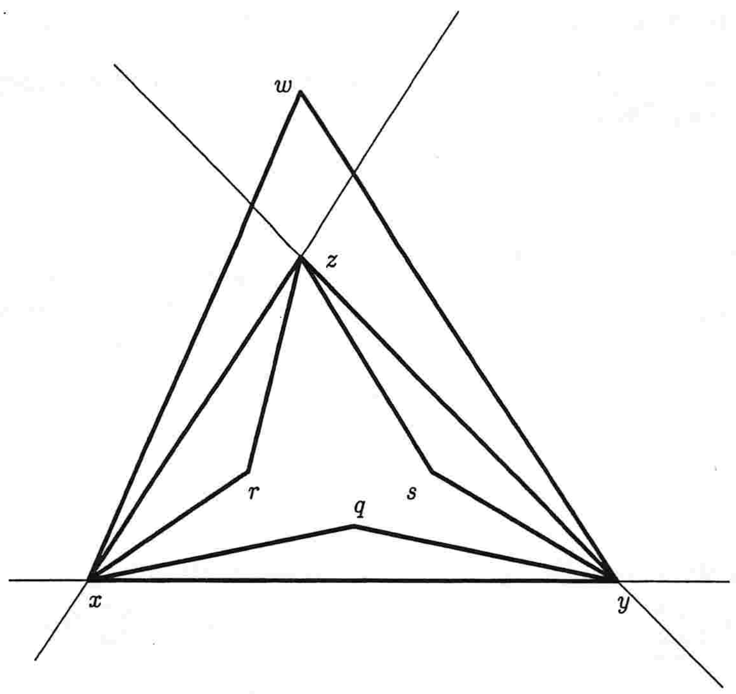


Fig. 2. Since the triangles xyz and xyq have been excluded from $\mathcal{S}_{\text{in}}$, on the side containing z of the line containing x and y there are at most two triangles in $\mathcal{S}$, those from $\mathcal{S}_{\text{out}}$ (one of them being xyw in this picture). On the side containing x of the line containing yz, there can be only the triangles syz (from $\mathcal{S}_{\text{in}}$) and xyz (which might be in $\mathcal{S}_{\text{out}}$). No other triangle of this type can exist in $\mathcal{S}_{\text{out}}$ because of the assumption.

w is the facial triangle xyz of P_{out}. Since x, y, and z are symmetric, we can satisfy the assumption by interchanging y and z.

If possible, let q, r, and s be vertices, not necessarily distinct, (strictly) inside the triangle xyz, such that xyq, xzr, and yzs are facial triangles. Let $\mathcal{S}_{\text{in}}$ be the set of triangles of P_{in}, excluding xyz and xyq (if xyq exists). Clearly, $\mathcal{S}_{\text{in}}$ is good for P_{in}. We will show that $\mathcal{S} = \mathcal{S}_{\text{in}} \cup \mathcal{S}_{\text{out}}$ is a good family for P. For intuition, we refer to Figure 2. Indeed, (1) any edge $e \neq xy, yz, xz$ participates in at most two triangles of $\mathcal{S}$ on each side of the line containing e (because $\mathcal{S}_{\text{in}}$ and $\mathcal{S}_{\text{out}}$ are good and the only edges shared by both are xy, yz, and xz). (2) On the side containing z of the line containing xy, in $\mathcal{S}_{\text{in}}$ there is no triangle containing x and y. This is true since in P_{in} there is no separating triangle, and the triangles xyz and xyq have been excluded from $\mathcal{S}_{\text{in}}$. The other side clearly contains no triangle of $\mathcal{S}_{\text{in}}$. In $\mathcal{S}_{\text{out}}$, on each of the corresponding sides of the line containing xy, there are at most two triangles containing x and y. (3) On the side containing x of the line containing the edge yz, there is at most one triangle of $\mathcal{S}_{\text{in}}$ containing z and y, namely, syz, since xyz has been excluded from $\mathcal{S}_{\text{in}}$. By assumption, in P_{out}, on the side containing x of the edge yz, there is only one triangle containing z and y, namely, xyz. On the side not containing x of the line containing the edge yz, there is no triangle of $\mathcal{S}_{\text{in}}$ and there are at most two of $\mathcal{S}_{\text{out}}$. (4) For the edge xz, the reasoning is similar to that for yz. This completes the proof that $\mathcal{S} = \mathcal{S}_{\text{in}} \cup \mathcal{S}_{\text{out}}$ is good.

Let $\beta_{\text{in}} = \sum_{\Delta \in \mathcal{S}_{\text{in}}} b_{\text{in}}(\Delta)$, where b_{in}, the value of the median edge, is computed according to w_{in}. Then, using the fact that $w_{\text{in}} \le w$,

$$\sum_{\Delta \in \mathcal{S}_{\text{in}}} b(\Delta) \ge \beta_{\text{in}}.$$

If we remove the median edge (according to w_{in}) from each triangle in P_{in}, never choosing xz or yz as a median-weight edge (so we remove xy from P_{in}), we get a triangle-free graph P'_{in} still having edges xz and yz of weight $w_{\text{in}} = 0$. By Lemma 17, there is a spanning forest F_{in} in P'_{in} not connecting any two of x, y, and z and having weight

$$w(F_{\text{in}}) = w_{\text{in}}(F_{\text{in}}) \ge \tfrac{1}{2} w_{\text{in}}(P'_{\text{in}}) \ge \tfrac{1}{2}(w_{\text{in}}(P_{\text{in}}) - \beta_{\text{in}}).$$

Then

$$w(F_{\text{in}}) + \tfrac{1}{2} \sum_{\Delta \in \mathcal{S}_{\text{in}}} b(\Delta) \ge \tfrac{1}{2} w_{\text{in}}(P_{\text{in}}).$$

Putting together T_{out} and F_{in} to get a forest $\bar{T}$ in P, we conclude with

$$\begin{aligned} w(\bar{T}) + \tfrac{1}{2} \sum_{\Delta \in \mathcal{S}} b(\Delta) &= [w(F_{\text{in}}) + w(T_{\text{out}})] + \tfrac{1}{2} \sum_{\Delta \in \mathcal{S}_{\text{in}}} b(\Delta) + \tfrac{1}{2} \sum_{\Delta \in \mathcal{S}_{\text{out}}} b(\Delta) \\ &\ge \left[w(F_{\text{in}}) + \tfrac{1}{2} \sum_{\Delta \in \mathcal{S}_{\text{in}}} b(\Delta) \right] + \left[w(T_{\text{out}}) + \tfrac{1}{2} \sum_{\Delta \in \mathcal{S}_{\text{out}}} b(\Delta) \right] \\ &\ge \tfrac{1}{2}(w_{\text{in}}(P_{\text{in}}) + w(P_{\text{out}})) \\ &= \tfrac{1}{2} w(P). \end{aligned}$$

□

COROLLARY 20. *There exists a good family $\mathcal{S}$ in P such that*

$$w(T_0) \ge \tfrac{1}{2} w(P) - \tfrac{1}{2} \left[\sum_{\Delta \in \mathcal{S}} b(\Delta) \right].$$

PROOF. Choose $\mathcal{S}$, $\bar{T}$ as in Lemma 19. Since $w(T_0) \ge w(\bar{T})$, we have $w(T_0) \ge w(\bar{T}) \ge \frac{1}{2} w(P) - \frac{1}{2}[\sum_{\Delta \in \mathcal{S}} b(\Delta)]$. □

3.7. *A Bound on the Performance Ratio in the General Case*

THEOREM 21. *The performance ratio of the algorithm is at least $1/3 + 1/72$.*

PROOF. Let P be a straight-line embedding of a maximum weight planar subgraph of G and let F be a maximum spanning forest in P. We know from Corollaries 2 and 4 that the algorithm produces a triangular structure S_0 of weight

$$w(S_0) \ge \tfrac{1}{2}[w(T_0) + w(T_f)].$$

Now $w(T_0) \geq w(F) \geq \frac{1}{3}w(P)$ (Corollary 7). By Corollary 20, there is a good family $\mathcal{S}$ in P such that $w(T_0) \geq \frac{1}{2}w(P) - \frac{1}{2}\beta$, where $\beta = \sum_{\Delta \in \mathcal{S}} b(\Delta)$. Therefore

$$w(T_0) \geq \max\{\tfrac{1}{3}w(P), \tfrac{1}{2}w(P) - \tfrac{1}{2}\beta\}.$$

(It is important that we are not using only the trivial first bound here.) Corollary 16 implies the bound

$$w(T_f) \geq \tfrac{1}{3}w(P) + \tfrac{1}{12}\beta$$

for the same $\mathcal{S}$. We therefore have

$$w(S_0) \geq \tfrac{1}{2}[\max\{\tfrac{1}{3}w(P), \tfrac{1}{2}w(P) - \tfrac{1}{2}\beta\} + (\tfrac{1}{3}w(P) + \tfrac{1}{12}\beta)].$$

A simple balancing argument shows that this lower bound is minimized when $\beta = \frac{1}{3}w(P)$, so that we obtain

$$w(S_0) \geq (\tfrac{1}{3} + \tfrac{1}{72})w(P). \qquad \square$$

3.8. *An Upper Bound on the Performance Ratio of the Algorithm.* On the other hand, the performance ratio of the algorithm is at most 5/12. In fact, we will exhibit *planar* graphs for which the weight of a maximum weight triangular structure is at most $\frac{5}{12}+o(1)$ of the total weight. Running the algorithm on any of these planar graphs will result in a triangular structure and therefore the weight of the planar subgraph found by our algorithm is at most $\frac{5}{12} + o(1)$ of the optimal weight. These planar graphs are obtained from arbitrary triangulated plane graphs through a construction we describe below.

Let P be any triangulated (simple) plane graph with $n \geq 3$ vertices. By Euler's formula, P has $3n - 6$ edges; let the weight of each edge of P be two. In each of the $2n - 4$ faces of P add a new vertex, and add three edges adjacent to this new vertex and the three vertices of P which define the face. These new edges have weight one. We denote by G the plane graph obtained this way. G has $3n - 4$ vertices, is triangulated, and has total weight $2 \cdot [3n - 6] + 1 \cdot [3(2n - 4)] = 12n - 24$. (See Figure 3.) We will prove that a maximum weight triangular structure in G has weight at most $\frac{5}{12} + o(1)$ of the weight of G. To do so, we need a lemma (which will also be used later).

LEMMA 22. *Let $G = (V, E)$ be any graph and let T be a set (possibly intersecting E) of edges on V. Let w be a nonnegative weight function on $E \cup T$. Assume that T is a maximum spanning tree in the multigraph $G \cup T = (V, E \cup T)$. For every triangle $\Delta = xyz$ in G, let*

$$\begin{aligned} g(\Delta) &= w(\Delta) - \max\{index_T(x, y), index_T(x, z), index_T(y, z)\} \\ &\quad - \min\{index_T(x, y), index_T(x, z), index_T(y, z)\}. \end{aligned}$$

If $g(\Delta) \leq 0$ for every triangle Δ in G, then $w(T) \geq w(A)$ for every triangular structure A in G.

PROOF. Let A be an arbitrary triangular structure in G. We iteratively define edge set E_i, graph $G_i = (V, E_i)$, and triangular structure A_i in G_i. To start, let $E_0 = E$, $G_0 = G$, $A_0 = A$, and $i = 0$.

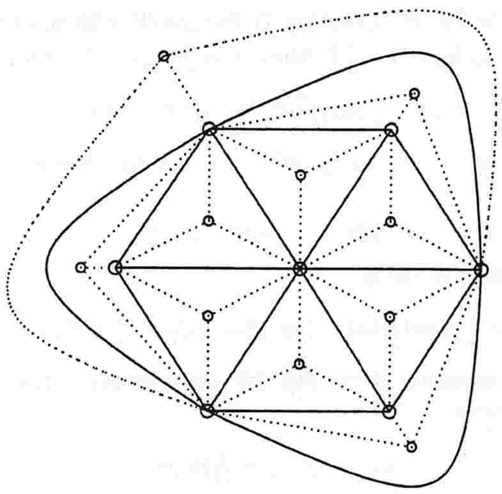


Fig. 3. An example of graph G. The solid edges show graph P. Each of them has weight two. The dotted part shows the extra vertices and edges in G. The dotted edges have weight one.

Let Δ be a triangle in A_i. Label the vertices of Δ as x, y, z so that $\mathit{index}_T(x, y) \leq \mathit{index}_T(x, z) \leq \mathit{index}_T(y, z)$. We modify A_i by removing the edges of Δ and adding two new edges e_1 and e_2 to get A_{i+1}. The endpoints of e_1 are x and y and $w(e_1) = \mathit{index}_T(x, y)$. The endpoints of e_2 are y and z and $w(e_2) = \mathit{index}_T(y, z)$. We set $E_{i+1} = E_i \cup \{e_1, e_2\}$ and $G_{i+1} = (V, E_{i+1})$.

Since $g(\Delta) \leq 0$, $w(\Delta) \leq \mathit{index}_T(x, y) + \mathit{index}_T(y, z) = w(e_1) + w(e_2)$. So $w(A_{i+1}) \geq w(A_i)$. Also, A_{i+1} is a triangular structure in G_{i+1} all of whose triangles are in G, having fewer triangles than A_i.

We iterate this process for all q triangles of A. At the end we have a triangular structure A_q in a graph G_q, with $w(A_q) \geq w(A)$. A_q has no triangles so it is a forest in G_q. We know that T is a maximum spanning tree in $G \cup T$. Each additional edge (with endpoints, say, u and v) we add has weight at most (in fact, exactly) $\mathit{index}_T(u, v)$. This guarantees that the one tree T is a maximum spanning tree in each $G_i \cup T$ and in particular in $G_q \cup T$. Since A_q is a forest in G_q, $w(T) \geq w(A_q)$. So we conclude that $w(T) \geq w(A_q) \geq w(A)$. □

Now we are ready to prove that a maximum weight triangular structure in G has weight at most $\frac{5}{12} + o(1)$ of the weight of G. Let A be a set of new weight-three edges forming any spanning tree on $V(P)$, the vertex set of P.

Let B be a subset of the edges of G: for each of the $2n-4$ vertices inserted in the faces of P, choose exactly one edge of G incident to it. Then we know that $T = (V(G), A \cup B)$ is a spanning tree of the graph $G \cup T$.

In fact, it is easy to verify that T is a maximum spanning tree in $G \cup T$. Consider an edge e of G. If e has weight one, the path in T between e's endpoints obviously cannot contain an edge of weight smaller than one. If e has weight two, then e is an edge of P. Let u and v be the endpoints of e; then $u, v \in V(P)$. The set of edges A is a spanning tree on $V(P)$. As there is a path in A from u to v, the unique path in T from u to v is contained in A and therefore the weight of any edge on the unique path in T from u to v is three.

Also, no triangle of G has positive gain with respect to T, for the following reason. Consider a triangle Δ in G. Label the vertices of Δ as x, y, z so that $index_T(x, y) \le index_T(x, z) \le index_T(y, z)$. If all the edges in Δ have weight two, then Δ is a triangle in P and the path in T between any pair of its vertices contains only edges of weight three. Thus we have $w(\Delta) = 6$, $index_T(x, y) = 3 = index_T(y, z)$. This makes $g(\Delta) = w(\Delta) - index_T(x, y) - index_T(y, z) = 0$. If Δ contains at least one edge of weight one, then it contains exactly two edges of weight one and one edge of weight two. This means $w(\Delta) = 4$. Since $index_T(x, y) \le index_T(x, z) \le index_T(y, z)$, we have $index_T(x, y) = 1$ and $index_T(y, z) = 3$. Thus again $g(\Delta) = w(\Delta) - index_T(x, y) - index_T(y, z) = 0$.

Therefore, we can apply Lemma 22 and conclude that $w(T) \ge w(S)$ for any triangular structure S in G. In conclusion, if S is a triangular structure in G,

$$w(S) \le w(T) = w(A) + w(B) = 3(n-1) + 1(2n-4) = 5n - 7 = \left(\frac{5}{12} + \frac{1}{4n-8}\right) w(G).$$

In fact, we conjecture that $w(S_{\text{opt}}) \ge \frac{5}{12} w(P)$, where S_{opt} is a heaviest triangular structure in planar graph P, for any weight function.

3.9. *The Triangle Inequality.* The next theorem gives a better performance ratio for the algorithm if the input graph is a weighted clique whose edge weights satisfy the triangle inequality.

THEOREM 23. *Suppose that the edge weights in a weighted clique G satisfy the triangle inequality. Then $w(S_0) \ge \frac{3}{8} w(P)$, where P is a heaviest planar subgraph of G.*

PROOF. Let P be a straight-line embedding of a heaviest planar subgraph of G. We assume, without loss of generality, that P is triangulated. By the triangle inequality, we have $b(\Delta) \ge \frac{1}{2} a(\Delta)$ for any (facial) triangle Δ of P. Recall (from Section 3.5, p. 190) that $w'(xy) = index_f(x, y)$ for all edges $xy \in E(P)$. Using Lemma 13, it follows that for a triangle Δ of P we have

$$w'(\Delta) = a'(\Delta) + b'(\Delta) + c'(\Delta) \ge a(\Delta) + b(\Delta) + c(\Delta) + b(\Delta).$$

Using $b(\Delta) \ge c(\Delta)$ and $a(\Delta) \le b(\Delta) + c(\Delta)$, we can deduce that

$$b(\Delta) \ge \tfrac{1}{3}[a(\Delta) + c(\Delta)]$$

and hence

$$w'(\Delta) \ge \tfrac{5}{4}[a(\Delta) + b(\Delta) + c(\Delta)].$$

Therefore

$$w'(\Delta) \ge \tfrac{5}{4} w(\Delta).$$

Then, because P is triangulated,

$$w'(P) = \tfrac{1}{2} \sum_{\text{facial triangles } \Delta \text{ of P}} w'(\Delta) \geq \tfrac{1}{2} \sum_{\text{facial triangles } \Delta \text{ of P}} \tfrac{5}{4} w(\Delta) = \tfrac{5}{4} w(P).$$

This bound and $w(T_f) \geq \frac{1}{3} w'(P)$ (Lemma 14) imply that $w(T_f) \geq \frac{5}{12} w(P)$ and hence, using Corollaries 4 and 7,

$$w(S_0) \geq \tfrac{1}{2}(w(T_0) + w(T_f)) \geq \tfrac{1}{2}(\tfrac{1}{3} + \tfrac{5}{12}) w(P) = \tfrac{3}{8} w(P). \qquad \square$$

3.10. *Outerplanar Subgraphs.* Consider the MAXIMUM WEIGHT OUTERPLANAR SUBGRAPH problem: given an edge-weighted graph G, find an outerplanar subgraph of G of maximum weight. This problem is known to be NP-hard [GJ, p. 197].

We know of no published research on the problem. The naive approximation ratio is $1/2$, which is obtained by any algorithm that produces a maximum spanning tree T_0 of G (assuming G is connected). To prove this, we need the following lemma.

LEMMA 24. *Let P be an outerplanar graph with nonnegative weight function w on the edges. Then a maximum spanning forest F of P satisfies $w(F) \geq \frac{1}{2} w(P)$.*

PROOF. Any subgraph of an outerplanar graph is outerplanar, and, by Euler's formula, $|E(H)| \leq 2(|V(H)| - 1)$ for any outerplanar graph H [H1, Corollary 11.9]. Thus, it is enough to apply Lemma 6 with $c = 2$ to the family of all outerplanar graphs. $\square$

Let P be any outerplanar subgraph of G. By Lemma 24, we deduce that $w(T_0) \geq \frac{1}{2} w(P)$, and the performance ratio of $1/2$ follows.

A triangular structure is an outerplanar graph. In the unweighted case, a maximum triangular structure in a graph G (one with the maximum number of edges) has at least two-thirds as many edges as a maximum outerplanar subgraph of G [CFFK]. We will show that a maximum *weight* triangular structure in a weighted graph G has weight at least two-thirds of the weight of a maximum weight outerplanar subgraph of G. Unfortunately, we do not know how, in polynomial time, to find a maximum weight triangular structure in a given weighted graph. The algorithm presented in this chapter produces a heavy triangular structure in a given weighted graph. We will prove that the algorithm improves the best known performance ratio for MAXIMUM WEIGHT OUTERPLANAR SUBGRAPH to $7/12 > 0.583$.

The proof of the next key lemma takes a few pages.

LEMMA 25. *In any maximal outerplanar graph P, there are at most three (not necessarily distinct) triangular structures in P such that each edge of P appears in exactly two of them.*

PROOF. If P has fewer than three vertices, then P and the empty graph are triangular structures. Assume P has at least three vertices. Embed P in the plane as a triangulation of a polygon. (Every maximal outerplanar graph with at least three vertices is a triangulation of a polygon. That is, the boundary of the exterior face of P is a Hamiltonian cycle H

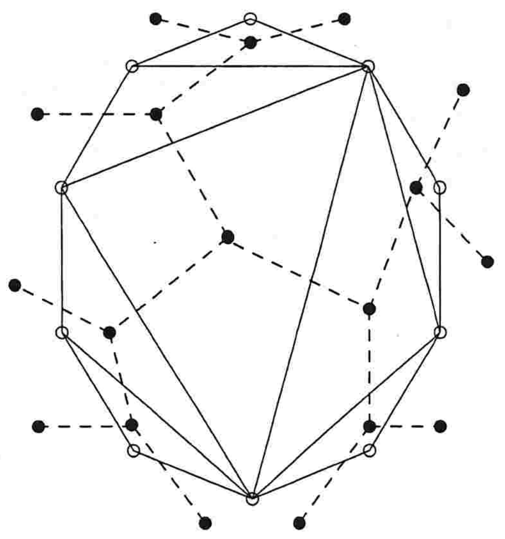


Fig. 4. An outerplanar graph (solid lines) and the tree D' (filled vertices and dotted lines) obtained from its dual.

and each interior face is triangular [H1, p. 106].) Let E be the edge set of P, let b be the exterior face of P, and let F be the set of faces of P other than b.

Let D be the dual multigraph of P. We call the vertices of D (which are faces of P) *nodes*, and the edges of D, *arcs*. All nodes of D but b have degree three. Also, the edges in the Hamiltonian cycle H correspond to the arcs incident to b in D.

Let D' be the graph obtained from D by subdividing each arc incident to b, and then removing b. See Figure 4 for an example.

LEMMA 26. *D' is a tree all of whose internal nodes have degree three.*

PROOF. First, we prove that D' has no cycle. It is enough to show that any cycle in D contains b. A cycle in D corresponds to a cut in P [BM, p. 143, Exercise 9.2.3]. Because H is a Hamiltonian cycle, any cut in P contains at least two edges of H, which correspond to arcs incident to b. Therefore, any cycle in D contains at least two arcs incident to b, so it contains b.

Second, we prove that D' is connected. If D' were not connected, there would be two nodes u and v in different components of D'. We argue that we can choose u and v to be nodes in $V(D)$. If u were not a node in $V(D)$, then it would be a node which originated from the subdivision of an arc incident to b, and thus it would have degree one in D'. Change u to its unique neighbor in D'. Do the same for v. Note that the new u and v must still be in different components of D', since they are in the same component as the initial u and v, respectively. So we can assume u and v are nodes of D. Because D is connected, there is a path in D between u and v. For this path not to exist in D', it has to go through node b. However, this implies that b would be a cut vertex in D. If b were a cut vertex in D, then there would be a *minimal* cut in D containing exactly a proper subset of the arcs incident to b (consider the set of edges going from b to one of the components of D after the removal of b). A minimal cut in D corresponds to a cycle in P [BM, p. 143, Exercise 9.2.3]. This implies that there would be a cycle in P whose edges are a proper subset of the edges of H, a contradiction (a proper subset of the edges of any cycle induces an acyclic graph, since it is enough to remove one edge of a cycle to be left with a path, which is acyclic).

Therefore D' is, in fact, a tree. Recall that all nodes of D but b have degree three. Before removing b, we subdivided all of the arcs incident to b. The new nodes have degree one in D', and all others have the same degree as in D, i.e., three. □

A *special* 3-*coloring of* D' is a partition of the set of nodes of D' into three sets (X_1, X_2, X_3) (each set referred to as a *color class*) such that

1. adjacent nodes have different colors, and
2. for $i = 1, 2, 3$, if we remove all nodes of color i, in the resulting graph there is a path from any node to a leaf in D'.

LEMMA 27. *There is a special* 3-*coloring of* D'.

PROOF. Root D' at one of its leaves. In the rooted D', all internal nodes except the root have two children. Color the root and its unique child with distinct colors. Now, start at level $i = 1$. If there are nodes in level $i + 1$, for each node in level i with children, give its children distinct colors which differ from its own color. Proceed to level $i + 1$.

Clearly, adjacent nodes get distinct colors.

Suppose we remove all nodes of color i, for some i; let j and k be the two remaining colors. Consider a remaining node. Either it is a leaf in D', and there is nothing to prove, or it is an internal node of the rooted tree D' different from the root (since the root is a leaf of D'). Any internal node colored j which is not the root has a child of color k, and vice versa. This means that there is a path from any node to a leaf in D'. □

Given a special 3-coloring (X_1, X_2, X_3) of D', we now describe three triangular structures S_1, S_2, and S_3 as required by Lemma 25. Establish the natural one-to-one correspondence between edges of P and arcs of D': each edge of P corresponds to the unique arc of D' which "crosses" it.

Let S_i be the set of edges in P whose corresponding arc in D' has an endpoint of color i (i.e., in X_i). (An arc has either one or zero endpoints of a given color.)

LEMMA 28. *S_i is a triangular structure in P.*

PROOF. Suppose that there is a cycle C in S_i which is not a triangle. C partitions the set of faces of P into two sets $F_0 \ni b$ (outside C) and $F_1 \subseteq F$ (inside C), F being the set of faces of D other than b.

Because C is not a triangle, $|F_1| \geq 2$. (If $|F_1| = 1$, then C would be the boundary of the unique face in $F_1 \subseteq F$, which is a triangle.)

A cycle in P corresponds to a minimal cut in D [BM, p. 143, Exercise 9.2.3]. So F_1 must induce a connected subgraph of D. Since D' differs from D only in arcs incident to b, F_1 induces the same connected subgraph in D'. Also, F_1 consists only of internal nodes of D'.

Thus F_1 induces a connected subgraph of D' with at least two nodes. Hence, in the given special 3-coloring of D', not all nodes of F_1 can be of color i; there is a node $d \in F_1$ of color $j \neq i$.

All leaves of D' are outside C. Since C is in S_i, each edge of C corresponds to an arc in D' with one endpoint of color i. Removing the nodes of D' of color i hence eliminates all arcs with one endpoint inside C and one outside. Hence, after removing from D' the nodes (faces of P) of color i, there can be no path from node d to a leaf of D'. This is a contradiction to the fact that we have a special 3-coloring. □

Clearly, S_1, S_2, and S_3 satisfy the statement of Lemma 25: each edge of P appears in exactly two of them. □

THEOREM 29. *Let G be a graph with a nonnegative weight function w on the edges and let S_{opt} be a maximum weight triangular structure in G. Then $w(S_{\text{opt}}) \geq \frac{2}{3}w(P)$ for any outerplanar subgraph P of G.*

PROOF. By adding edges possibly not in G, extend P to a maximal outerplanar graph $\bar{P}$. For any edge e in $\bar{P}$ but not in G, let $w(e) = 0$. Clearly, $w(\bar{P}) \geq w(P)$.

Let S_1, S_2, S_3 be three triangular structures in $\bar{P}$ as given by Lemma 25. Each edge of $\bar{P}$ appears in exactly two of these triangular structures.

Then $w(S_1) + w(S_2) + w(S_3) = 2w(\bar{P}) \geq 2w(P)$. Moreover, $w(S_{\text{opt}}) \geq w(S_i)$, $i = 1, 2, 3$. Therefore $2w(P) \leq 3w(S_{\text{opt}})$, implying that $w(S_{\text{opt}}) \geq \frac{2}{3}w(P)$. □

COROLLARY 30. *The algorithm described in Section 2 has performance ratio at least 7/12 for* MAXIMUM WEIGHT OUTERPLANAR SUBGRAPH.

PROOF. Lemma 9 tells us that T_f is a maximum spanning tree in G_f; because G is a subgraph of G_f, T_f is a maximum spanning tree in $G \cup T_f$. By Lemma 12 we know that every triangle of G has nonpositive gain with respect to T_f. Thus, by Lemma 22, we infer that $w(T_f) \geq w(S_{\text{opt}})$, where S_{opt} is a maximum weight triangular structure in G.

Together with Corollary 4, this means that the algorithm produces a triangular structure S_0 such that

$$w(S_0) \geq \frac{w(T_0) + w(S_{\text{opt}})}{2},$$

where T_0 is a maximum spanning tree of G. So far, everything we have said applies to MAXIMUM WEIGHT PLANAR SUBGRAPH.

Let P be any maximum weight outerplanar subgraph of G. We know, by Lemma 24, that $w(T_0) \geq \frac{1}{2}w(P)$. From Theorem 29, we have $w(S_{\text{opt}}) \geq \frac{2}{3}w(P)$. Hence

$$w(S_0) \geq \frac{w(T_0) + w(S_{\text{opt}})}{2} \geq \frac{\frac{1}{2}w(P) + \frac{2}{3}w(P)}{2} = \frac{7}{12}w(P). \qquad \square$$

Acknowledgments. We thank Leonard Schulman and Robin Thomas for helpful discussions and suggestions. We are also very grateful to the referees for their very thorough readings and for their comments.

References

[AMO] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows*, Prentice-Hall, Upper Saddle River, NJ, 1993.

[B] B. Bollobas, *Graph Theory—An Introductory Course*, Springer-Verlag, New York, 1979.

[BETT] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis, Algorithms for Drawing Graphs: an Annotated Bibliography, *Computational Geometry: Theory and Applications* **4**(5) (1994), 235–282.

[BFP] R.P.G. Barlow, G.N. Fisher, and S.N. Papantonopoulos, The Application of a Knowledge-Based Engineering System to the Planning of Airport Facilities—a Case Study, in *Proc. IEE Colloquium on Knowledge-Based Approaches to Automation in Construction*, 1995, pp. 1–4.

[BM] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, MacMillan, New York, 1976.

[BNTT] C. Batini, E. Nardelli, M. Talamo, and R. Tamassia, A Graph-Theoretic Approach to Aesthetic Layout of Information Systems Diagrams, in *Proc. 10th International Workshop on Graph-Theoretic Concepts in Computer Science*, 1984, pp. 9–18.

[BR] P. Berman and V. Ramaiyer, Improved Approximations for the Steiner Tree Problem, *Journal of Algorithms* **17** (1994), 381–408.

[C] R. Cimikowski, An Analysis of Heuristics for Graph Planarization, *Journal of Information & Optimization Sciences* **18**(1) (1997), 49–73.

[CFFK] G. Călinescu, C.G. Fernandes, U. Finkler, and H. Karloff, A Better Approximation Algorithm for Finding Planar Subgraphs, *Journal of Algorithms* **27** (1998), 269–302.

[CMW+] B. Chen, M. Matsumoto, J. Wang, Z. Zhang, and J. Zhang, A Short Proof of Nash-Williams' Theorem for the Arboricity of a Graph, *Graphs and Combinatorics* **10** (1994), 27–28.

[CNS] N. Chiba, T. Nishizeki, and I. Shirakawa, An Algorithm for Maximal Planarization of Graphs, in *Proc. IEEE International Symposium on Circuits and Systems*, 1979, pp. 649–652.

[CON] N. Chiba, K. Onogushi, and T. Nishizeki, Drawing Planar Graphs Nicely, *Acta Informatica* **22** (1985), 187–201.

[CST] P. Crescenzi, R. Silvestri, and L. Trevisan, To Weight or Not to Weight: Where is the Question?, in *Proc. 4th Israel Symposium on Theory of Computing and Systems*, 1996, pp. 68–77.

[CYN] N. Chiba, T. Yamanouchi, and T. Nishizeki, Linear Algorithms for Convex Drawings of Planar Graphs, in *Progress in Graph Theory*, J.A. Bondy and U.S.R. Murty (eds.), Academic Press, New York, 1984, pp. 153–173.

[DFF] M.E. Dyer, L.R. Foulds, and A.M. Frieze, Analysis of Heuristics for Finding a Maximum Weight Planar Subgraph, *European Journal of Operations Research* **20** (1985), 102–114.

[DH] J.W. Dickey and J. Hopkins, Campus Building Arrangements Using TOPAZ, *Transportation Research* **6** (1972), 59–68.

[E] A.N. Elshafei, Hospital Layout as a Quadratic Assignment Problem, *Operational Research Quarterly* **28** (1977), 167–179.

[EFG] P. Eades, L. Foulds, and J. Giffin, An Efficient Heuristic for Identifying a Maximal Weight Planar Subgraph, Lectures Notes in Mathematics 952, Springer-Verlag, Berlin, 1982, pp. 239–251.

[F] L.R. Foulds, *Graph Theory Applications*, Springer-Verlag, New York, 1992.

[FG] U. Feige and M.X. Goemans, Approximating the Value of Two-Prover Proof Systems, with Applications to MAX 2SAT and MAX DICUT, in *Proc. 3rd Israel Symposium on the Theory of Computing and Systems*, 1995, pp. 182–189.

[FGG] L.R. Foulds, P.B. Gibbons, and W.J. Giffin, Graph-Theoretic Heuristics for the Facilities Layout Problem: an Experimental Comparison, *Operations Research* **33** (1985), 1091–1106.

[FR] L.R. Foulds and D.F. Robinson, Graph-Theoretic Heuristics for the Plant Layout Problem, *International Journal of Production Research* **12** (1978), 27–37.

[FT] L.R. Foulds and H.V. Tran, Library Layout via Graph Theory, *Computers and Industrial Engineering* **10**(3) (1986), 245–252.

[GJ] M.R. Garey and D.S. Johnson, *Computers and Intractability*, Freeman, San Francisco, CA, 1979.

[GS] H.N. Gabow and M. Stallmann, Efficient Algorithms for Graphic Matroid Intersection and Parity, in *Automata, Language and Programming*: 12*th Colloquium*, Lecture Notes in Computer Science 194, Springer-Verlag, Berlin, 1985, pp. 210–220.

[GVY] N. Garg, V.V. Vazirani, and M. Yannakakis, Multiway Cuts in Directed and Node Weighted Graphs, manuscript. Preliminary version appeared in *Proc.* 21*st ICALP*, Lecture Notes in Computer Science 820, Springer-Verlag, Berlin, 1994, pp. 487–498.

[GW] M.X. Goemans and D.P. Williamson, Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming, *Journal of the ACM* **42**(6) (1995), 1115–1145.

[H1] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1972.

[H2] D. Hochbaum, Approximation Algorithms for the Set Covering and Vertex Covering Problems, *SIAM Journal of Computing* **11** (1982), 555–556.

[H3] D. Hochbaum, *Approximation Algorithms for NP-Hard Problems*, PWS, Boston, MA, 1997.

[HW] A. Hammouche and D.B. Webster, Evaluation of an Application of Graph Theory to the Layout Problem, *International Journal of Production Research* **23**(5) (1985), 987–1000.

[JM1] M. Jünger and P. Mutzel, Solving the Maximum Weight Planar Subgraph Problem, in *Proc.* 3*rd Integer Programming and Combinatorial Optimization Conference*, 1993, pp. 479–492.

[JM2] M. Jünger and P. Mutzel, Maximum Planar Subgraphs and Nice Embeddings: Practical Layout Tools, *Algorithmica* **16**(1) (1996), 33–59 (special issue on graph drawing).

[JTS] R. Jayakumar, K. Thulasiraman, and M.N.S. Swamy, On Maximal Planarization of Nonplanar Graphs, *IEEE Transactions on Circuits and Systems* **CAS-33** (1986), 843–854.

[K] K. Krejcirik, Computer Aided Plant Layout, *Computer Aided Design* **2** (1969), 7–17.

[KH] B. Korte and D. Hausmann, An Analysis of the Greedy Heuristic for Independence Systems, *Annals of Discrete Mathematics* **2** (1978), 65–74.

[L] A. Liebers, Planarizing Graphs—a Survey and Annotated Bibliography, *Journal of Graph Algorithms and Applications* **5**(1) (2001), 1–74.

[LG] P.C. Liu and R.C. Geldmacher, On the Deletion of Nonplanar Edges of a Graph, in *Proc.* 10*th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, 1977, pp. 727–738.

[LNS] R. Lipton, S. North, and J. Sandberg, A Method for Drawing Graphs, in *Proc. ACM Symposium on Computational Geometry*, 1985, pp. 153–160.

[M1] M. Marek-Sadowska, Planarization Algorithms for Integrated Circuits Engineering, in *Proc. IEEE International Symposium on Circuits and Systems*, 1978, pp. 919–923.

[M2] P. Mutzel, The Maximum Planar Subgraph Problem, Ph.D. Thesis, Universität zu Köln, 1994.

[N] C.St.J.A. Nash-Williams, Decomposition of Finite Graphs into Forests, *Journal of the London Mathematical Society* **39** (1964), 12.

[OT] T. Ozawa and H. Takahashi, A Graph-Planarization Algorithm and Its Applications to Random Graphs, Lectures Notes in Computer Science 108, Springer-Verlag, Berlin, 1981, pp. 95–107.

[S] D.B. Shmoys, Computing Near-Optimal Solutions to Combinatorial Optimization Problems, in *Combinatorial Optimization*, W. Cook and L. Lovász (eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science 20, AMS, Providence, RI, 1995, pp. 355–397.

[T] W.T. Tutte, How to Draw a Graph, *Proceedings of the London Mathematical Society* **13** (1963), 743–768.

[TBB] R. Tamassia, G. Di Battista, and C. Batini, Automatic Graph Drawing and Readability of Diagrams, *IEEE Transactions on Systems, Man and Cybernetics* **18** (1988), 61–79.

Primal-Dual Algorithms for QoS Multimedia Multicast

G. Calinescu* C.G. Fernandes† I.I. Mandoiu‡ A. Olshevsky§ K. Yang¶ A. Zelikovsky¶

Abstract— **The QoS Steiner Tree Problem asks for the most cost-efficient way to multicast multimedia to a heterogeneous collection of users with different consumption rates. We assume that the cost of using a link is not constant but rather depends on the maximum bandwidth routed through the link. Formally, given a graph with costs on the edges, a source node and a set of terminal nodes, each one with a bandwidth requirement, the goal is to find a Steiner tree containing the source, and the cheapest assignment of bandwidth to each of its edges so that each source-to-terminal path in the tree has bandwidth at least as large as the bandwidth required by the terminal. Our main contributions are: (1) new covering-type integer linear program formulations for the problem; (2) two new heuristics based on the primal-dual framework; (3) a primal-dual constant-factor approximation algorithm; (4) an extensive experimental study of the new heuristics and of several previously proposed algorithms.**

I. Introduction

Recent progress in audio, video, and data storage techniques has given rise to a host of high-bandwidth real-time applications such as video conferencing. Quality of Service (QoS) requirements for these applications must be provided by the underlying networks. In light of this, multicast routing algorithms which manage network resources efficiently and satisfy the QoS requirements have come under increased scrutiny in recent years. As applications such as video conferencing gain popularity, the focus on multimedia data transfer capability in networks will grow [12].

Multimedia distribution is usually done via multicast trees. There are two reasons for basing efficient multicast routes on trees: the data can be transmitted concurrently to destinations along the branches of the tree and only a minimum number of copies of the data is transmitted since information needs to be replicated only at forks in the tree [14]. The bandwidth savings obtained from the use of multicast trees can be maximized by using optimal or nearly optimal tree algorithms. Future networks will no doubt integrate such algorithms into basic operational performance [3].

Several versions of the QoS multicast problem have been studied in the literature. These versions seek routing tree cost minimization subject to (1) end-to-end delay, (2) delay variation, and/or (3) minimum bandwidth constraints (see, e.g., [3], [11], [7]). In this paper, we consider the case of minimum bandwidth constraints, that is, the problem of finding an optimal multicast tree when each terminal possesses a different rate of receiving information. This problem is a generalization of the classical Steiner tree problem and known to be NP-hard [4]. Formally, given a graph $G = (V, E)$, a source s, a set of terminals S, and two functions: $\mathit{length} : E \rightarrow \mathbb{R}^+$ representing the length of each edge and $\mathit{rate} : S \rightarrow \mathbb{R}^+$ representing the rate of each node, a *multicast tree* T is a tree in G spanning s and S. The *rate* $\mathit{rate}(e, T)$ of an edge e in a multicast tree T is the maximum rate of a terminal in the connected component of $T - e$ which does not contain s. The *cost* of a multicast tree T is defined as

$$\mathit{cost}(T) = \sum_{e \in T} \mathit{length}(e) \cdot \mathit{rate}(e).$$

Quality of Service Steiner Tree Problem (QoSST): Given a network $G = (V, E, \mathit{length}, \mathit{rate})$ with a source s in V and a set of terminals $S \subseteq V$, find a minimum cost multicast tree in G.

This formulation assumes that each link in the network has maximum possible rate, that is, has the maximum possible bandwidth. This is somewhat unrealistic due to different cables and traffic influence. A more sophisticated version of this problem would include a maximum possible rate function $\mathit{maxrate} : E \rightarrow \mathbb{R}^+$. Moreover, in practice the rates are rarely constant; another possible generalization would consider the case of dynamic rates.

The rest of the paper is organized as follows. In the next section we give a short summary of the algorithms proposed by [7], [4], and [6] and show that the approximation ratio of the algorithm from [7] is unbounded. In Section III, we consider an integer linear program formulation (ILP) and describe two heuristics based on the primal-dual framework. Then we prove that a primal-dual algorithm based on an enhanced ILP has an approximation ratio 4.311. Finally, in Section IV we conclude with an experimental comparison of our two primal-dual heuristics with algorithms from [7], [4].

*Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616. E-mail: calinesc@cs.iit.edu.

†Department of Computer Science, University of São Paulo, Rua do Matão, 1010, 05508-900 São Paulo, Brazil. E-mail: cris@ime.usp.br.

‡ Department of Computer Science & Engineering, UC San Diego, La Jolla, CA 92093. E-mail: mandoiu@cs.ucsd.edu.

§ Department of Electrical Engineering, Georgia Institute of Technology, Atlanta, GA 30332. E-mail: alex.olshevsky@resnet.gatech.edu.

¶ Department of Computer Science, Georgia State University, Atlanta, GA 30303. E-mail: {kenan,alexz}@cs.gsu.edu.

Input: A graph $G = (V, E, \mathit{length}, \mathit{rate})$ with a source s in V and a collection of terminals $S \subseteq V$.
Output: A QoS Steiner tree spanning the source and the terminals.

```
(1) Initialize the current tree to $\{s\}$.
(2) Find a non-reached terminal $t$ of highest rate with the shortest
    distance to the current tree.
(3) Add $t$ to the current tree along with a shortest path connecting it
    to the current tree.
(4) Repeat until all terminals are spanned.
```

Fig. 1. Maxemchuk's Algorithm for the QoS Steiner Tree Problem.

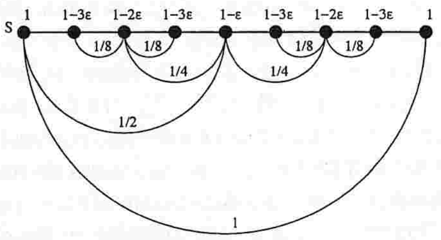


Fig. 2. A bad example for Maxemchuk's algorithm, with $k = 4$ rates. In the figure, $\varepsilon = 1/2^{2k-1}$. The rate of each node is given above the node. The edge lengths are given on the thin curved arcs, while on the solid horizontal line each segment has length $1/2^{k-1} + \varepsilon$. The optimum, of total cost $1 + 2^{k-1}\varepsilon = 1 + 2^{k-1}(1/2^{2k-1}) = 1 + 1/2^k$, uses the solid horizontal line at rate 1. Maxemchuk's algorithm picks the thin curved arcs at a cost of $1 + (1/2)(1-\varepsilon) + 2(1/4)(1-2\varepsilon) + 4(1/8)(1-3\varepsilon) \geq ((k+1)/2)(1 - 1/2^k)$.

II. Previous Work

A. *Maxemchuk's Approach*

Maxemchuk [7] proposed a heuristic algorithm for the QoS Steiner Tree Problem. His algorithm is a modification of the MST heuristic for Steiner Trees [13] (see Figure 1).

The extensive experiments given in [7] demonstrate that this method works well in practice. Nevertheless, the following example shows that the method may produce arbitrarily large error (linear in the number of rates) compared with the optimal tree. Consider the natural generalization of the example in Figure 2 with an arbitrary number k of distinct rates. Its optimal solution has a cost of about 1, whereas Maxemchuk's method returns a solution of cost about $(k+1)/2$. As there are $2^{k-1}+1$ nodes, this cost can also be written as $1 + \frac{1}{2}\log_2(n-1)$, where n is the number of nodes in the graph. We conclude that the approximation ratio of Maxemchuk's algorithm is at least linear in the number of rates and at least logarithmic in the number of nodes in the graph.

B. *The Charikar-Naor-Schieber Algorithm*

The Charikar-Naor-Schieber algorithm [4] is the first constant-factor approximation algorithm given for the QoS Steiner tree problem. In its first step, all rates are rounded to the closest power of two to produce the rounded up instance of this problem (clearly, this at most doubles the cost of an optimal solution). In its second step, Steiner trees are computed separately for each rate (within some approximation ratio α). The union of these trees is the final solution.

Replace each edge of rate 2^i by edges of the rate $2^0, 2^1, \ldots, 2^{i-1}, 2^i$, respectively. In the new network, all edges of a specific rate form a Steiner tree. Since the optimal cost in this new network is no more than twice the cost of the rounded up instance, taking the union of all the computed Steiner trees introduces another factor of two to the approximation ratio. Thus the final approximation factor is $2 \cdot \alpha \cdot 2 = 4\alpha$.

Using a randomization technique, Charikar, Naor, and Schieber [4] reduce the approximation ratio to $e\alpha \approx 4.21$, where $e \approx 2.71$ is the Euler constant and $\alpha \approx 1.55$ is the currently best approximation ratio for the Steiner Tree problem. The approximation factor has been further improved to 3.802 by Karpinski et al. [6].

C. *Algorithms for Two or Three Rates*

In practice, it is often the case that only few distinct rates are requested by the terminals. This is why the QoS problem with two or three rates has a long history, having been considered in [1], [2], [8], and [15]. The previously-best results of [8] and [15] have produced algorithms with approximation factor equal to 2.667 (provided that the MST heuristic is used to compute Steiner trees). Karpinski et al. [6] improved this ratio to 2.414 and showed that it can be further improved to 1.96 if more sophisticated time consuming Steiner tree algorithms are used.

III. Primal-Dual Motivated Algorithms

The problem can be formulated as an integer program as follows. Consider a network $G = (V, E, \mathit{length}, \mathit{rate})$ with a source node s and a set of terminal nodes. Let $r_1 < r_2 < \cdots < r_k$ be all rate values assigned to the terminals. It simplifies notation to assume that every node has a rate by considering an extra rate $r_0 = 0$ (assign rate r_0 to each non-terminal node). Also, we may assume that s has the highest rate. Construct a new network $G' = (V, E', \mathit{cost}, \mathit{rate})$ by replacing each edge e of G with k edges $(e, r_1), (e, r_2), \ldots, (e, r_k)$ and setting $\mathit{cost}((e, r_i)) = r_i \cdot \mathit{length}(e)$.

Let $x_{(e,r)}$ be a boolean variable denoting whether edge e is used at rate r in an optimum tree. The QoS Steiner tree problem can be formulated as

$$\min \sum_{(e,r)\in E'} x_{(e,r)} \cdot r \cdot \mathit{length}(e) \qquad \text{(III.1)}$$

$$\text{s.t.} \sum_{\substack{(e,r)\in\delta(C)\\ r\geq r_C}} x_{(e,r)} \geq 1, \quad \forall C \subseteq V \setminus \{s\} \qquad \text{(III.2)}$$

$$x_{(e,r)} \in \{0,1\} \qquad \text{(III.3)}$$

where $\delta(C)$ denotes the set of edges with exactly one endpoint in C and r_C denotes the maximum rate of a node in C. Note that (III.1) gives the cost of an optimal solution, while (III.2) guarantees that each terminal is connected to the source through a collection of edges of rate no less than its rate.

We relax the integrality constraints (III.3) and consider the dual linear program. For each (e, r), we define $C^*(e,r) = \{C \in V \setminus \{s\} : (e,r) \in \delta(C), r \geq r_C\}$. In words, $C^*(e,r)$ is the set of subsets C of $V \setminus \{s\}$ such that (e, r) has at least one endpoint in C and r is at least as large as r_C. Using this definition, the dual is as follows:

$$\begin{aligned} \max \quad & \sum_{C} y_C \\ \text{s.t.} \quad & \sum_{C \in C^*(e,r)} y_C \leq r \cdot \mathit{length}(e), \quad \forall (e,r) \\ & y_C \geq 0 \end{aligned}$$

A. *The Naive Primal-Dual Method*

The primal-dual framework applied to network design problems usually grows uniformly the dual variables associated to the "active" components of the current forest [5]. This approach fails to take into account the different rates of different nodes in the QoS problem. In Figure 3 we give a modification, referred to as the "Naive Primal-Dual" algorithm. Our modification takes into account the different rates by varying the speed at which each component grows. While the simulations in the ensuing sections show that this is a good method in practice, the solution it produces on some graphs may be very large compared to the optimal solution, as shown by the following example.

Input: A graph $G = (V, E, \mathit{length}, \mathit{rate})$ with a source s in V and a collection of terminals $S \subseteq V$.
Output: A QoS Steiner tree spanning the source and the terminals.

```
(1) Start from the spanning forest of G with no edges.
(2) Grow $y_C$ with speed $r_C$ for each "active" component $C$ of the
    current forest. (A component $C$ is inactive if it contains $s$ and all
    vertices of rate $r_C$.)
(3) Stop growing once the dual inequality for a pair $(e, r)$ becomes
    tight, with $e$ connecting two distinct components of the forest.
(4) Add $e$ to the forest, collapsing the two components.
(5) Terminate when there is no active component left.
(6) Keep an edge of the resulting tree at the minimum needed rate.
```

Fig. 3. The Naive Primal-Dual algorithm for the QoS Steiner Tree Problem.

The Frame Example. Consider two nodes of rate 1 connected by an edge of length 1 (see Figure 4). There is an arc between these two nodes, and on this arc there is a chain of nodes of rate ϵ. Each two consecutive nodes in the chain are at a distance δ from each other, where $\delta < 1$. Each extreme node in the chain is at a distance $\delta/2$ of its neighboring rate-1 node.

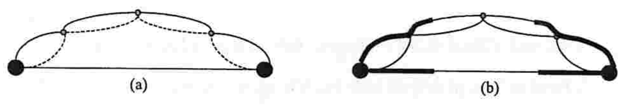


Fig. 4. The Restarting Primal-Dual avoids the mistake of the Naive Primal-Dual. Part (a) shows duplication of the edges. Part (b) shows the components growing along the respective edges.

Input: A Graph $G' = (V, E, cost, rate)$ with source s, and a collection of terminals S.
Output: A QoS Steiner Tree spanning the source and the terminal.

```
(1) Grow each active $C_{r_i}$ with speed $r_i$ along incident edges $(e, r_j)$,
    for $j \leq i$, picking edges which become tight.
(2) Continue this process until there is no active component of rate
    $r_k$.
(3) Remove all edges which are not necessary for maintaining con-
    nectivity of nodes of rate $r_k$.
(4) Accept (keep in the solution) and contract all edges of $C_{r_k}$ (i.e.,
    set their length/cost to 0)
(5) Restart the algorithm with the new graph
```

Fig. 5. The Restarting Primal-Dual algorithm for the QoS Steiner Tree Proble.

The Naive Primal-Dual applied to this graph connects the rate-ϵ nodes first, since $\frac{\delta}{2} < \frac{1}{2}$. So, the algorithm connects the rate-1 nodes via the rate-ϵ nodes, and not via the direct edge connecting them. Thus, the Naive Primal-Dual can make arbitrarily large errors (just take an arbitrarily long chain).

B. *Restarting Primal-Dual Algorithm*

An improved algorithm is given in Figure 5. One can easily see that this is a primal-dual algorithm. Indeed, each addition of an edge to the current solution is the result of growing dual variables. Moreover, since the feasibility requirement for edge a is $\Sigma_{a \in \delta(C)} y_C \leq r \cdot \mathit{length}(a)$, this addition preserves the feasibility of the dual solution. The algorithm maintains forests F^{r_i} given by the edges picked at rate r_i, and the connected components of F^{r_i}, seen as sets of vertices, are denoted in the algorithm by C_{r_i}. Such a component is *active* if $r_{C_{r_i}} = r_i$ and C_{r_i} is disjoint from components of higher rate.

The Restarting Primal-Dual avoids the mistake made by the Naive Primal-Dual on the frame example in Figure 4(a). Then, at time $\frac{\delta}{2}$ the rate-ϵ nodes become connected. This means that $\delta(1-\epsilon)$ of each rate-1 edge between the ϵ-rate nodes is not covered. Meanwhile, the rate-1 nodes are growing on the respective edges as shown in Figure 4(b).

Let us assume that the Restarting Primal-Dual uses the chain of rate-ϵ nodes to connect the two rate-1 nodes instead of the direct edge. This would imply that it takes less time to cover the chain, i.e., $\frac{1}{2}\delta(1-\epsilon)n \leq \frac{1}{2} - \frac{\delta}{2}$, where n is the number of rate-ϵ nodes. With ϵ small, we obtain $n\delta \leq 1$, so if the Restarting Primal-Dual uses the chain then it is correct to do so.

C. Primal-Dual 4.311-Approximation Algorithm

A primal-dual constant-factor approximation algorithm can be obtained based on the enhanced integer linear programming formulation below. It takes into account the fact that if a set $C \subset V \setminus \{s\}$ is connected to the source with edges of rate $r' > r_C$, then there should be at least **two** edges of rate r' with exactly one endpoint in C. The integer program is

$$\begin{aligned} \min \quad & \sum_{(e,r)\in E'} x_{(e,r)} \cdot r \cdot \mathit{length}(e) \\ \text{s.t.} \quad & \sum_{\substack{e\in\delta(C)\\ r=r_C}} x_{(e,r)} + \frac{1}{2} \sum_{\substack{e\in\delta(C)\\ r>r_C}} x_{(e,r)} \geq 1, \ \forall C \subseteq V \setminus \{s\} \\ & x_{(e,r)} \in \{0,1\} \end{aligned}$$

The corresponding dual of the LP relaxation is

$$\begin{aligned} \max \quad & \sum_{C\subseteq V\setminus\{s\}} y_C \\ \text{s.t.} \quad & \sum_{\substack{C\,:e\in\delta(C)\\ r_C=r}} y_C + \frac{1}{2} \sum_{\substack{C\,:e\in\delta(C)\\ r_C<r}} y_C \leq r \cdot \mathit{length}(e) \qquad \text{(III.4)} \\ & y_C \geq 0 \end{aligned}$$

The core of the algorithm is presented in Figure 6. Before that, we do a random bucketing of rates following [4]. Let a be a real (to be picked later) and γ be a real picked uniformly at random from the interval $[0..1]$. Every node of rate r is replaced by a node of rate $a^{\gamma+j}$, where j is the integer satisfying $a^{\gamma+j-1} < r \leq a^{\gamma+j}$.

The primal-dual part follows the classical framework [5], and works in stages starting from the lower rate to the highest. During the execution of the algorithm, edges are picked at a certain rate (in other words, $x_{(e,r)}$ is set to 1) one by one. Before executing step 3 at rate r for the ith time, the set of edges picked at rate r by the algorithm forms a forest F_i^r. (An edge can be picked at several rates, but it is kept in at most one such rate in the final solution because of the reverse delete step.) A component C of F_i^r is called an *r-component* if $r_C = r$.

Input: A graph $G = (V, E, \mathit{length}, \mathit{rate})$ with source s in V and a collection of terminals $S \subseteq V$.
Output: A QoS Steiner tree spanning the source and the terminal.

```
(1) For each $r = r_1, r_2, \ldots, r_k$, execute steps 2-6.
(2) Start from the spanning forest $F^r$ of $G$ with no edges.
(3) Grow $y_C$ uniformly for each $r$-component $C$ of the current forest $F^r$.
(4) Stop growing once the dual inequality for a pair $(e,r)$ becomes tight, with $e$ connecting two distinct components of $F^r$.
(5) Add $(e,r)$ to $F^r$, collapsing two of its components.
(6) Terminate when there is no $r$-component of $F^r$ left.
(7) Traversing the list of picked edges in reverse order, remove an edge $(e,r)$ from $F^r$ if after $(e,r)$'s removal the set of edges picked form a feasible tree.
```

Fig. 6. The 4.311-approximation algorithm for QoS Steiner Tree.

Using Constraint (III.4), it follows by induction on i that, for an edge e and a rate $a^{\gamma+j}$, we have

$$\begin{aligned} \sum_{\substack{C\,:e\in\delta(C)\\ r_C\leq a^{\gamma+j}}} y_C &\leq \mathit{length}(e) a^{\gamma+j} \sum_{j=0}^{i} \left(\frac{1}{2a}\right)^j \\ &\leq \mathit{length}(e) a^{\gamma+j} \frac{2a}{2a-1}. \end{aligned}$$

For an edge picked by the algorithm at rate r, Constraint (III.4) is tight and therefore

$$\sum_{\substack{C\,:e\in\delta(C)\\ r_C\leq a^{\gamma+j}}} y_C \geq \mathit{length}(e) \frac{2a-2}{2a-1} a^{\gamma+j}. \qquad \text{(III.5)}$$

Exactly as in [5], we have that the number of edges of rate r in the final solution which cross the active r-components at some moment (an edge being counted twice if it crosses two r-components) is at most twice the number of active r-components. Using Equation (III.5) and exactly the same argument as in Theorem 4.2 of [5], we obtain that the cost of the solution of the algorithm is bounded by $(2(2a-1)/(2a-2)) \sum y_C \leq ((2a-1)/(a-1))\, opt$, as any feasible solution for the dual linear program has value at most the value of any feasible solution of the primal.

The same argument as in [4] shows that the approximation ratio of the algorithm above is $(2a-1)/\ln a$. Numerically picking the best value for a, we obtain:

Theorem III.1: The output cost of the algorithm on Figure 6 is at most 4.311 times the optimum cost.

IV. Experimental Study

All algorithms except the very recent 4.311-approximation Primal-Dual were implemented in C++. The heuristics were compiled using `gpp` with `-O2` optimization, and run on a Sun workstation Ultra-60. The experiments were run on randomly generated testcases. Table I gives a comparison of the performance of of the aforementioned algorithms. The experiments were conducted in the presence of no Steiner nodes, respectively 50% Steiner nodes. Moreover, both arithmetic and geometric distributions of rates were tested.

Table I gives the results of a multitude of experiments; however, the results are fairly uniform throughout. It can be observed that the Naive Primal-Dual and the Charikar-Naor-Schieber algorithms most often produce comparable results which are slight improvements over the results produced by Maxemchuk's algorithm. The Restarting Primal-Dual typically produces the best result, which is typically $0.25 - 6$ percent better than the result produced by Maxemchuk's algorithm; this, however, occurs at the expense of greater CPU time. It can also be observed

TABLE I

COST IMPROVEMENT OVER MAXEMCHUCK'S ALGORITHM (%) AND CPU SECONDS FOR CHARIKAR-NAOR-SCHIEBER AND PRIMAL-DUAL ALGORITHMS (AVERAGES OVER 10 TESTCASES).

R	N	Maxem.	Charikar		Naive-PD		Restart-PD	
		CPU	%G	CPU	%G	CPU	%G	CPU
50% steiner nodes, geometric progression rates								
1	200	0.017	0.00	0.017	-0.01	0.544	-0.01	0.325
1	300	0.050	0.00	0.052	0.04	1.372	0.04	0.946
2	200	0.027	0.00	0.026	0.43	1.271	1.03	1.125
2	300	0.070	0.00	0.072	0.93	4.573	2.17	3.747
5	200	0.044	0.00	0.044	-2.13	1.490	1.30	5.321
5	300	0.123	0.00	0.120	-0.91	5.221	1.10	16.798
10	200	0.065	0.00	0.068	-2.53	1.636	0.66	17.848
10	300	0.180	0.00	0.176	-2.61	6.582	0.24	107.125
50% steiner nodes, arithmetic progression rates								
1	200	0.016	0.00	0.017	-0.01	0.541	-0.01	0.327
1	300	0.052	0.00	0.051	0.04	1.370	0.04	0.946
2	200	0.027	0.00	0.023	-0.69	1.373	-0.00	1.136
2	300	0.071	0.00	0.070	-0.32	4.491	0.24	3.773
5	200	0.043	-0.01	0.040	1.70	1.564	2.66	5.256
5	300	0.123	-0.10	0.107	1.92	5.392	4.19	17.271
10	200	0.067	1.79	0.043	4.25	1.556	6.11	16.856
10	300	0.181	2.36	0.126	3.38	5.444	5.73	92.575
0% steiner nodes, geometric progression rates								
1	100	0.002	0.00	0.002	0.00	0.052	0.00	0.077
1	200	0.028	0.00	0.028	0.00	0.251	0.00	0.465
2	100	0.007	0.00	0.007	1.21	0.088	1.69	0.185
2	200	0.038	0.00	0.033	2.14	0.698	2.31	1.517
5	100	0.012	0.00	0.013	1.24	0.120	2.82	0.665
5	200	0.059	0.00	0.056	-0.25	1.296	1.70	6.314
10	100	0.019	0.00	0.018	-0.68	0.133	1.63	1.953
10	200	0.090	0.00	0.091	-1.97	1.466	0.73	20.525
0% steiner nodes, arithmetic progression rates								
1	100	0.005	0.00	0.005	0.00	0.054	0.00	0.078
1	200	0.026	0.00	0.026	0.00	0.247	0.00	0.457
2	100	0.005	0.00	0.006	-0.11	0.111	-0.04	0.187
2	200	0.036	0.00	0.034	-0.02	1.078	0.30	1.570
5	100	0.011	-0.17	0.011	3.70	0.114	4.60	0.656
5	200	0.059	-0.15	0.052	3.13	1.235	3.85	5.952
10	100	0.019	2.62	0.012	6.65	0.113	7.12	1.922
10	200	0.091	2.67	0.058	5.83	1.203	6.38	17.689

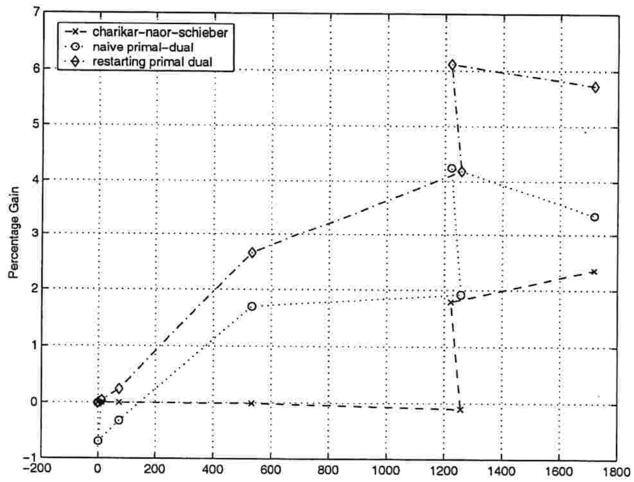


Fig. 7. The gain of several algorithms versus Maxemchuk's algorithm, 50% Steiner nodes.

that the difference between the algorithms increases as the number of rates increases. Figures 7 and 8 illustrate these results in graphical form.

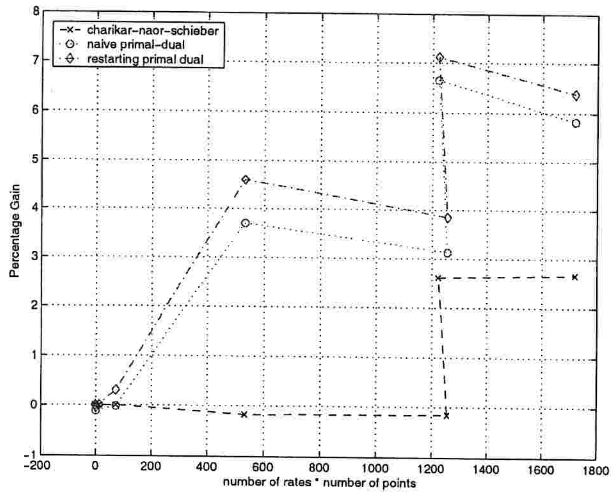


Fig. 8. The gain of several algorithms versus Maxemchuk's algorithm, 0% Steiner nodes.

REFERENCES

[1] A. Balakrishnan, T.L. Magnanti, and P. Mirchandani, *Modeling and Heuristic Worst-Case Performance Analysis of the Two-Level Network Design Problem*, Management Science **40**:846–867, (1994).

[2] A. Balakrishnan, T.L. Magnanti, and P. Mirchandani, *Heuristics, LPs, and Trees on Trees: Network Design Analyses*, Operations Research **44**:478–496, (1996).

[3] R. Bajaj, C.P. Ravikumar, and S. Chandra, *Distributed Delay Constrained Multicast Path Setup High Speed Networks*, Proceedings of the Fourth International Conference on High Performance Computing, 438–442, 1997.

[4] M. Charikar, J. Naor, and B. Schieber, *Resource Optimization in QoS Multicast Routing of Real-Time Multimedia*, Proceedings of the 19th Annual IEEE INFOCOM, (2000).

[5] M. Goemans and D. Williamson, *The Primal-Dual Method for Approximation Algorithms and its Application to Network Design Problems*, in Approximation Algorithms, D. Hochbaum, Ed., 1997.

[6] M. Karpinski, I. Mandoiu, A. Olshevsky, and A. Zelikovsky, *Improved Approximation Algorithms for Some Generalization of the Steiner Tree Problem*, Proc. Int. Workshop on Algorithms and Data Structures, LNCS 2748, (2003), pp. 401–411.

[7] N. Maxemchuk, *Video Distribution on Multicast Networks*, IEEE Journal on Selected Issues in Communications **15**:357-372 (1997).

[8] P. Mirchandani, *The multi-tier tree problem*, INFORMS Journal on Computing **8**:202–218 (1996).

[9] H.J. Promel and A. Steger, *A new approximation algorithm for the Steiner tree problem with performance ratio $\frac{5}{3}$*, Journal of Algorithms **36**:89–101, (2000).

[10] G. Robins and A. Zelikovsky, *Improved Steiner Tree Approximation in Graphs*, Proc. of ACM/SIAM Symposium on Discrete Algorithms (SODA 2000), 770–779.

[11] G.N. Rouskas and I. Baldine, *Multicast routing with end-to-end delay and delay variation constraints*, IEEE J. on Selected Areas in Communications **15**:346–356, (1997).

[12] H.F. Salama, D.S. Reeves, and Y. Viniotis, *Evaluation of Multicast Routing Algorithms for Real-Time Communication on High-Speed Networks*, IEEE Journal on Selected Areas in Communications, **3**:332–345 (1997).

[13] H. Takahashi and A. Matsuyama, *An Approximate Solution for the Steiner Tree Problem in Graphs*, Math. Japonic, **6**:573–577, (1980).

[14] H. Ural and K. Zhu, *An Efficient Distributed QoS Based Multicast Routing Algorithm*, Proceedings of the 21st IEEE International Performance, Computing, and Communication Conference, 27–36 (2002).

[15] G. Xue, G.-H. Lin, and D.-Z. Du, *Grade of service Steiner minimum trees in the Euclidean plane*, Algorithmica **31**:479–500, (2001).

Multicuts in Unweighted Graphs with Bounded Degree and Bounded Tree-Width

Gruia Călinescu[1]*, Cristina G. Fernandes[2]**, and Bruce Reed[3]***

[1] College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280. E-mail: gruia@cc.gatech.edu.

[2] Department of Computer Science, University of São Paulo Rua do Matao, 1010 05508-900 Sao Paulo - Brazil E-mail: cris@ime.usp.br.

[3] CNRS - Paris, France and Department of Computer Science, University of São Paulo, Brazil. E-mail: reed@ime.usp.br.

Abstract. The MULTICUT problem is defined as follows: given a graph G and a collection of pairs of distinct vertices (s_i, t_i) of G, find a smallest set of edges of G whose removal disconnects each s_i from the corresponding t_i. Our main result is a polynomial-time approximation scheme for MULTICUT in unweighted graphs with bounded degree and bounded tree-width: for any $\epsilon > 0$, we presented a polynomial-time algorithm with performance ratio at most $1 + \epsilon$. In the particular case when the input is a bounded-degree tree, we have a linear-time implementation of the algorithm. We also provided some hardness results. We proved that MULTICUT is still NP-hard for binary trees and that, unless $P = NP$, no polynomial-time approximation scheme exists if we drop any of the the three conditions: unweighted, bounded-degree, bounded-tree-width. Some of these results extend to the vertex version of MULTICUT.

1 Introduction

Multicommodity Flow problems have been intensely studied for decades [7, 11, 9, 13, 15, 17] because of their practical applications and also of the appealing hardness of several of their versions. The fractional version of a Multicut problem is the dual of a Multicommodity Flow problem and, therefore, Multicut is of similar interest [3, 9, 10, 13, 20].

The WEIGHTED MULTICUT is the following problem: given an undirected graph G, a weight function w on the edges of G, and a collection of k pairs of distinct vertices (s_i, t_i) of G, find a minimum weight set of edges of G whose removal disconnects each s_i from the corresponding t_i.

* Research supported in part by NSF grant CCR-9319106.

** Research partially supported by NSF grant CCR-9319106 and by FAPESP (Proc. 96/04505–2).

*** Research supported in part by ProNEx (MCT/FINEP) (Proj. 107/97) and FAPESP (Proc. 96/12111–4).

The particular case in which $k = 1$ is characterized by the famous *Max-Flow Min-Cut Theorem* [6], and is solvable in strongly polynomial time [4]. For $k = 2$, a variant of the Max-Flow Min-Cut Theorem holds [11, 12] and Multicut is solvable in polynomial time. For $k \geq 3$, the problem is NP-hard [3].

Since many variants of the Weighted Multicut are known to be NP-hard, we search for efficient approximation algorithms. The *performance ratio of an approximation algorithm* A for a minimization problem is the supremum, over all possible instances I, of the ratio between the weight of the output of A when running on I and the weight of an optimal solution for I. We say A is an α-*approximation algorithm* if its performance ratio is at most α. The smaller the performance ratio, the better.

The best known performance ratio for Weighted Multicut in general graphs is $O(\log k)$ [10]. Important research has been done for improving the performance ratio when the input graph G belongs to special classes of graphs. For planar graphs, Tardos and Vazirani [20], see also [13], give an approximate Max-Flow Min-Cut theorem and an algorithm with a constant performance ratio.

The case when the input graph is restricted to a tree has been studied in [9]. Unweighted Multicut problem (in which $w(e) = 1$ for all edges e of G) restricted to stars (trees of height one) is equivalent (including performance ratio) to Minimum Vertex Cover, by Proposition 1 in [9]. It follows that Unweighted Multicut restricted to stars is NP-hard and Max SNP-hard. In fact, getting a performance ratio better than two seems very hard, since getting a performance ratio better than two for Minimum Vertex Cover remains a challenging open problem [16]. Garg, Vazirani and Yannakakis give an algorithm in [9] with a performance ratio of two for the Weighted Multicut problem in trees. Note that the integral unweighted Multicommodity Flow problem in trees is solvable in polynomial time [9].

We find useful two variations of the Multicut problem. The VERTEX MULTICUT problem is: given an undirected graph G and a collection of k pairs of distinct nonadjacent vertices (s_i, t_i) of G called *terminals*, find a minimum set of nonterminal vertices whose removal disconnects each s_i from the corresponding t_i. The UNRESTRICTED VERTEX MULTICUT problem is: given an undirected graph G and a collection of k pairs of vertices (s_i, t_i) of G called *terminals*, find a minimum set of vertices whose removal disconnects each s_i from the corresponding t_i. (Note that in this variation, terminals might be removed.) Observe that Vertex Multicut is at least as hard as Unrestricted Vertex Multicut. From an instance of Unrestricted Vertex Multicut we can obtain an instance of Vertex Multicut by adding, for each s_i, a new vertex s'_i adjacent only to s_i, and, for each t_i, a new vertex t'_i adjacent only to t_i. Each pair (s_i, t_i) is substituted by the new pair (s'_i, t'_i). Solving Vertex Multicut in this instance is equivalent to solving Unrestricted Vertex Multicut in the original instance.

Both Vertex Multicut and Unrestricted Vertex Multicut might be of interest on their own. Garg, Vazirani and Yannakakis considered the weighted version of Vertex Multicut and proved that their algorithm in [10] achieves a performance ratio of $O(\log k)$ for the weighted version of Vertex Multicut in general graphs.

never remove vertices from S. At the end of the algorithm, no pair in $\mathcal{C}$ is active, meaning that S is a solution for the problem. For the minimality of S, note that the paths joining s_i to t_i in T for all marked pairs (s_i, t_i) form a pairwise disjoint collection of paths. Any solution should contain at least one vertex in each of these paths. But there are $|S|$ marked paths, meaning that any solution has at least $|S|$ vertices. This implies that $|S|$ is a minimum-size solution. Besides, it is not hard to see that the algorithm can be implemented in polynomial time.

2.1 Bounded-Tree-Width Graphs

Next we present a PTAS for Unrestricted Vertex Multicut in graphs with bounded tree-width. A PTAS consists of, for each $\epsilon > 0$, a polynomial-time algorithm for the problem with a performance ratio of at most $1 + \epsilon$. Let us describe such an algorithm.

The input of our algorithm is a graph $G = (V, E)$, a tree decomposition $\Theta = (T, (X_w)_{w \in V(T)})$ of G, and a set $\mathcal{C}$ of pairs of vertices of G.

Given a subgraph G' of G, denote by $\mathcal{C}(G')$ the set of pairs in $\mathcal{C}$ whose two vertices are in G', and by $G \setminus G'$ the subgraph of G induced by $V(G) \setminus V(G')$. For the description of the algorithm, all the instances we mention are on a subgraph G' of G and the set of pairs to be disconnected is $\mathcal{C}(G')$. So we will drop $\mathcal{C}(G')$ of the notation and refer to an instance only by the graph G'. Denote by $opt(G')$ the size (i.e., the number of vertices) of an optimal solution for G'.

Root the tree T (of the given tree decomposition) at a vertex r and consider an arbitrary ordering of the children of each vertex of T. For a vertex u of T, let $T(u)$ be the subtree of T rooted at u. Let $G(u)$ be the subgraph of G induced by the union of all X_w, $w \in V(T(u))$. Let $t = \lceil (tw(G) + 1)/\epsilon \rceil$.

Here is a general description of the algorithm: label the vertices of T in posorder. Find the lowest labeled vertex u such that an optimal solution for $G(u)$ has at least t vertices. If there is no such vertex, let u be the root. Find an approximate solution S_u for $G(u)$ such that $|S_u| \leq (1 + \epsilon) opt(G(u))$ and $X_u \subseteq S_u$. If u is the root of T, then output S_u. Otherwise, let $G' = G \setminus G(u)$ and let $\Theta' = (T', (X'_w)_{w \in V(T')})$ be the tree decomposition of G' where $T' = T \setminus T(u)$ and $X'_w = X_w \setminus V(G(u))$, for all $w \in V(T')$. Recursively get a solution S' for G'. Output $S = S' \cup S_u$.

Next we present a detailed description of the algorithm. It works in iterations. Iteration i starts with a subgraph G^{i-1} of G, a tree decomposition $\Theta^{i-1} = (T^{i-1}, (X_w^{i-1})_{w \in V(T^{i-1})})$ of G^{i-1} with T^{i-1} rooted at r, and a set S^{i-1} of vertices of G. Initially, $G^0 = G$, $\Theta^0 = \Theta$, $S^0 = \emptyset$ and $i = 1$. The algorithm halts when $G^{i-1} = \emptyset$. When G^{i-1} is nonempty, the algorithm starts calling a procedure *Get* (u, A), which returns a vertex u of T^{i-1} and a solution A for $G^{i-1}(u)$ such that $|A| \leq (1 + \epsilon) opt(G^{i-1}(u))$ and $X_u^{i-1} \subseteq A$. Then the algorithm starts a new iteration with $G^i = G^{i-1} \setminus G^{i-1}(u)$, $\Theta^i = (T^i, (X_w^i)_{w \in V(T^i)})$, where $T^i = T^{i-1} \setminus T^{i-1}(u)$ and $X_w^i = X_w^{i-1} \setminus V(G^{i-1}(u))$, for all $w \in V(T^i)$, and $S^i = S^{i-1} \cup A$. The formal description of the algorithm appears in Figure 2.1.

Algorithm:

```
$G^0 \leftarrow G$;
$\Theta^0 \leftarrow \Theta$;
$S^0 \leftarrow \emptyset$;
$i \leftarrow 1$;
while $G^{i-1} \neq \emptyset$ do
    $Get\ (u_i, A^i)$;        /*    $|A^i| \leq (1+\epsilon)opt(G^{i-1}(u_i))$ and $X_u^{i-1} \subseteq A^i$    */
    $G^i \leftarrow G^{i-1} \setminus G^{i-1}(u_i)$;
    $T^i \leftarrow T^{i-1} \setminus T^{i-1}(u_i)$;
    $X_w^i \leftarrow X_w^{i-1} \setminus V(G^{i-1}(u_i))$, for each $w \in V(T^i)$;
    $S^i \leftarrow S^{i-1} \cup A^i$;
    $i \leftarrow i+1$;
endwhile;
$f \leftarrow i-1$;
output $S^f$.
```

Fig. 1. The algorithm for Unrestricted Vertex Multicut in bounded-tree-width graphs.

We will postpone the description of *Get* (u, A) and, for now, assume that it works correctly and in polynomial time. The next lemma states a property of tree decompositions that we will use later.

Lemma 1. *Consider a graph G and a tree decomposition $\Theta = (T, (X_w)_{w \in V(T)})$ of G. Let u be a vertex of T, x be a vertex of $G(u)$ and y be a vertex of $G \setminus G(u)$. Then any path in G from x to y contains a vertex of X_u.*

Next we prove that the output of the algorithm is in fact a solution.

Lemma 2. *S^f is a solution for G.*

Proof. Let (s, t) be a pair in $\mathcal{C}$ and P be a path in G from s to t. We need to show that there is a vertex of P in S^f. Note that the vertex sets $V(G^{i-1}(u_i))$ define a partition of $V(G)$.

Let i be such that s is in $G^{i-1}(u_i)$. If all vertices of P lie in $G^{i-1}(u_i)$ then, in particular, both s and t are in $G^{i-1}(u_i)$, which means $(s,t) \in \mathcal{C}(G^{i-1}(u_i))$. Since S^f contains a solution for $G^{i-1}(u_i)$, S^f must contain a vertex of P.

If not all vertices of P lie in $G^{i-1}(u_i)$, let y be the first vertex of P that does not lie in $G^{i-1}(u_i)$. If y is in $G^{i-1} \setminus G^{i-1}(u_i)$ then, by Lemma 1, there is a vertex of $X_{u_i}^{i-1}$ in the segment of P from s to y. Since $X_{u_i}^{i-1} \subseteq S^f$, there is a vertex of P in S^f. If y is not in $G^{i-1} \setminus G^{i-1}(u_i)$, then y is not in G^{i-1}. This means y is in $G^{j-1}(u_j)$, for some $j < i$. Moreover, s is in $G^{j-1} \setminus G^{j-1}(u_j)$ (because this is a supergraph of G^{i-1}). Again by Lemma 1, there is a vertex of $X_{u_j}^{j-1} \subseteq S^f$ in P, concluding the proof of the lemma. ■

The next lemma proves that the performance ratio of the algorithm is at most $1 + \epsilon$.

Lemma 3. $|S^f| \leq (1+\epsilon)opt(G)$.

From now on, we refer to Multicut as EDGE MULTICUT, to avoid confusion. Let us mention some results we obtained for Vertex Multicut and Unrestricted Vertex Multicut. We have a proof that Vertex Multicut is NP-hard in bounded-degree trees. Unrestricted Vertex Multicut is easier: it is polynomially solvable in trees, but it becomes NP-hard in bounded-degree series-parallel graphs.

The tree-width notion (first introduced by Robertson and Seymour [19]) seems to often capture a property of the input graph which makes hard problems easy. Various NP-hard problems, like Clique or Coloring, have a polynomial-time algorithm (linear time in fact) if the input graph has bounded tree-width (see for example [2]). We will present the formal definition of tree-width in Section 2.

Bounded tree-width can also be used to obtain good approximation algorithms for those problems that remain NP-hard even if restricted to graphs of bounded tree-width. In our case, Unrestricted Vertex Multicut is NP-hard in graphs of tree-width at most two, since this class of graphs coincides with the series-parallel graphs (see for example [21]). We give a straightforward PTAS for Unrestricted Vertex Multicut in graphs of bounded tree-width.

We present an approximation-ratio preserving reduction from Edge Multicut to Unrestricted Vertex Multicut. If the Edge Multicut instance graph has bounded degree and bounded tree-width, the Unrestricted Vertex Multicut instance obtained by the reduction has bounded tree-width. Combining the reduction with the PTAS for Unrestricted Vertex Multicut in graphs of bounded tree-width, we obtain a PTAS for Unweighted Edge Multicut in graphs with bounded degree and bounded tree-width. This is the main result of the paper. Note that, according to [8, page 140, Theorem 6.8], a FPTAS cannot exist for this problem, unless P=NP.

We also have a linear-time implementation of our PTAS for Edge Multicut in bounded-degree trees. The running time of our implementation is $O((n + k)\lceil 1/\epsilon\rceil d^{d\lceil 1/\epsilon\rceil+2})$, where n is the number of vertices of the tree, k is the number of (s_i, t_i) pairs, d is the maximum degree of the tree and $1 + \epsilon$ is the desired approximation ratio of the algorithm. The size of the input is $\Theta(n + k)$.

We show that Edge Multicut is still NP-hard for binary (degree bounded by three) trees. Thus, on the class of graphs of bounded degree and bounded tree-width, which contains binary trees, Edge Multicut is easier (there is a PTAS) than on general graphs, yet still NP-hard. Identifying this class is the main theoretical result of this paper.

Hardness results indicate why we cannot eliminate any of the three restrictions—unweighted, bounded degree and bounded tree-width—on the input graph and still obtain a PTAS. It is known [1] that for a Max SNP-hard problem, unless P=NP, no PTAS exists. We have already seen that Unweighted Edge Multicut is Max SNP-hard in stars [9], so letting the input graph have unbounded degree makes the problem harder. We show that Weighted Edge Multicut is Max SNP-hard in binary trees, therefore letting the input graph be weighted makes the problem harder. Finally, we show that Unweighted Edge Multicut is Max SNP-hard if the input graphs are walls. Walls, to be formally defined in Section 4, have degree at most three and there are walls with tree-width as large as we wish. We conclude that letting the input graph have unbounded tree-width makes the problem significantly harder.

In Section 2 we present the polynomial-time algorithm for Unrestricted Vertex Multicut in trees and the polynomial-time approximation scheme for Unrestricted Vertex Multicut in bounded-tree-width graphs. In Section 3, we show the approximation-preserving reduction from Edge Multicut to Unrestricted Vertex Multicut. Finally, in Section 4 we present our hardness results.

2 Algorithms for Unrestricted Vertex Multicut

In this section we concentrate on Unrestricted Vertex Multicut. We present a polynomial-time algorithm for trees and a PTAS for graphs with bounded tree-width. Let us start defining tree-width.

Let G be a graph and Θ be a pair $(T, (X_w)_{w \in V(T)})$, which consists of a tree T and a multiset whose elements X_w, indexed by the vertices of T, are subsets of $V(G)$. For a vertex v of G, we denote by F_v the subgraph of T induced by those vertices w of T for which X_w contains v. Then Θ is called a *tree decomposition of* G if it satisfies the two conditions below:

(1) For every edge $e = xy$ of G, there is a vertex w of T such that $\{x, y\} \subseteq X_w$;

(2) For every vertex v of G, the subgraph F_v of T is a tree.

The *width of* Θ is the maximum, over all vertices w of T, of $|X_w| - 1$, and the *tree-width of* G, denoted by $tw(G)$, is the minimum of the widths of all tree decompositions of G.

Consider an instance of Unrestricted Vertex Multicut, that is, a graph $G = (V, E)$ and a set $\mathcal{C}$ of pairs (s_i, t_i) of vertices of G. We say a pair (s_i, t_i) in $\mathcal{C}$ is *disconnected by a set* $S \subseteq V$ if s_i is disconnected from t_i in the subgraph of G induced by $V - S$. A set S is a *solution for* G if S disconnects all pairs (s_i, t_i) in $\mathcal{C}$. If S has minimum size (i.e., minimum number of vertices), then S is an *optimal solution for* G.

Now, let us describe the polynomial-time algorithm for trees. The input of the algorithm is a tree T and a set $\mathcal{C}$ of pairs (s_i, t_i) of vertices of T.

Consider the tree T rooted at an arbitrary vertex and consider also an arbitrary ordering of the children of each vertex (so that we can talk about postorder).

Algorithm:

```
Input: a tree $T$.
Start with $S = \emptyset$.
Call a pair $(s_i, t_i)$ in $\mathcal{C}$ *active* if it is not disconnected by $S$.
Traverse the tree in postorder.
When visiting vertex $v$, if $v$ is the least common ancestor of some active
pair $(s_i, t_i)$ in $\mathcal{C}$, then insert $v$ into $S$ and mark $(s_i, t_i)$.
Output $S$.
```

Clearly the following invariant holds: all non-active pairs in $\mathcal{C}$ are disconnected by S. A pair in $\mathcal{C}$ that becomes non-active does not go back to active since we

Proof. We have that

$$|S^f| = \sum_{i=1}^{f} |A^i| \le \sum_{i=1}^{f} (1+\epsilon)\,opt(G^{i-1}(u_i))$$
$$= (1+\epsilon) \sum_{i=1}^{f} opt(G^{i-1}(u_i)) \le (1+\epsilon)\,opt(G),$$

because the subgraphs $G^{i-1}(u_i)$ are vertex disjoint. ■

Now we proceed with the description of a straightforward polynomial-time implementation of *Get* (u, A).

Search the vertices of the tree T^{i-1} in postorder. Stop if the vertex u being visited is either the root or is such that $opt(G^{i-1}(u)) \ge t$. Let us show how we check whether $opt(G^{i-1}(u)) \ge t$ in polynomial time.

If we are searching vertex u, it is because all children of u have been searched and have an optimal solution with less than t vertices. Compute an optimal solution for each child v of u. This can be done in $O(n^{t+1})$ time by brute force: check all subsets of $G(v)$ of size at most t. The time is polynomial, since $t = \lceil (tw(G)+1)/\epsilon \rceil$ is fixed. Let s be the sum of the sizes of the solutions for the children of u.

Let us show that the optimum of $G^{i-1}(u)$ is at most $s + tw(G) + 1$. We do this by presenting a solution B for $G^{i-1}(u)$ of size at most $s + tw(G) + 1$. The set B is the union of X_u and an optimal solution for $G^{i-1}(v)$, for each child v of u. Thus $|B| \le |X_u| + s \le tw(G) + 1 + s$. Now we must prove that B is in fact a solution for $G^{i-1}(u)$. Let (s, t) be a pair in $\mathcal{C}(G^{i-1}(u))$ and P be a path in $G^{i-1}(u)$ from s to t. We need to show that there is a vertex of P in B. If there is a vertex of P in X_u, then clearly B contains a vertex of P. If, on the other hand, P contains no vertex of X_u, we must have all vertices of P in the same $G^{i-1}(v)$, for some child v of u, by Lemma 1. But B contains a solution for $G^{i-1}(v)$. Therefore, B contains a vertex of P. This completes the proof that B is a solution for $G^{i-1}(u)$, and so the optimum of $G^{i-1}(u)$ is at most $s + tw(G) + 1$.

Now, let us proceed with the description of *Get* (u, A). If $s < t$, then $opt(G(u)) \le s + tw(G) + 1 < t + tw(G) + 1$, and we can compute in polynomial time an optimal solution A_0 for $G(u)$ (by brute force in $O(n^{t+tw(G)+2})$ time, which is polynomial since $t = \lceil (tw(G)+1)/\epsilon \rceil$). If $|A_0| < t$ then proceed to the next vertex in postorder. If $|A_0| \ge t$, then we output u and the set $A = A_0 \cup X_u$. Note that in fact $opt(G^{i-1}(u)) = |A_0| \ge t$, $X_u \subseteq A$ and $|A| \le opt(G^{i-1}(u)) + (tw(G)+1) \le opt(G^{i-1}(u)) + t\epsilon \le (1+\epsilon)\,opt(G^{i-1}(u))$, as desired. On the other hand, if $s \ge t$, then $t \le s \le opt(G^{i-1}(u)) \le s + tw(G) + 1 \le s + t\epsilon \le opt(G^{i-1}(u)) + opt(G^{i-1}(u))\epsilon = (1+\epsilon)\,opt(G^{i-1}(u))$. Thus B (from the previous paragraph) is a solution for $G^{i-1}(u)$ of size at most $s + tw(G) + 1 \le (1+\epsilon)\,opt(G^{i-1}(u))$ that can be computed in polynomial time. Moreover, $X_u \subseteq B$. So in this case we output u and $A = B$. This finishes the description of *Get* (u, A).

3 Edge Multicut

In this section we show that Edge Multicut can be reduced to Unrestricted Vertex Multicut by a reduction that preserves approximability.

The reduction has the following property. If the instance of Edge Multicut is a graph with bounded degree and bounded tree-width, then the corresponding instance of Unrestricted Vertex Multicut has bounded tree-width.

Given a graph $G = (V, E)$, the *line graph of* G is the graph whose vertex set is E and such that two of its vertices (edges of G) are adjacent if they share an endpoint in G. In other words, the line graph of G is the graph (E, L), where $L = \{ef : e, f \in E \text{ and } e \text{ and } f \text{ have a common endpoint}\}$.

Consider an instance of Edge Multicut, that is, a graph $G = (V, E)$ and a set $\mathcal{C}$ of pairs of distinct vertices of G. Let us describe the corresponding instance of Unrestricted Vertex Multicut. The input graph for Unrestricted Vertex Multicut is the line graph of G, denoted by G'. Now let us describe the set of pairs of vertices of G'. For each pair (s, t) in $\mathcal{C}$, we have in $\mathcal{C}'$ all pairs (e, f) such that e has s as endpoint and f has t as endpoint.

Clearly G' can be obtained from G in polynomial time. Note that $\mathcal{C}'$ has at most $k\Delta^2$ pairs, where $k = |\mathcal{C}|$ and Δ is the maximum degree of G. Also $\mathcal{C}'$ can be obtained from G and $\mathcal{C}$ in polynomial time.

The following theorem completes the reduction.

Theorem 1. *S is a solution for Edge Multicut in G if and only if S is a solution for Unrestricted Vertex Multicut in G'.*

Proof. Consider a solution S of Edge Multicut in G, that is, a set S of edges of G such that any pair in $\mathcal{C}$ is disconnected in $(V, E - S)$. Note that $S \subseteq E(G) = V(G')$. Let us verify that the removal of S from G' disconnects all pairs in $\mathcal{C}'$. For any pair (e, f) in $\mathcal{C}'$, there are s and t in $V(G)$ such that s is an endpoint of e, t is an endpoint of f and the pair (s, t) is in $\mathcal{C}$. Moreover, a path P' in G' from e to f corresponds to a path P in G from s to t whose edges are a subset of the vertices in P' (which are edges of G). Since S is a solution of Edge Multicut in G, there must be an edge of P in S, which means that there is a vertex of P' in S. Hence S is a solution for Unrestricted Vertex Multicut in G'.

Conversely, let S be a solution for Unrestricted Vertex Multicut in G', that is, S is a set of edges of G whose removal from G' disconnects all pairs of vertices of G' in $\mathcal{C}'$. Let (s, t) be a pair in $\mathcal{C}$, and P a path in G from s to t. (Recall that, by the description of Edge Multicut, $s \neq t$.) Let e be the first edge of P and f the last one (possibly e=f). Clearly s is incident to e, and t to f. Thus (e, f) is a pair in $\mathcal{C}'$. Corresponding to P, there is a path P' in G' from e to f containing as vertices all edges of P. Since S is a solution for Unrestricted Vertex Multicut in G' and (e, f) is in $\mathcal{C}'$, S must contain a vertex of P'. Therefore there is an edge of P in S, which implies that S is a solution of Edge Multicut in G. ■

The next lemma shows the previously mentioned property of this reduction.

Lemma 4. *If G has bounded degree and bounded tree-width, then the line graph of G has bounded tree-width.*

Proof. Denote by G' the line graph of G. Let us present a tree decomposition of G' whose tree-width is at most $(tw(G)+1)\Delta$, where Δ is the maximum degree of G.

Let $\Theta = (T, (X_u)_{u \in V(T)})$ be a tree decomposition of G of width $tw(G)$. For each $u \in V(T)$, let Y_u be the set of edges of G incident to some vertex in X_u. First let us prove that $\Theta' = (T, (Y_u)_{u \in V(T)})$ is a tree decomposition of G'. For this, given an edge e of G, denote by T_e the subgraph of T induced by those vertices in T for which Y_u contains e. We shall prove that (1) any edge h of G' has both endpoints in Y_u, for some u in $V(T)$; and (2) that T_e is a tree for any edge e of G'.

The endpoints of an edge h of G' are two edges e and f of G with a common endpoint, say, v. But $v \in X_u$ for some $u \in V(T)$. This implies that both e and f belong to Y_u, proving (1). For (2), let e be a vertex of G', that is, an edge $e = xy$ of G. For any u such that $e \in Y_u$, we must have that either $x \in X_u$ or $y \in X_u$. Therefore $T_e = T_x \cup T_y$. We know that the subgraphs T_x and T_y of T are subtrees of T. Moreover, T_x and T_y have a vertex in common, because both x and y belong to the same X_u, for some $u \in V(T)$. Hence T_e is a subtree of T. This completes the proof that Θ' is a tree decomposition of G'.

To verify that the width of Θ' is at most $(tw(G)+1)\Delta$, just note that $|Y_u| \leq |X_u|\Delta$, for all $u \in V(T)$. ■

The next corollary is a consequence of the previous reduction and the PTAS given in Section 2.1.

Corollary 1. *There is a PTAS for Edge Multicut in bounded-degree graphs with bounded tree-width.*

In fact we know how to implement the PTAS given in Section 2.1, for Edge Multicut in bounded-degree trees, in time $O((n+k)\lceil 1/\epsilon \rceil d^{d\lceil 1/\epsilon \rceil + 2})$, where n is the number of vertices of the tree, k is the number of (s_i, t_i) pairs, d is the maximum degree of the tree and $1+\epsilon$ is the desired approximation ratio of the algorithm. The size of the input is $\Theta(n+k)$. We omit the description of this linear-time implementation in this extended abstract.

4 Complexity Results

In this section, we examine the complexity of Edge, Vertex and Unrestricted Vertex Multicut. First we prove that Edge and Vertex Multicut are NP-hard in bounded-degree trees, while Unrestricted Vertex Multicut is NP-hard in series-parallel graphs of bounded degree. Second, we show that the Weighted Edge Multicut is Max SNP-hard in binary tree. Finally we prove that Edge, Vertex and Unrestricted Vertex Multicut are Max SNP-hard in walls (defined in Section 4).

Theorem 2. *Edge Multicut in binary trees is NP-hard.*

Proof. The reduction is from 3-SAT, a well-known NP-complete problem [8].

Consider an instance Φ of 3-SAT, that is, a set of m clauses $C_1, C_2, \ldots, C_m$ on n variables $x_1, x_2, \ldots, x_n$, each clause with exactly three literals.

Let us construct an instance of Edge Multicut: a binary tree T and a set of pairs of distinct vertices of T. The tree T is built as follows. For each variable x_i, there is a gadget as depicted in Figure 2 (a). The gadget consists of a binary tree with three vertices: the root and two leaves, one labeled x_i and the other labeled $\overline{x}_i$. For each clause C_j, there is a gadget as depicted in Figure 2 (b). The gadget consists of a binary tree with five vertices: the root, one internal vertex and three leaves, each one labeled by one of the literals in C_j.

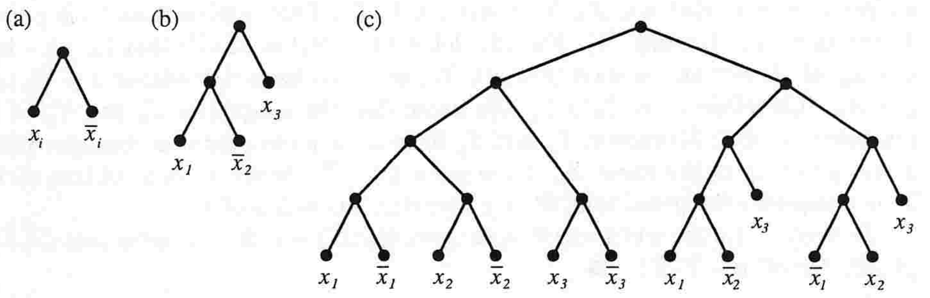


Fig. 2. (a) The gadget for variable x_i. (b) The gadget for clause $C_j = \{x_1, \overline{x}_2, x_3\}$. (c) Tree T built for the instance $\Phi = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3)$, that is, $C_1 = \{x_1, \overline{x}_2, x_3\}$ and $C_2 = \{\overline{x}_1, x_2, x_3\}$.

The tree T is built from these $n + m$ gadgets by arbitrarily connecting them using new vertices to get a binary tree. See Figure 2 (c) for an example.

Next, we give the set of pairs of vertices of T in our instance. For each variable x_i, there is a pair with the vertices labeled x_i and $\overline{x}_i$ in its gadget. For each clause C_j, there are two pairs: one formed by the two leaves that are siblings and the other formed by the last leaf and the internal vertex. Finally, each vertex labeled $\tilde{x}_i$ in the gadget for a clause is paired to the vertex labeled $\tilde{x}_i$ in the gadget for the variable x_i, where $\tilde{x}_i \in \{x_i, \overline{x}_i\}$. This ends the construction of the instance for Edge Multicut. Note that all this can be done in polynomial time in the size of Φ.

The next lemma completes the proof of Theorem 2. ■

Lemma 5. *Φ is satisfiable if an only if there is a solution for T of size exactly $n + 2m$. Moreover, we can construct in polynomial time such a solution for T from a truth assignment for Φ and vice versa.*

Proof. Assume Φ is satisfiable. Let us present a solution S for T of size exactly $n + 2m$. The edge set S consists of two types of edges:

1. For each variable x_i, S contains the edge in the gadget for x_i incident to the leaf labeled x_i if $x_i =$ *TRUE* or to the leaf labeled $\overline{x}_i$ if $x_i =$ *FALSE*.

2. For each clause C_j, S contains two distinct edges in the gadget for C_j. These edges are such that (1) they disconnect the two pairs in the gadget, and (2) the only leaf that is still connected to the root of the gadget is a leaf with a label $\tilde{x}_i \in C_j$ such that $\tilde{x}_i = TRUE$. (The four possible choices for the two edges are shown in Figure 3.)

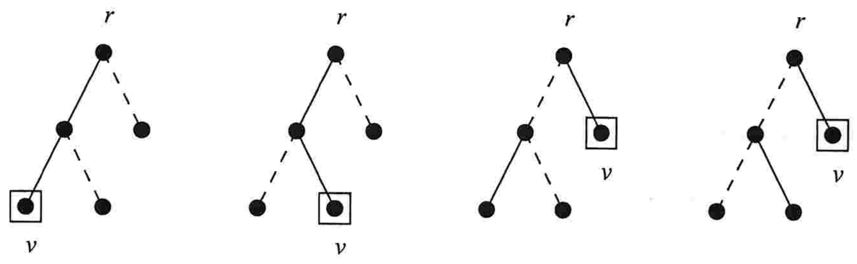


Fig. 3. Possible choices of two edges, the dashed edges, in the gadget for a clause that leave exactly one leaf (the marked leaf v) connected to the root r.

Clearly such set S has exactly $n + 2m$ edges and can be constructed in polynomial time from Φ. Let us prove that S is in fact a solution for T. It is easy to see that S disconnected the pairs for the variables, and the pairs for the clauses. The remaining pairs consist of two vertices labeled by a literal $\tilde{x}_i$, one in the variable gadget for x_i and the other in a clause gadget. If $\tilde{x}_i = TRUE$, then the edge in the variable gadget incident to the vertex labeled $\tilde{x}_i$ is in S, guaranteeing that the pair is disconnected. If $\tilde{x}_i = FALSE$, then the vertex labeled $\tilde{x}_i$ in the clause gadget is disconnected from the root of this gadget, and therefore, from the gadget for x_i. Thus S is a solution for T, and it has exactly $n + 2m$ edges.

Let us prove the inverse implication. Assume there is a solution S for T with exactly $n + 2m$ edges: one per variable and two per clause (one for each of the "disjoint" pairs). More specifically, S has exactly one edge in each variable gadget, and exactly two edges in each clause gadget in one of the configurations of Figure 3. Set $x_i = TRUE$ if the edge of S in the gadget for x_i is incident to the vertex labeled x_i; set $x_i = FALSE$ otherwise. Clearly, we can determine this truth assignment in polynomial time.

For each clause C_j, there is exactly one leaf v in the gadget for C_j that is connected to the root r of the gadget. Let $\tilde{x}_i \in \{x_i, \overline{x}_i\}$ be the label for this leaf. There is a pair formed by this leaf v and the leaf in the gadget for x_i whose label is $\tilde{x}_i$. In S, there must be an edge e in the path between these two leaves. Since leaf v is connected to the root r of the gadget for C_j and all edges in S are either in a variable gadget or in a clause gadget, this edge e has to be in the variable gadget. This means e is the edge incident to the leaf labeled $\tilde{x}_i$ in the gadget for x_i. Hence $\tilde{x}_i = TRUE$, and the clause is satisfied. Since this holds for all the clauses, the given assignment makes Φ *TRUE*, implying that Φ is satisfiable. ∎

Theorem 3. *Vertex Multicut in trees with maximum degree at most four is NP-hard.*

We omit the proof. The construction is similar to the one used in Theorem 2.

Theorem 4. *Unrestricted Vertex Multicut in series-parallel graphs with maximum degree at most three is NP-hard.*

We omit the proof. The construction is similar to the one used in Theorem 2.

Theorem 5. *Weighted Edge Multicut is Max SNP-hard in binary trees.*

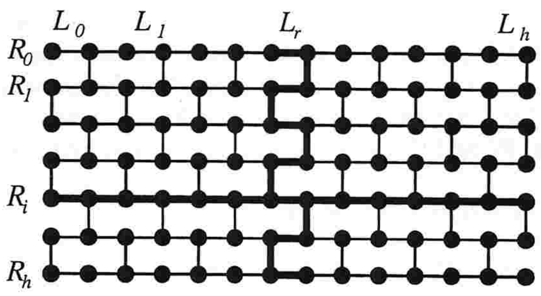
Proof sketch. Let us reduce Edge Multicut in stars to Weighted Edge Multicut in binary trees. From an instance of the Unweighted Edge Multicut restricted to stars, we construct an instance of the Weighted Edge Multicut restricted to binary trees in the following way: for each leaf of the star S, there is a corresponding leaf in the binary tree T. The pairs are the same (we may assume there is no pair involving the root of the star). We connect the leaves of T arbitrarily into a binary tree. The edges in T incident to the leaves get weight one and all other edges of T get weight $2n+1$, where n is the number of leaves in the star S (which is the same as the number of leaves in the tree T we construct). Any solution within twice the optimum for the Weighted Edge Multicut instance we constructed will contain only edges of T incident to the leaves, since any other edge is too heavy (removing all edges incident to the leaves, we get a solution of weight n). Then it is easy to see that any optimal solution for the Weighted Edge Multicut instance we constructed corresponds to an optimal solution for the original Unweighted Multicut star instance, and *vice versa*. Also approximability is preserved by this reduction. ■

A *wall of height h* consists of $h+1$ vertex disjoint paths $R_0, \ldots, R_h$, which we call *rows*, and $h+1$ vertex disjoint paths $L_0, \ldots, L_h$, which we call *columns*. A wall of height six is depicted in Figure 4 (a). The reader should be able to complete the definition by considering Figure 4 (a). The formal definition is as follows. Each row is a path of $2h+2$ vertices. Each column, a path with $2h+2$ vertices. Column r contains the $(2r+1)^{st}$ and the $(2r+2)^{nd}$ vertices of all rows, as well as the edge between them. For $i < h$ and even, each L_r contains an edge between the $(2r+2)^{nd}$ vertex of R_i and the $(2r+2)^{nd}$ vertex of R_{i+1}. For $i < h$ and odd, each L_r contains an edge between the $(2r+1)^{st}$ vertex of R_i and the $(2r+1)^{st}$ vertex of R_{i+1}. These are all the edges of the wall.

We prove that Edge, Vertex and Unrestricted Vertex Multicut are Max SNP-hard in walls. This means, by Arora et al. [1], that there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm for any of the three versions of Multicut with performance ratio at most $1+\epsilon$ implies that P=NP.

As in [18], we use the concept of *L-reduction*, which is a special kind of reduction that preserves approximability.

Let A and B be two optimization problems. We say A *L-reduces to B* if there are two polynomial-time algorithms f and g, and positive constants α and β, such that for each instance I of A,

(a)


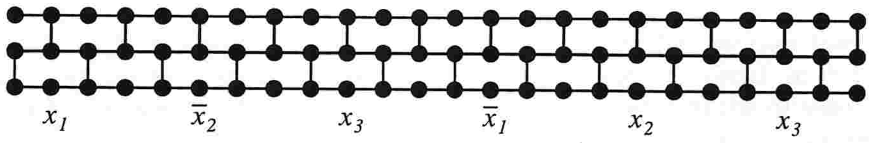
(b)


Fig. 4. (a) A wall of height six. The dark edges indicate row R_i and column L_r. (b) The three last rows of the wall built from $\Phi = (x_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee x_2 \vee x_3)$.

1. Algorithm f produces an instance $I' = f(I)$ of B, such that the optima of I and I', of costs denoted $Opt_A(I)$ and $Opt_B(I')$ respectively, satisfy $Opt_B(I') \leq \alpha \cdot Opt_A(I)$, and
2. Given any feasible solution of I' with cost c', algorithm g produces a solution of I with cost c such that $|c - Opt_A(I)| \leq \beta \cdot |c' - Opt_B(I')|$.

Theorem 6. *Edge, Vertex and Unrestricted Vertex Multicut are Max SNP-hard in walls.*

Proof sketch. The reduction is from the well-known Max SNP-hard problem MAX 3-SAT [18]. We show the reduction for Unrestricted Vertex Multicut. The other two reductions are similar.

The first part of the L-reduction is the polynomial-time algorithm f and the constant α. Given any instance Φ of MAX 3-SAT, f produces an instance $W, \mathcal{C}$ of Unrestricted Vertex Multicut such that W is a wall. Also, the cost of the optimum of $W, \mathcal{C}$ in Unrestricted Vertex Multicut, denoted $Opt_{MC}(W, \mathcal{C})$, is at most α times the cost of the optimum of Φ in MAX 3-SAT, denoted by $Opt_{SAT}(\Phi)$, i.e., $Opt_{MC}(W, \mathcal{C}) \leq \alpha \cdot Opt_{SAT}(\Phi)$.

Consider an instance Φ of MAX 3-SAT, that is, a collection of m clauses on n variables $x_1, \ldots, x_n$, each consisting of exactly three literals. Let us describe the corresponding instance for Unrestricted Vertex Multicut. The wall W is a wall of height $6m$. To describe the collection $\mathcal{C}$ of pairs of vertices of W, consider the last row of W partitioned into m same length paths, each one associated to one of the clauses of Φ. Each path has length 12. Label the $2^{nd}, 6^{th}$ and 10^{th} vertices in the j^{th} path each with one of the literals in the j^{th} clause. See Figure 4 (b) for an example. For each pair of vertices u, v in W, u labeled x_i and v labeled $\overline{x}_i$, include into $\mathcal{C}$ the pair u, v. For each clause, include three pairs. The three pairs formed by each two of the vertices labeled by its three literals. This ends the description of the instance of Unrestricted Vertex Multicut.

First note that W and $\mathcal{C}$ can be obtained in polynomial time in the size of Φ.

Lemma 6. $Opt_{MC}(W, \mathcal{C}) \leq 6 \cdot Opt_{SAT}(\Phi)$.

Proof sketch. $W, \mathcal{C}$ clearly has a solution of size $3m$. Also $Opt_{SAT}(\Phi) \geq m/2$. ■

Lemma 7. *From a solution to Φ of size s, $0 \leq s \leq m$, we can obtain a solution to $W, \mathcal{C}$ of size $3m - s$ and vice versa.*

Proof sketch. Given an assignment that satisfies s clauses of Φ, let S be the set of all labeled vertices of W except one labeled vertex per satisfied clause. Choose to not include in S a vertex labeled by a literal that is assigned TRUE. One can verify that this set S is a solution for $W, \mathcal{C}$ of size $3m - s$.

Now, consider a solution S for $W, \mathcal{C}$ of size $3m - s$. Since W has height $6m$, there is a row R_i of W which has no vertex of S. Set to TRUE any literal which appears as a label of a vertex of W that is connected to R_i after the removal of S. If some variable was not assigned a value by this rule, assign it an arbitrary value. Note that, since vertices labeled x_i are not connected to vertices labeled $\overline{x}_i$ after the removal of S, the assignment is well-defined. Consider the six columns of the wall corresponding to the j^{th} clause of Φ. S should contain at least two vertices in these columns, otherwise there would be a path connecting at least two of the labeled vertices in these columns. This means that at least s clauses have only two vertices removed from their columns of W. Thus one of the labeled vertices is connected to row R_i, meaning that this clause is satisfied. ■

The previous two lemmas can be used in an obvious way to show the reduction we presented is an L-reduction. ■

5 Acknowledgments

The first two authors would like to thank Howard Karloff for suggesting the problem, and for some helpful discussions.

References

1. S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof Verification and Hardness of Approximation Problems," *Proc. 33rd IEEE Symposium on Foundations of Computer Science*, 14-23, 1992.
2. S. Arnborg and J. Lagergren, "Problems Easy for Tree-Decomposable Graphs," *Journal of Algorithms*, 12 (2), 308-340, 1991.
3. E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour and M. Yannakakis, "The Complexity of Multiterminal Cuts," *SIAM Journal on Computing*, 23 (4), 864-894, 1994.

4. E. A. Dinic, "Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation," *Soviet Mathematics Doklady*, 11, 1277-1280, 1970.
5. G. Even, J. S. Naor, B. Schieber and L. Zosin, "Approximating Minimum Subset Feedback Sets in Undirected Graphs with Applications," *Proc.* 4^{th} *Israel Symposium on Theory of Computing and Systems*, 78-88, 1996.
6. L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, 8, 399-404, 1956.
7. L. R. Ford and D. R. Fulkerson, "A Suggested Computation for Maximal Multicommodity Network Flows," *Management Science*, 5, 97-101, 1958.
8. M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Co., 1979.
9. N. Garg, V. Vazirani and M. Yannakakis. "Approximate Max-Flow Min-(Multi)Cut Theorems and Their Applications," *SIAM Journal on Computing*, 25 (2), 235-251, 1996.
10. N. Garg, V. Vazirani and M. Yannakakis. "Primal-Dual Approximation Algorithms for Integral Flow and Multicut in Trees," *Algorithmica*, 18 (1), 3-20, 1997.
11. T. C. Hu, "Multicommodity Network Flows," *Operations Research*, 9, 898-900, 1963.
12. A. Itai, "Two-Commodity Flow," *Journal of ACM*, 25, 596-611, 1978.
13. P. Klein, A. Agrawal, R. Ravi and S. Rao, "Approximation through Multicommodity Flow," *Proc.* 31^{st} *IEEE Symposium on Foundations of Computer Science*, 726-737, 1990.
14. P. Klein, S. Plotkin, S. Rao and E. Tardos, "Approximation Algorithms for Steiner and Directed Multicuts," *Journal of Algorithms,* 22 (2), 241-269, 1997.
15. F. T. Leighton and S. Rao, "An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Application to Approximation Algorithms," *Proc.* 29^{th} *IEEE Symposium on Foundations of Computer Science*, 422-431, 1988.
16. D. B. Shmoys, "Computing Near-Optimal Solutions to Combinatorial Optimization Problems," in *Combinatorial Optimization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science (20), W. Cook and L. Lovász (eds.), AMS Publications, 1995, 355-397.
17. S. Plotkin and E. Tardos, "Improved Bounds on the Max-Flow Min-Cut Ratio for Multicommodity Flows," *Proc.* 25^{th} *Annual ACM Symp. on Theory of Computing*, 691-697, 1993.
18. C. H. Papadimitriou and M. Yannakakis. "Optimization, Approximation, and Complexity Classes," *Journal of Computer and System Sciences*, 43:425-440, 1991.
19. N. Robertson and P. Seymour, "Graph Minor II. Algorithmic Aspects of Tree-Width," *Journal of Algorithms*, 7, 309-322, 1986.
20. E. Tardos and V. V. Vazirani, "Improved Bounds for the Max-Flow Min-Multicut Ratio for Planar and $K_{r,r}$-free graphs," *Information Processing Letters*, 47 (2), 77-80, 1993.
21. J. van Leeuwen, "Graph Algorithms," *Handbook of Theoretical Computer Science*, Volume A, chapter 10, 525-631, The MIT Press/Elsevier, 1990.

A Better Approximation Ratio for the Minimum k-edge-connected Spanning Subgraph Problem

Cristina G. Fernandes *

Abstract

Consider the minimum k-edge-connected spanning subgraph problem: given a positive integer k and a k-edge-connected graph G, find a k-edge-connected spanning subgraph of G with minimum number of edges. This problem is known to be NP-complete. Khuller and Raghavachari presented the first algorithm with a performance ratio smaller than 2 for all k. They proved an upper bound of 1.85 for the performance ratio of their algorithm. We improve their analysis, proving that the performance ratio of their algorithm is smaller than 1.7 for large enough k, and that it is at most 1.75 for all k. Our analysis improves the best known ratios for any fixed $k \geq 4$, in particular, for $k = 4$ from 1.75 to 1.65, and for $k = 5$ from 1.733... to 1.68. Last, we show that the minimum k-edge-connected spanning subgraph problem is MAX SNP-hard, even for $k = 2$.

1 Introduction

The study of connectivity in graph theory has important applications in the areas of network reliability and network design. In this paper, we concentrate in the minimum k-edge-connected spanning subgraph problem: given a positive integer k and a k-edge-connected graph G, find a k-edge-connected spanning subgraph of G with minimum number of edges.

This problem is known to be NP-complete [GJ79], even for $k = 2$: if the graph G is Hamiltonian, a 2-edge-connected spanning subgraph of G with minimum number of edges must be a Hamiltonian cycle. So the goal is to look for good polynomial-time approximation algorithms for the problem.

The quality of an approximation algorithm is measured by its so called *approximation* or *performance ratio*. For the minimum k-edge-connected spanning subgraph problem, the approximation ratio of an algorithm is the infimum, over all possible inputs, of the ratio between the number of edges of the output of the algorithm and the number of edges of the optimum.

For a long time, 2 was the best approximation ratio achieved for all k. Just to illustrate, let us quickly describe how to get a ratio of 2. Assume we have a c-edge-connected spanning subgraph H of G, for some $c < k$. Observe that if we add to H the edges of a maximum forest in $G - E(H)$, the resulting spanning subgraph of G is $(c+1)$-edge-connected. Using this fact, we can easily derive a procedure to construct a k-edge-connected spanning subgraph of G. Start with $S = \emptyset$, and repeatedly add to S the edges of a maximum forest in $G-S$. After k iterations, S has at most $k(n-1)$ edges, where n is the number of vertices in G, and it is the edge set of a k-edge-connected spanning subgraph of G. Now, note that any k-edge-connected spanning subgraph of G must have at least $kn/2$ edges, since each vertex must have degree at least k. Thus this procedure leads to a polynomial-time algorithm with approximation ratio at most 2. In fact, there are examples which prove that the performance ratio is exactly 2.

Khuller and Raghavachari [KR95] presented the first algorithm with a performance ratio smaller than a constant smaller than 2 for all k. They proved an upper bound of 1.85 for the performance ratio of their algorithm. In this paper, we improve their analysis, proving that the performance ratio of their algorithm is smaller than 1.7 for large enough k, and that it is at most 1.75 for all k.

Previously to [KR95], Karger [K94] presented an algorithm with performance ratio $1 + O(\sqrt{(\log n)/k})$. This is smaller than 2 only when $k \gg \log n$. Also, there were algorithms with approximation ratio smaller than 2, for some particular values of k. An algorithm for $k = 2$ with 1.5 ratio was presented in [KV94]. As observed in [KR95], by combining the biconnectivity algorithm in [KV94], and the sparse certificate algorithm in [CKT93], one can easily obtain a ratio of $2 - 1/k$.

The bound on the approximation ratio given in [KR95] for small values of k is actually better than 1.85: it is 1.5 for $k = 2$, 1.666... for $k = 3$, 1.75 for $k = 4$, 1.733... for $k = 5$, etc. To our knowledge, these were the best known. The bound in [KR95] on the performance ratio of Khuller and Raghavachari's algorithm is tight for $k = 2$ and 3. Our analysis improves their bound for any fixed $k \geq 4$. In particular, we get 1.65 for $k = 4$, and 1.68 for $k = 5$.

*College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280. E-mail: cris@cc.gatech.edu. Research supported in part by the CNPq (Brazil), under contract 200975/92-7.

Khuller and Vishkin [KV94] have introduced the following concept: a *tree-carving in a graph $G = (V, E)$* is a partition of the vertex set V into subsets $V_1, \dots, V_t$ with the following properties. Each subset constitutes a node of a tree Γ. For every vertex v in V_j, all the neighbors of v in G belong either to V_j itself or to V_i, where V_i is adjacent to V_j in the tree Γ. They have used this to prove the 1.5 bound on the ratio of their algorithm for $k = 2$. We generalize the concept of tree-carving, and from this generalization, we derive that the performance ratio of Khuller and Raghavachari's algorithm is smaller than 1.7 for large enough k.

Finally, we show that the minimum k-edge-connected spanning subgraph problem is MAX SNP-hard. This implies that there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at most $1 + \epsilon$ would imply that $P = NP$ [ALMSS92].

In the next section, we review some results proved in [KR95], the algorithm, and the analysis. In section 3, we present a better analysis and prove a new upper bound of 1.75 for all k on the performance ratio of the algorithm. Section 4 generalizes the concept of a tree-carving, and shows some properties which are then used to prove that the performance ratio of the algorithm is less than 1.7 for large enough k. The proof of MAX SNP-hardness appears in section 5. Finally, the conclusions are presented in the last section.

2 Preliminaries

Consider a graph G, a maximum depth first search forest F in G, and the post-order numbering of the vertices of G given by F. For each edge e in G, call *top (bottom)* the endpoint of e with biggest (smallest) post-order number. Recall that any edge of G not in F is a *back edge*, that is, its top endpoint is an ancestor in F of its bottom endpoint.

A *cut in a graph* $G = (V, E)$ is denoted by $(X, \overline{X})$, where X is a subset of V and $\overline{X}$ stands for $V - X$. A cut $(X, \overline{X})$ in G consists of the set of all edges of G which have one endpoint in X and the other in $\overline{X}$.

The following lemma is a restatement of lemmas 3.1, 3.2, 3.3 and 3.7 in [KR95]. It permits a better understanding of the algorithm and thus of our analysis. We omit its proof (which is not very complicated).

LEMMA 2.1. *Let k be a positive integer, $G = (V, E)$ be a k-edge-connected graph, i be an integer, $0 \leq i < k - 1$. Let S and F be two disjoint subsets of E such that (V, S) is an i-edge-connected spanning subgraph of G and (V, F) is a maximum depth first search forest in $(V, E - S)$. Then the following hold:*

1. *$(V, S \cup F)$ is $(i + 1)$-edge-connected.*
2. *Any cut in G with exactly $(i + 1)$ edges in $S \cup F$ contains exactly one edge of F.*
3. *Consider a cut in G with $(i + 1)$ edges in $S \cup F$. Let e be the edge of F in this cut. The edges of $E - S$ in this cut are exactly e plus any back edge in $E - S - F$ which has e on the unique path in F between its endpoints.*

2.1 Khuller and Raghavachari's algorithm.

The input for Khuller and Raghavachari's algorithm is a positive integer k and a k-edge-connected graph $G = (V, E)$. Their algorithm works in phases. It consists of $\lfloor k/2 \rfloor$ phases, plus an extra final phase when k is odd. For now, let us assume that k is even. Later we comment on the final phase for odd k.

After phase i, the algorithm will have selected a set of edges S_i of G which induces a $2i$-edge-connected spanning subgraph of G. Hence at the end of $k/2$ iterations, the current set $S_{k/2}$ induces a k-edge-connected spanning subgraph of G which is the output of their algorithm.

How does each phase work? Let $S_0 = \emptyset$. In phase $i \geq 1$, the algorithm chooses two disjoint edge sets F_i and B_i, both disjoint from S_{i-1}. The set F_i is simply the edge set of a maximum depth first search spanning forest in the graph $(V, E - S_{i-1})$. For each edge e in F_i, we call an edge f in $E - (S_{i-1} \cup F_i)$ a *back edge of* e if e lies on the unique path in (V, F_i) between the two endpoints of f. (Note that any edge in $E - (S_{i-1} \cup F_i)$ must have the two endpoints in the same component of F_i, because (V, F_i) is a maximum forest in $(V, E - S_{i-1})$.) The edge set B_i is built as follows: initially $B_i = \emptyset$. Then each edge e in F_i is scanned in post-order. If a minimum cut in G separating the endpoints of e contains exactly $2i - 1$ edges in $S_{i-1} \cup F_i \cup B_i$, then add to B_i a back edge of e which has the biggest post-order numbered top endpoint. (Because $k \geq 2i$, and G is k-edge-connected, there must be some edge of G not in $S_{i-1} \cup F_i \cup B_i$ which is in this minimum cut. And, by 3 of lemma 2.1, this is a back edge of e.)

Next lemma was proved in [KR95] and is basically a corollary of lemma 2.1, so we omit the proof. The correctness of the algorithm is a direct consequence of it.

LEMMA 2.2. *For all i, $0 \leq i \leq k/2$, S_i is a $2i$-edge-connected spanning subgraph of G.*

We need some extra notation for the analysis. Given an edge e in B_i, let t_e be the edge in F_i which made e to be included in B_i. Let $(X_e, \overline{X_e})$ be the cut in G containing t_e with $2i-1$ edges in $S_{i-1} \cup F_i \cup B'_i$, where B'_i is B_i just before e was included in B_i by the algorithm. Let P_e be the set of edges in $(X_e, \overline{X_e})$ which are in $E - S_{i-1}$. Denote by OPT a k-edge-connected spanning subgraph of G with minimum number of edges, and by *opt* the number of edges in OPT.

Note that, by 3 in lemma 2.1, the cut $(X_e, \overline{X_e})$ contains exactly $2(i-1)$ edges in S_{i-1}, and the rest of the edges are in P_e. Because G is k-edge-connected, G must contain at least $k - 2(i-1)$ edges in P_e, for all $e \in B_i$. Next lemma states a key fact, proved in [KR95], essential for the analysis.

LEMMA 2.3. *For any distinct edges e and f in B_i, the edge sets P_e and P_f are disjoint.*

From this lemma, we get that

$$(2.1)\quad (k-2(i-1))|B_i| \leq opt, \quad \text{for all } i, 1 \leq i \leq k/2.$$

We also have that $opt \geq nk/2$, $|F_i| \leq n$, and $|B_i| \leq n$. From all this, Khuller and Raghavachari's upper bound on the performance ratio for even k follows:

$$\begin{aligned} ratio &= \frac{\sum_{i=1}^{k/2}(|F_i| + |B_i|)}{opt} \\ &= \frac{\sum_{i=1}^{k/2}|F_i| + \sum_{i=1}^{\lceil k/4\rceil}|B_i| + \sum_{i=\lceil k/4\rceil+1}^{k/2}|B_i|}{opt} \\ &\leq \frac{\frac{k}{2}n + \lfloor\frac{k}{4}\rfloor n}{\frac{kn}{2}} + \sum_{i=1}^{\lceil k/4\rceil}\frac{1}{k-2(i-1)} \\ &= 1 + \frac{\lfloor\frac{k}{4}\rfloor}{\frac{k}{2}} + \sum_{i=1}^{\lceil k/4\rceil}\frac{1}{k-2(i-1)} \\ &\leq \frac{3}{2} + \frac{\ln 2}{2} \leq 1.85. \end{aligned}$$

This completes the review of Khuller and Raghavachari's algorithm and analysis for even k. For odd k, the algorithm runs $\lfloor k/2\rfloor$ phases, and in the end, it runs a "half" phase, computing $F_{\lceil k/2\rceil}$ only (not $B_{\lceil k/2\rceil}$). The final $S_{\lceil k/2\rceil}$ is simply $S_{\lfloor k/2\rfloor} \cup F_{\lceil k/2\rceil}$. The analysis for odd k is similar , so we omit it.

For fixed k, the above analysis gives bounds smaller than 1.85. More specifically, it gives 1.5 for $k = 2$, 1.666... for $k = 3$, 1.75 for $k = 4$ and 1.733... for $k = 5$. These bounds are tight for $k = 2$ and 3.

In figure 1, we present our tight examples for $k = 2$ and 3. Our results originated from attempts of generalizing these examples for $k > 3$, and we think they can give some intuition. There are two essentially different types of examples for $k = 2$: G_1 and G_2. Denoting by n the number of vertices in each of these graphs, we have that in both cases $opt = n$. Also, if the algorithm chooses as F_1 the dark edges, then the output of the algorithm will have $n + (n-1)/2$ edges (in both cases). As n grows, the ratio between the size of the output and *opt* approaches 1.5. Note that both these examples can be extended for large values of n. Graph G_3 is our tight example for $k = 3$ (only

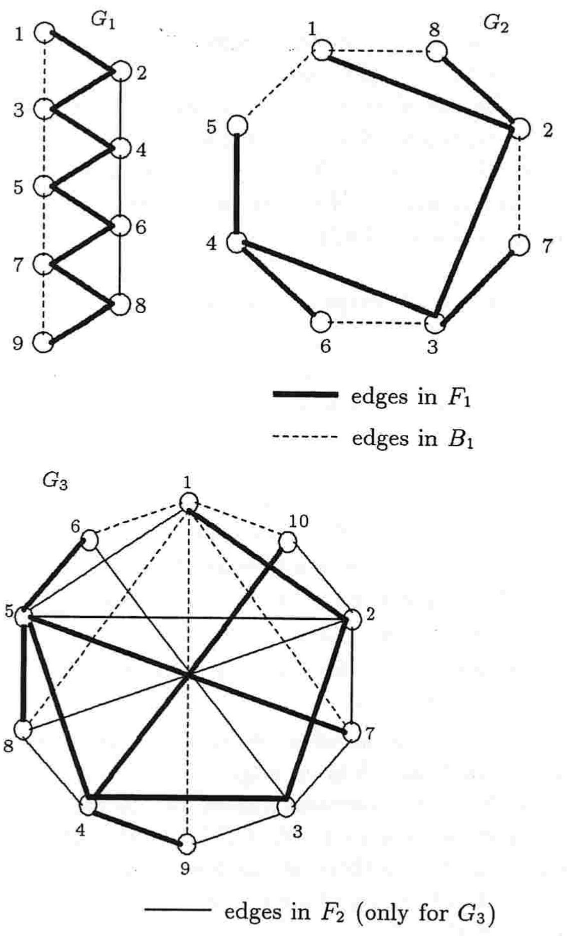


Figure 1: Graphs G_1, G_2 and G_3, with the depth first search numbering given by F_1.

for even n). In this case, OPT can be the outer cycle and the chords from each vertex in the outer cycle to the opposite vertex in the outer cycle. We have that $opt = 3n/2$. Also, assume that the algorithm chooses as F_1 the dark edges. In general, F_1 is built as in G_2: it is a path starting at any vertex in the cycle, and passing by each other vertex in the cycle, until it has visited half of the vertices. Then the remaining vertices are leaves of F_1. Note that F_1 is in fact a depth first search tree. Consider that all edges from the leaves in F_1 to the root are present in G_3, so that these edges are chosen as B_1. In our example, the dashed edges would be B_1. Besides that, the remaining edges of G_3 should contain a tree. In our example, all the remaining edges would be F_2. The output of the algorithm will have $2(n-1) + n/2$ edges. As n grows, the ratio between the size of the output and *opt* approaches $5/3 = 1.666...$ This example can be extended, as suggested, for any large value of n such that $n = 2(2t+1)$ for some positive integer t.

The intuition we would like to get of these examples comes from the following reasoning. For the previous bound to be tight, *opt* must be approximately $kn/2$, and the initial B_i's must have approximately $opt/(k-2(i-1))$ edges. But, for large k, the only way to expect that $|B_i| \approx opt/(k-2(i-1))$, for the initial B_i's, is if F_i has many leaves. However, as we will see, this cannot happen if $opt \approx kn/2$.

3 The 1.75 upper bound on the performance ratio

In this section, we improve the analysis of Khuller and Raghavachari's algorithm by presenting a better upper bound on $|B_i|$.

For each e in B_i, recall that t_e is the edge in F_i which made e to be included in B_i. Denote by b_e the bottom endpoint of t_e. For each i, $1 \leq i \leq \lfloor k/2 \rfloor$, let l_i be the number of b_e's which are leaves of F_i.

Our improvement is first based on the following stronger version of (2.1):

LEMMA 3.1. $(k-2(i-1))|B_i| + (i-1)l_i \leq \text{opt}$, *for all* i, $1 \leq i \leq \lfloor k/2 \rfloor$.

Proof. The number of pairs (u, h), where u is a vertex of G and h is an edge of OPT incident to u, is $2opt$. Next, we compute a lower bound for this number.

For each edge e in B_i, OPT has at least $k-2(i-1)$ edges in P_e. So there are at least $2(k-2(i-1))$ pairs associated to e: two for each of the $k-2(i-1)$ edges of OPT in P_e.

In fact, if b_e is a leaf of F_i, e can be associated to $(k-2(i-1))+k$ pairs. The first $k-2(i-1)$ pairs would be the ones mentioned above whose first coordinate is not b_e. The remaining k pairs have b_e as first coordinate: since OPT is k-edge-connected, there are k edges of OPT incident to b_e.

Let us show that no two edges in B_i have an associated pair in common. As said in lemma 2.3, the sets P_e's are pairwise disjoint, so the pairs coming from P_e's are certainly distinct. The k pairs corresponding to b_e cannot be repeated also, since the only edges in P_f's with b_e as endpoint are in P_e.

Therefore, we must have $l_i((k-2(i-1))+k) + (|B_i| - l_i)2(k-2(i-1)) \leq 2opt$, which is equivalent to the statement of the lemma, concluding the proof. ■

We will present a lower bound on l_i, for $1 \leq i \leq \lfloor k/2 \rfloor$, which will imply an improved upper bound on $|B_i|$. Let us fix i such that $1 \leq i \leq \lfloor k/2 \rfloor$. For such an i, $|P_e| \geq k-2(i-1) \geq 2$ for any e in B_i. Denote by B the set $\{b_e : e \in B_i\}$. The following fact will be used in the next lemma.

FACT 3.1. *For any* $e, f \in B_i$ *such that* b_f *is a child of* b_e, *there is a child of* b_f *not in* B.

Proof. Consider $e, f \in B_i$ such that b_f is a child of b_e. Because $P_e \cap P_f = \emptyset$, all the edges in P_f must have b_e as their top endpoint. Since $|P_f| \geq 2$, there is at least one edge g in $P_f - \{t_f\}$. The bottom endpoint b_g of g is not b_f (otherwise g would have the same two endpoints of f). Because b_f is a child of b_e, we must have that f was inserted in B_i before e. At the moment f was included in B_i, there could not be other edge h in B_i such that t_h is on the path in F_i between the two endpoints of f. This is true because if, by contradiction, we assume there is an h in B_i such that t_h is on the path in F_i between the two endpoints of f, we must have that $f \in P_h$. But $f \in P_f$ too, contradicting $P_f \cap P_h = \emptyset$. So, the child of b_f on the path from b_f to the bottom endpoint of g is not in B. ■

The lower bound on l_i is given by the following lemma.

LEMMA 3.2. $l_i \geq 2|B_i| - n$, *for any* i, $1 \leq i < k/2$, *and, for even* k, $l_{k/2} \geq 3|B_{k/2}| - 2n$.

Proof. Fix i such that $1 \leq i < k/2$. Denote by N the set of b_e's in B which are not leaves in F_i. We will associate to each b_e in N a distinct vertex z_e in $V - B$. The association is made in two phases:

Phase 1. For each b_e in N which has a child in $V - B$. Let z_e be a child of b_e in $V - B$.

Phase 2. For each b_e in N whose children are all in B. Let $h \in B_i$ be such that b_h is one of b_e's children. By fact 3.1, b_h has a child not in B.

Case 1: b_h has two or more children not in B. Let z_e be any child of b_h not in B and distinct of z_h.

Case 2: b_h has exactly one child not in B. Note that b_h satisfies the statement of phase 1, which means that z_h was assigned in phase 1. So this child is in fact z_h. But $i \leq (k-1)/2$, which implies that $|P_h| \geq k-2(i-1) \geq 3$. This means that there is at least one edge g in P_h which has a descendent of z_h as bottom endpoint. Let z_e be the child of z_h on the path from z_h to the bottom endpoint of g. Observe that z_e cannot be in B, otherwise the edge inserted in B_i by the edge whose bottom endpoint is z_e would forbid b_h to be in B.

Let us verify that distinct b_e's are assigned distinct z_e's. In the first phase, clearly no two distinct b_e's are assigned the same z_e. Also no two z_e's assigned in phase 2, case 1, coincide. The same for two z_e's assigned in phase 2, case 2. We explicitly make sure that a z_e assigned in phase 2, case 1, does not coincide with a

z_e assigned in phase 1. A z_e assigned in phase 2, case 2, cannot coincide with a z_e assigned in phase 1, because $z_h \notin B$. Finally, note that in phase 2, case 1, z_e is a grandchild of b_e, while in phase 2, case 2, z_e is a grandgrandchild of b_e. Thus, the only way of phase 2, case 1 and phase 2, case 2 select the same z_e is if b_h in case 2 had its z_h assigned in phase 2, case 1. But, as we have said, b_h had its z_h assigned in phase 1. This completes the proof that we can assign a different vertex in $V - B$ to each vertex in N.

Let $Z = \{z_e : e \in B_i \text{ and } b_e \in N\}$. We have that $|Z| = |N|$, and $|B| = |B_i|$. Moreover, $N \subseteq B$ and $Z \subseteq V - B$. Thus $l_i = |B - N| = |B| - |N| = |B| - |Z| \geq |B| - |V - B| = |B| - (n - |B|) = 2|B| - n = 2|B_i| - n$, completing the proof of the first part of lemma 3.2.

The proof of the second part differs from the above proof only at phase 2, case 2. For $i = k/2$ (even k), we just have $|P_h| \geq 2$. Therefore, z_h might not have a child to be selected as z_e. So, instead of associating to each b_e in N a *distinct* z_e in $V - B$, we associate a z_e which might have already been selected, but at most once. In other words, each $z \in Z = \{z_e : e \in B_i \text{ and } b_e \in N\}$ will correspond to at most two b_e's in N. More specifically, in phase 2, case 2, we select $z_e = z_h$. This implies $|N| \leq 2|Z|$, and we have $l_{k/2} = |B| - |N| \geq |B| - 2|Z| \geq |B| - 2(n - |B|) = 3|B| - 2n = 3|B_{k/2}| - 2n$, completing the proof of the lemma. ■

Now, using lemma 3.2 and $n \leq 2opt/k$ in lemma 3.1, follows a new upper bound on $|B_i|$.

COROLLARY 3.1.

$$|B_i| \leq \left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right) opt, \quad \textit{for} \quad 1 \leq i < k/2.$$

$$|B_{k/2}| \leq \frac{2(3k-4)}{k(3k-2)}\, opt.$$

This bound is not the best we can get using this technique, as we will show in the next section. But it implies the better upper bound of 1.75 on the performance ratio of the algorithm, as stated below.

THEOREM 3.1. *The performance ratio of Khuller and Raghavachari's algorithm is at most 1.75 for all k. More specifically, it is at most*

$$1.75 - \frac{3k^2 - 6k + 8}{2k^2(3k-2)}, \textit{ for even } k, \textit{ and}$$

$$1.75 - \frac{2k-3}{4k^2}, \textit{ for odd } k > 1.$$

Proof. For $k = 1$, the algorithm is optimum, i.e., the ratio is 1. For odd $k > 1$,

$$\begin{aligned}
ratio &= \frac{\sum_{i=1}^{(k+1)/2} |F_i| + \sum_{i=1}^{(k-1)/2} |B_i|}{opt} \\
&\leq \frac{\frac{k+1}{2}n}{\frac{kn}{2}} + \sum_{i=1}^{(k-1)/2} \left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right) \\
&= \left(1 + \frac{1}{k}\right) + \frac{k-1}{2k} + \frac{2}{k^2} \sum_{i=1}^{(k-1)/2} (i-1) \\
&= \frac{3}{2} + \frac{1}{2k} + \frac{(k-1)(k-3)}{4k^2} \\
&= \frac{3}{2} + \frac{k^2 - 2k + 3}{4k^2} = \frac{7}{4} - \frac{2k-3}{4k^2}.
\end{aligned}$$

The account is similar for even k.

$$\begin{aligned}
ratio &= \frac{\sum_{i=1}^{k/2} |F_i| + \sum_{i=1}^{k/2} |B_i|}{opt} \\
&\leq \frac{\frac{k}{2}n}{\frac{kn}{2}} + \sum_{i=1}^{k/2-1} \left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right) + \frac{2(3k-4)}{k(3k-2)} \\
&= 1 + \left(\frac{1}{2} - \frac{1}{k}\right) + \frac{2}{k^2} \sum_{i=1}^{k/2-1} (i-1) + \frac{2(3k-4)}{k(3k-2)} \\
&= \frac{3}{2} + \frac{k^2 - 10k + 8}{4k^2} + \frac{2(3k-4)}{k(3k-2)} \\
&= \frac{3}{2} + \frac{1}{4} - \frac{5k-4}{2k^2} + \frac{2(3k-4)}{k(3k-2)} \\
&= \frac{7}{4} - \frac{3k^2 - 6k + 8}{2k^2(3k-2)}.
\end{aligned}$$

Observe that $2k - 3 \geq 0$ for all $k \geq 3$, $4k^2 > 0$ for all positive k, $3k^2 - 6k + 8 \geq 0$ always, and $2k^2(3k-2) > 0$ for all positive k. Hence the ratio is in fact always smaller than 1.75 for all values of k. ■

4 Generalized tree-carvings

In this section, we generalize some of the results from the previous section. In particular, we present stronger versions of both lemmas 3.1 and 3.2. From these, we get an even better upper bound on the $|B_i|$'s, and, consequently, on the performance ratio of the algorithm for large values of k.

We start by introducing a generalization of the tree-carving concept and of some results given in [KV94].

Consider a positive integer k, a k-edge-connected graph $G = (V, E)$, a non-negative integer $c < k$, and a subset S of E such that (V, S) is a c-edge-connected spanning subgraph of G.

A *c-tree-carving in G with respect to S* is a partition of the vertex set V into subsets $V_1, V_2, \ldots, V_t$ with the following properties.

1. Each subset consists of a node of a rooted tree Γ. (We will refer to vertices of Γ as *nodes*, and to its edges as *arcs*.)

2. For every vertex v in V_j, all of the neighbors of v in $(V, E - S)$ belong to either V_j itself or to V_i, where V_i is adjacent to V_j in the tree Γ.
3. Each arc e in Γ defines a partition of V into two sets X_e and $\overline{X_e}$ (the deletion of arc e breaks Γ into two trees, Γ_1 and Γ_2, where V_1 belongs to Γ_1. The set X_e is defined to be the union of the sets V_y that belong to Γ_1). The cut $(X_e, \overline{X_e})$ in G must contain exactly c edges of S. Denote by P_e the set of edges in $(X_e, \overline{X_e})$ not in S.

When $c = 0$ and $S = \emptyset$, this definition is the same as the tree-carving definition.

How does this c-tree-carving relate to the algorithm? In fact, the algorithm can be modified so that, at each phase, a $2(i-1)$-tree-carving of G with respect to S_{i-1} is computed. This can be done as follows. Each time an edge f in F_i makes an edge e to be included in B_i, a new set V_j of the $2(i-1)$-tree-carving is defined. This set V_j consists of b_f (the bottom endpoint of f) and all of its descendents in F_i which do not appear in previously V_y's defined in this phase. The parent of this set V_j will be the set V_i containing the top endpoint of f. At the end of a phase, we define the final V_j, the root, to be the set of vertices remaining (i.e., which do not appear in V_y's defined previously in this phase). Observe that the tree corresponding to this $2(i-1)$-tree-carving has exactly $|B_i|$ arcs.

Consider a c-tree-carving $V_1, \ldots, V_t$ of G with respect to S. And let Γ, $(X_e, \overline{X_e})$ and P_e be as in the definition above.

FACT 4.1. *Any k-edge-connected spanning subgraph of G has at least $k - c$ edges in each P_e.*

Proof. Let H be a k-edge-connected spanning subgraph of G. For each arc e in Γ, there must be at least k edges of H in $(X_e, \overline{X_e})$. Exactly c edges in $(X_e, \overline{X_e})$ are in S. Hence, there must be at least $k - c$ edges of H in $(X_e, \overline{X_e})$ not in S, that is, in P_e. ∎

FACT 4.2. *Any edge in P_e, for $e = (V_i, V_j)$, has one endpoint in V_i and the other in V_j.*

Proof. Since $P_e \subseteq E - S$, each of its edges has endpoints either in the same V_y or in V_y's adjacent in Γ. But P_e is a subset of $(X_e, \overline{X_e})$, and either $V_y \cap X_e = \emptyset$ or $V_y \cap \overline{X_e} = \emptyset$. So each of its edges cannot have the two endpoints in the same V_y. By the definition of $(X_e, \overline{X_e})$, V_i and V_j are the only nodes in Γ which are adjacent and such that one is a subset of X_e and the other of $\overline{X_e}$. Thus, each edge in P_e must have one endpoint in V_i and the other in V_j. ∎

FACT 4.3. *For any distinct arcs e and f in Γ, the edge sets P_e and P_f are disjoint.*

Proof. The arcs e and f can have at most one common endpoint. Therefore, by fact 4.2, their sets P_e and P_f cannot intersect. ∎

Denote by l the number of leaves of Γ. The following result is an extension of lemma 3.1.

LEMMA 4.1. *Any k-edge-connected spanning subgraph of G has at least $(k-c)(t-1) + cl/2$ edges.*

Proof. Let H be a k-edge-connected spanning subgraph of G. Let us count the number of pairs (V_y, h), where h is an edge of H in $(V_y, \overline{V_y})$.

For each V_j which is not the root of Γ, let V_i be its parent in Γ, and e be the arc (V_i, V_j). Recall that $P_e \subseteq E - S$. By fact 4.1, H has at least $k - c$ edges in P_e. So there are at least $2(k-c)$ pairs associated to e: two for each of the $k - c$ edges of H in P_e (one with V_i as first coordinate, the other with V_j).

In fact, if V_j is a leaf of Γ, arc e can be associated with $(k-c)+k = 2k-c$ pairs. The $k-c$ first pairs would be the ones mentioned above whose first coordinate is V_i. The remaining k pairs have V_j as first coordinate: since H is k-edge-connected, there are k edges of H in $(V_j, \overline{V_j})$.

Let us show that no two arcs have an associated pair in common. By fact 4.3, the sets P_e's are pairwise disjoint, so the pairs coming from P_e's are certainly distinct. The k pairs corresponding to a leaf V_j cannot be repeated also since V_j is the endpoint of only one arc in Γ.

Denoting by m the number of edges in H, we can have at most $2m$ pairs. Therefore, $2(k-c)(t-1-l) + (2k-c)l \leq 2m$, which implies that $m \geq (k-c)(t-1) + cl/2$, completing the proof of the lemma. ∎

The generalization of lemma 3.2 we want to present gives a lower bound on l. But before stating the generalization, we need to prove one more fact.

FACT 4.4. *For each arc $e = (V_i, V_j)$ in Γ, $|V_i| + |V_j| \geq 2\sqrt{k-c}$.*

Proof. Consider the cut $(X_e, \overline{X_e})$ in G corresponding to e. Since G is k-edge-connected, and $(X_e, \overline{X_e})$ contains exactly c edges of S, there must be at least $k - c$ edges of $E - S$ in $(X_e, \overline{X_e})$. These edges must have one endpoint in V_i and other in V_j. The maximum number of edges with one endpoint in V_i and the other in V_j is $|V_i| \cdot |V_j|$. Thus $|V_i| \cdot |V_j| \geq k - c$. This implies that $|V_i| + |V_j| \geq 2\sqrt{k-c}$. ∎

Now we can prove the following lower bound on l, the number of leaves of the tree Γ.

LEMMA 4.2.

$$l \geq \left(1 + \frac{2}{\lceil 2\sqrt{k-c}\rceil - 2}\right)t - \frac{2}{\lceil 2\sqrt{k-c}\rceil - 2}\, n.$$

Proof. The proof of this lemma uses basically the same technique used in lemma 3.2. We will associate to each V_i, a set Z_i of vertices in V. The sets will be pairwise disjoint and will contain $\lceil 2\sqrt{k-c}\rceil/2$ vertices, if V_i is not a leaf of Γ, and one vertex, if V_i is a leaf of Γ. The association is done in two phases.

Phase 1. For each V_i which is a leaf or such that $|V_i| \geq \lceil 2\sqrt{k-c}\rceil/2$.
Let Z_i be a subset of V_i with $\lceil 2\sqrt{k-c}\rceil/2$ vertices, if V_i is not a leaf of Γ, and with one vertex, if V_i is a leaf. (Note that $|V_i| \geq 1$ always.)

Phase 2. For each V_i which is not a leaf and such that $|V_i| < \lceil 2\sqrt{k-c}\rceil/2$.
Let $e = (V_i, V_j)$ be an arc in Γ joining V_i to one of its children V_j. By fact 4.4, $|V_i| + |V_j| \geq 2\sqrt{k-c}$, which implies $|V_i| + |V_j| \geq \lceil 2\sqrt{k-c}\rceil$. Thus we must have $|V_j| \geq \lceil 2\sqrt{k-c}\rceil - |V_i| > \lceil 2\sqrt{k-c}\rceil/2$, and so V_j had its set Z_j assigned in phase 1. Choose Z_i to be a subset of $V_i \cup (V_j - Z_j)$ with $\lceil 2\sqrt{k-c}\rceil/2$ vertices. (There is such a subset since $|V_i \cup (V_j - Z_j)| \geq |V_i| + |V_j| - \lceil 2\sqrt{k-c}\rceil/2 \geq \lceil 2\sqrt{k-c}\rceil/2$.)

In phase 1, clearly the assigned sets are pairwise disjoint. It is also clear that a set assigned in phase 2 does not intersect a set assigned in phase 1. By fact 4.4, two sets assigned in phase 2 cannot be associated to adjacent sets V_y's. So they are disjoint.

Since the union of the sets Z_i's is a subset of V, we must have $(t-l)\lceil 2\sqrt{k-c}\rceil/2 + l \leq n$. This means

$$\begin{aligned} l &\geq \frac{\lceil 2\sqrt{k-c}\rceil}{\lceil 2\sqrt{k-c}\rceil - 2}t - \frac{2}{\lceil 2\sqrt{k-c}\rceil - 2}n \\ &= (1 + \frac{2}{\lceil 2\sqrt{k-c}\rceil - 2})t - \frac{2}{\lceil 2\sqrt{k-c}\rceil - 2}n, \end{aligned}$$

concluding the proof of the lemma. ■

As we have said before, from the algorithm, we can extract at each phase a $2(i-1)$-tree-carving of G with respect to S_{i-1} with $|B_i|$ arcs. Recall that l_i is the number of b_e's which are leaves of F_i. Then note that this $2(i-1)$-tree-carving has at least l_i leaves. To simplify the formulas, consider the function h defined as $h(x) = \lceil 2\sqrt{k-2(x-1)}\rceil$. Applying the lemma above to this $2(i-1)$-tree-carving, we get the following inequality:

$$(4.2) \qquad l_i \geq (1 + \frac{2}{h(i)-2})|B_i| - \frac{2}{h(i)-2}n.$$

Observe that, since $n \geq |B_i|$, we also have

$$(4.3) \qquad l_i \geq (1 + \frac{2}{h(j)-2})|B_i| - \frac{2}{h(j)-2}n,$$

for any $1 \leq i \leq j$.

Using (4.2), (4.3), and lemma 4.1 for OPT, we conclude the new upper bounds on $|B_i|$.

COROLLARY 4.1.

$$(4.4) \quad |B_i| \leq \frac{k(h(i)-2) + 4(i-1)}{k((k-(i-1))(h(i)-2) + 2(i-1))} opt,$$

and

$$(4.5) \quad |B_i| \leq \frac{k(h(j)-2) + 4(i-1)}{k((k-(i-1))(h(j)-2) + 2(i-1))} opt,$$

for any $1 \leq i \leq j$.

Using these new upper bounds, we derive our final result.

THEOREM 4.1. *The performance ratio of Khuller and Raghavachari's algorithm, for large enough k, is at most 1.7.*

Proof. We will show the proof for even k. The proof for odd k is similar.

We start by using, for some fixed r, bound (4.4) for $B_{\frac{k}{2}-r+1}, \ldots, B_{\frac{k}{2}}$. For the remaining B_i's, we use bound (4.5) with $j = \frac{k}{2} - r$. From this, we obtain that, for even k,

$$\begin{aligned} ratio &= \frac{\sum_{i=1}^{k/2}(|F_i| + |B_i|)}{opt} \\ &\leq 1 + \sum_{i=1}^{j}\frac{|B_i|}{opt} + \sum_{i=j+1}^{k/2}\frac{|B_i|}{opt} \\ &\leq 1 + \sum_{i=j+1}^{k/2}\frac{k(h(i)-2) + 4(i-1)}{k((k-(i-1))(h(i)-2) + 2(i-1))} \\ &\quad + \sum_{i=1}^{j}\frac{k(h(j)-2) + 4(i-1)}{k((k-(i-1))(h(j)-2) + 2(i-1))}. \end{aligned}$$

Note that $j+1 \leq i \leq k/2$ and $h(x)$ is decreasing for x, $j+1 \leq x \leq k/2$. Also, recall that $j = k/2 - r$. Therefore,

$$\begin{aligned} ratio &\leq 1 + \sum_{i=j+1}^{k/2}\frac{k(h(j+1)-2) + 4(k/2-1)}{k((k-(k/2-1))(h(k/2)-2) + 2j)} \\ &\quad + \sum_{i=1}^{j}\frac{k(h(j)-2) + 4(i-1)}{k((k-(i-1))(h(j)-2) + 2(i-1))} \\ &\leq 1 + r\frac{2(kh(j+1)-4)}{k(k+2+4j)} \\ &\quad + \sum_{i=1}^{j}\frac{k(h(j)-2) + 4(i-1)}{k((k-(i-1))(h(j)-2) + 2(i-1))} \\ &\leq 1 + \frac{2r(kh(j+1)-4)}{k(k+2+4j)} - \frac{4j}{k(h(j)-4)} \end{aligned}$$

$$+\frac{h(j)(h(j)-2)}{h(j)-4}\sum_{i=1}^{j}\frac{1}{(h(j)-2)k-(h(j)-4)(i-1)}$$
$$\leq 1+\frac{2r(kh(j+1)-4)}{k(k+2+4j)}-\frac{4j}{k(h(j)-4)}$$
$$+\frac{h(j)(h(j)-2)}{(h(j)-4)^2}\log\frac{k(h(j)-2)}{(h(j)-2)k-(h(j)-4)j}$$
$$\leq 1+\frac{2r(kh(j+1)-4)}{3k^2+2k-4r}-\frac{2k-4r}{k(h(j)-4)}$$
$$+\frac{h(j)(h(j)-2)}{(h(j)-4)^2}\log\frac{k(h(j)-2)}{h(j)k-2(h(j)-4)r}.$$

Since r is fixed, $h(j)=\lceil 2\sqrt{2r+2}\rceil$, $h(j+1)=\lceil 2\sqrt{2r}\rceil$ are also fixed. And, as k goes to infinity, the bound on the ratio converges to

$$1-\frac{2}{h(j)-4}+\frac{h(j)(h(j)-2)}{(h(j)-4)^2}\log\frac{2(h(j)-2)}{h(j)}.$$

Now, let us make r goes to infinity, which implies that $h(j)=\lceil 2\sqrt{2r+2}\rceil$ goes to infinity as well. The above expression converges to $1+\log 2<1.7$. From this, we finally get that, for large enough k, the performance ratio of the algorithm is at most 1.7. ■

5 The complexity of the problem

In this section, we show that the minimum k-edge-connected spanning subgraph problem is MAX SNP-hard. This means that there is a constant $\epsilon>0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at most $1+\epsilon$ implies that $P=NP$, by results of Arora et al. [ALMSS92].

As in [PY91], we use the concept of L-reduction, which is a special kind of reduction that preserves approximability. Let A and B be two optimization problems. We say *A L-reduces to B* if there are two polynomial-time algorithms f and g, and positive constants α and β, such that for each instance I of A,

1. Algorithm f produces an instance $I'=f(I)$ of B, such that the optima of I and I', of costs denoted $opt_A(I)$ and $opt_B(I')$ respectively, satisfy $opt_B(I')\leq\alpha\cdot opt_A(I)$, and
2. Given any feasible solution of I' with cost c', algorithm g produces a solution of I with cost c such that $|c-opt_A(I)|\leq\beta\cdot|c'-opt_B(I')|$.

THEOREM 5.1. *The minimum k-edge-connected spanning subgraph problem is MAX SNP-hard.*

Proof. Denote by VC_7 the vertex cover problem restricted to graphs with maximum degree seven. Papadimitriou and Yannakakis [PY91] showed that VC_7 is MAX SNP-hard. We prove that VC_7 L-reduces to the minimum k-edge-connected spanning subgraph problem, here denoted by k-MECSS. The reduction comes from [KRY95], where a directed version of the 2-MECSS is proved to be MAX SNP-hard.

The first part of the L-reduction is a polynomial-time algorithm f and a constant α. Given any instance G of VC_7, f produces an instance H of the 2-MECSS such that the minimum number of edges in a 2-edge-connected spanning subgraph of H, denoted by $opt_{k-MECSS}(2,H)$, is at most α times the minimum size of a vertex cover in G, denoted by $opt_{VC_7}(G)$. In other words, $opt_{k-MECSS}(2,H)\leq\alpha\cdot opt_{VC_7}(G)$.

Let us describe algorithm f. Consider an instance G of VC_7. G is a graph with maximum degree seven. Here is a procedure to construct an instance H of the 2-MECSS. Similarly to [KRY95], start with a special vertex, the root. Each vertex in G will have a "current vertex", initially the root. For each edge uv, add a "cover-testing gadget" to H, as illustrated in figure 2. Specifically, add six new vertices x_1,x_2,x_3,y_1,y_2,y_3. Vertex x_2 is adjacent only to vertices x_1 and x_3. Analogously, vertex y_2 is adjacent only to vertices y_1 and y_3. There is an edge labeled u^- between x_1 and y_3, and an edge labeled v^- between y_1 and x_3. There is one more edge incident to x_1: an edge labeled u^+ between x_1 and the current vertex of u; and one more edge incident to y_1: an edge labeled v^+ between y_1 and the current vertex of v. Make y_3 the new current vertex of u, and x_3 the new current vertex of v. Finally, after all edges of G have been considered, for each vertex v in G, add an edge labeled v^+ between its final current vertex and the root. The gadgets are implicitly numbered in the order we have added them. Clearly H can be obtained in polynomial time in the size of G. This completes the description of f.

Next fact will be used in the proof of the existence of a constant α. Let m be the number of edges in G, and s be a positive integer.

FACT 5.1. *If G has a vertex cover with at most s vertices, then H has a 2-edge-connected spanning subgraph with at most $6m+s$ edges.*

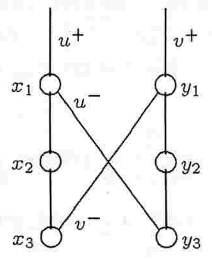


Figure 2: The gadget for edge uv.

Proof. Suppose G has a vertex cover S with at most s vertices. Consider the spanning subgraph H' of H with all edges incident to vertices of type x_2 and y_2, all edges labeled u^+ for $u \in S$, and all edges labeled u^- for $u \notin S$. Observe that all vertices in H'', but the root, have degree two. Besides, the only possible cycles in H' are of two types: either they are like $x_1x_2x_3y_1y_2y_3$, or they include the root. Because S is a vertex cover, no cycles of type $x_1x_2x_3y_1y_2y_3$ can occur. This implies that H is a collection of cycles intersecting only at the root, and spanning all vertices. Therefore, H' is a 2-edge-connected spanning subgraph of H. To count how many edges there are in H', just notice that H' has six edges per gadget plus one extra edge per vertex in S. Therefore, in total, H' has at most $6m + s$ edges. ∎

LEMMA 5.1. $opt_{k-MECSS}(2, H) \leq 43 \cdot opt_{VC_7}(G)$.

Proof. By fact 5.1, $opt_{k-MECSS}(2, H) \leq 6m + opt_{VC_7}(G)$. Also, because G has maximum degree seven, $opt_{VC_7}(G) \geq m/7$. Putting these together, we get that

$$\begin{aligned} opt_{k-MECSS}(2, H) &\leq 6m + opt_{VC_7}(G) \\ &\leq (6 \cdot 7 + 1) opt_{VC_7}(G) \\ &= 43 \cdot opt_{VC_7}(G). \blacksquare \end{aligned}$$

So, we can consider $\alpha = 43$.

The second part of the L-reduction is the constant β and algorithm g. We may assume that $m \geq 1$. Note that H has $6m + 1$ vertices, therefore any 2-edge-connected spanning subgraph of H must have at least $6m+1$ edges. Consider a 2-edge-connected spanning subgraph H' of H with $6m + s$ edges, where s is some positive integer. Algorithm g will produce in polynomial time a vertex cover of size at most s, from H'. From this, and from fact 5.1, we will have that $opt_{k-MECSS}(2, H) = 6m + opt_{VC_7}(G)$. And then $|6m + s - opt_{k-MECSS}(2, H)| = |6m + s - 6m - opt_{VC_7}(G)| = |s - opt_{VC_7}(G)|$, meaning that $\beta = 1$ suffices.

So let us see how algorithm g works. In a first phase, g produces another 2-edge-connected subgraph H'' of H with at most as many edges as H', and such that in H'' all vertices, but the root, have degree two. To get H'', each gadget is checked for vertices of degree three (all vertices in H, but the root, have degree at most three). If a vertex x has degree three in H', then the edges labeled u^+ and u^- incident to x appear in H'. Remove the edge labeled u^- incident to x and add (if not already there) the edge labeled u^+ incident to the vertex adjacent to x through the edge labeled u^-. See figure 3.

H'' is the graph obtained after applying this modification while there are vertices, other than the root, of degree three. We can make sure the modification is applied to the lowest numbered gadget, and inside the gadget, to x_1, y_1, x_3, y_3, in this order. This procedure takes polynomial time: the modification can be done in polynomial time in the size of G, and after each modification the number of edges labeled u^- decreases. Clearly H'' has at most as many edges as H'.

FACT 5.2. *H'' is a 2-edge-connected spanning subgraph of H.*

Proof. First, let us prove that all cycles in H'' contain the root. Consider a cycle C in H''. Assume, by contradiction, that C does not contain the root. This means that there are gadgets $D_1, D_2, \ldots, D_g$ such that C can be partitioned into paths $P_1, P_2, \ldots, P_g$, where P_i is a maximal (non-trivial) path using only edges of gadget D_i.

Since H' is connected and by the way H'' is constructed from H', for a gadget for an edge uv, we cannot have both edges labeled u^- and v^- in H''. This means C must contain vertices of at least two gadgets, that is, $g > 1$.

Observe that P_i must contain exactly one edge between the two edges labeled u^+ and v^+ in D_i. This is true because all vertices in the gadget have degree two in H'', and C is a cycle in H''. Besides, if the first edge of P_1 is labeled u^+, for some u, then, for all i, the first edge of P_i is labeled u^+, for some u. Analogously, if the last edge of P_1 is labeled u^+, for some u, then, for all i, the last edge of P_i is labeled u^+, for some u. The two cases are similar, so let us assume the first one holds. In this case, the last vertex of each P_i is a vertex of type x_3 or y_3.

Consider two gadgets D and D' such that there is an edge from x_3 or y_3 in D to x_1 or y_1 in D'. Then D' has a number higher than D (that is, D' was added to H after D).

We may assume without loss of generality that D_1 is the lowest numbered gadget among $D_1, D_2, \ldots, D_g$.

For $1 \leq i < g$, the last vertex of P_i, which is of type x_3 or y_3, is connected to a vertex of type x_1 or y_1 in

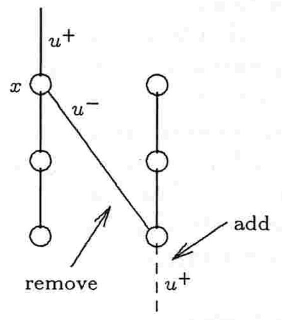


Figure 3: The modification of the gadget for edge uv.

D_{i+1}. Therefore, D_{i+1} has a number higher than D_i. From this, and because $g > 1$, we conclude that D_g has a number higher than D_1. But the last vertex of P_g, which is of type x_3 or y_3, is connected to a vertex of type x_1 or y_1 in D_1, a contradiction. This completes the proof that any cycle in H'' contains the root. And this, plus the fact that all vertices, but the root, have degree two, implies that H'' is a collection of cycles intersecting only at the root, and spanning all vertices of H. Consequently, H'' is a 2-edge-connected spanning subgraph of H. ■

Since all vertices in H'', but the root, have degree two, for each vertex in G, either all edges labeled u^+ or all edges labeled u^- appear in H''. In a second phase, a vertex cover of size at most s is computed: let S be the set of all vertices in H'' whose all edges labeled u^+ appear in H''. In any gadget, there must be edges labeled u^+ appearing in H'', otherwise H'' is not connected. Thus S is a vertex cover in G. Recall that H' had $6m + s$ edges. Hence H'' has at most $6m + s$ edges. But since all vertices but the root have degree two, H'' has $6m + t$ edges, for $t \leq s$, and t is the number of vertices in S. This finishes the description of algorithm g, completing the proof of theorem 5.1. ■

6 Conclusions

In this paper, we have investigated the minimum k-edge-connected spanning subgraph problem, which is known to be NP-complete. The goal is to look for good polynomial-time approximation algorithms. Our paper has two main contributions: first, the best known approximation ratio is reduced from 1.85 to 1.75; and second, the problem is proved to be MAX SNP-hard.

The best known ratio was 1.85 [KR95]. We present an improved analysis of Khuller and Raghavachari's algorithm [KR95], and prove an upper bound of 1.75 on its performance ratio. We also generalize the concept of a tree-carving, and use it to prove that the performance ratio of Khuller and Raghavachari's algorithm is less than 1.7 for large enough k. The MAX SNP-hardness proof guarantees that there is a small $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at most $1 + \epsilon$ implies P=NP. We believe that our analysis significantly contributes for a better understanding of the structure of the output of Khuller and Raghavachari's algorithm. We hope that this will help in the development of new algorithms with better performance ratios.

7 Acknowledgments

I would like to thank Vijay Vazirani for suggesting the problem, and S. Khuller for the main reference used on the MAX SNP-hardness proof. I am also grateful to Gruia Calinescu, Naveen Garg, Ion Mandoiu, and Robin Thomas for helpful discussions, comments and suggestions.

References

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof Verification and Hardness of Approximation Problems," *Proc. 33rd IEEE Symposium on Foundations of Computer Science*, 14–23, 1992.

[CKT93] J. Cheriyan, M. -Y. Kao and R. Thurimella, "Algorithms for Parallel k-vertex Connectivity and Sparse Certificates," *SIAM Journal of Computing*, 22 (1), 157–174, 1993.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Co., 1979.

[K94] D. Karger, "Random Sampling in Cut, Flow, and Network Design Problems," *Proc. 26th Annual ACM Symposium on the Theory of Computing*, 648–657, 1994.

[KR95] S. Khuller and B. Raghavachari, "Improved Approximation Algorithms for Uniform Connectivity Problems," *Proc. 27th Annual ACM Symposium on the Theory of Computing*, 1995.

[KRY95] S. Khuller, B. Raghavachari and N. Young, "Approximating the Minimum Equivalent Digraph," *SIAM Journal of Computing*, 24 (4), 859–872, 1995.

[KV94] S. Khuller and U. Vishkin, "Biconnectivity Approximations and Graph Carvings," *J. Assoc. Comput. Mach.*, 41 (2), 214–235, 1994.

[PY91] C. H. Papadimitriou and M. Yannakakis. "Optimization, Approximation, and Complexity Classes," *Journal of Computer and System Sciences*, 43:425–440, 1991.

ACADEMIC PRESS

Available at www.MATHEMATICSWEB.ORG
POWERED BY SCIENCE @ DIRECT•

Journal of Algorithms 48 (2003) 333–359

Journal of Algorithms

www.elsevier.com/locate/jalgor

Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width [☆]

Gruia Călinescu,[a,*,1] Cristina G. Fernandes,[b,2] and Bruce Reed[a,c,3]

[a] *Department of Computer Science, Illinois Institute of Technology, Stuart Building, Room 236, 10 West 31st Street, Chicago, IL 60616, USA*
[b] *Department of Computer Science, University of São Paulo, Brazil*
[c] *CNRS, Paris, France*

Received 27 June 2001

Abstract

The Multicut problem can be defined as: given a graph G and a collection of pairs of distinct vertices $\{s_i, t_i\}$ of G, find a minimum set of edges of G whose removal disconnects each s_i from the corresponding t_i. Multicut is known to be NP-hard and Max SNP-hard even when the input graph is restricted to being a tree. The main result of the paper is a polynomial-time approximation scheme (PTAS) for Multicut in unweighted graphs with bounded degree and bounded tree-width. That is, for any $\epsilon > 0$, we present a polynomial-time $(1+\epsilon)$-approximation algorithm. In the particular case when the input is a bounded-degree tree, we have a linear-time implementation of the algorithm. We also provide some hardness results: we prove that Multicut is still NP-hard for binary trees and that it is Max SNP-hard if we drop any of the three conditions (unweighted, bounded-degree, bounded tree-width). Finally we show that some of these results extend to the vertex version of Multicut and to a directed version of Multicut.

Keywords: Approximation algorithms; Multicut; Polynomial-time approximation schemes; Tree-width

[☆] A preliminary version of this work appeared in the proceedings of IPCO '98.

[*] Corresponding author.

E-mail addresses: calinesc@cs.iit.edu (G. Călinescu), cris@ime.usp.br (C.G. Fernandes), reed@cs.toronto.edu (B. Reed).

[1] Research partially done while at Georgia Institute of Technology and supported in part by NSF grant CCR-9319106.

[2] Research partially done while at Georgia Institute of Technology and supported in part by NSF grant CCR-9319106, FAPESP (Proc. 96/04505–2) and ProNEx 107/97 (Proc. CNPq 664107/1997-4).

[3] Research partially done at University of São Paulo (Brazil) and supported in part by FAPESP (Proc. 96/12111–4) and ProNEx (MCT/FINEP) (Proj. 107/97).

0196-6774/$ – see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/S0196-6774(03)00073-7

1. Introduction

Multicommodity Flow problems have been intensely studied for decades [6,9,12,17,18, 20] because of their practical applications and also of the appealing hardness of several of their versions. The fractional version of a Multicut problem is the dual of a Multicommodity Flow problem and, therefore, Multicut is of similar interest [3,9,10,17,24].

The *Weighted Multicut* is the following problem: given an undirected graph G, a weight function w on the edges of G, and a collection of k pairs of distinct vertices $\{s_i, t_i\}$ of G, find a minimum weight set of edges of G whose removal disconnects each s_i from the corresponding t_i.

The particular case in which $k = 1$ is characterized by the famous *Max-Flow Min-Cut Theorem* [5], and is solvable in strongly polynomial time [4]. For $k = 2$, a variant of the Max-Flow Min-Cut Theorem holds [12,13] and Multicut is solvable in polynomial time. For $k \geqslant 3$, the problem is NP-hard [3].

Since many variants of the Weighted Multicut are known to be NP-hard, we search for efficient approximation algorithms. The *performance ratio of an approximation algorithm* A for a minimization problem is the supremum, over all possible instances I, of the ratio between the weight of the output of A when running on I and the weight of an optimal solution for I. We say A is an *α-approximation algorithm* if its performance ratio is at most α. The smaller the performance ratio, the better.

The best known performance ratio for Weighted Multicut in general graphs is $O(\log k)$ [8]. Important research has been done for improving the performance ratio when the input graph G belongs to special classes of graphs. For planar graphs, Tardos and Vazirani [24] (see also [15]) give an approximate Max-Flow Min-Cut theorem and an algorithm with a constant performance ratio.

The case when the input graph is restricted to a tree has been studied by Garg et al. [10]. The Unweighted Multicut problem (in which $w(e) = 1$ for all edges e of G) restricted to stars (trees of height one) is equivalent (including performance ratio) to Minimum Vertex Cover, by Proposition 1 in [10]. It follows that Unweighted Multicut restricted to stars is NP-hard and Max SNP-hard. In fact, getting a performance ratio better than two seems very hard, since getting a performance ratio better than two for Minimum Vertex Cover remains a challenging open problem [19]. Garg et al. [10] give an algorithm with a performance ratio of two for the Weighted Multicut problem in trees. Note that the integral unweighted Multicommodity Flow problem in trees is solvable in polynomial time [10].

We find useful two variations of the Multicut problem. The *Vertex Multicut* problem is: given an undirected graph G and a collection of k pairs of distinct nonadjacent vertices $\{s_i, t_i\}$ of G called *terminals*, find a minimum set of nonterminal vertices whose removal disconnects each s_i from the corresponding t_i. The *Unrestricted Vertex Multicut* problem is: given an undirected graph G and a collection of k pairs of vertices $\{s_i, t_i\}$ of G called *terminals*, find a minimum set of vertices whose removal disconnects each s_i from the corresponding t_i. (Note that in this variation, terminals might be removed.) Observe that Vertex Multicut is at least as hard (including performance ratio) as Unrestricted Vertex Multicut. From an instance of Unrestricted Vertex Multicut we can obtain an instance of Vertex Multicut by adding, for each s_i, a new vertex s_i' adjacent only to s_i, and, for each t_i, a new vertex t_i' adjacent only to t_i. Each pair $\{s_i, t_i\}$ is substituted by the new pair

$\{s_i', t_i'\}$. Solving Vertex Multicut in this instance is equivalent to solving Unrestricted Vertex Multicut in the original instance.

Both Vertex Multicut and Unrestricted Vertex Multicut might be of interest on their own. Garg, Vazirani, and Yannakakis considered the weighted version of Vertex Multicut, and proved that their algorithm in [10] achieves a performance ratio of $O(\log k)$ for the weighted version of Vertex Multicut in general graphs.

1.1. New results

From now on, we refer to Multicut as *Edge Multicut*, to avoid confusion. Let us mention some results we obtained for Vertex Multicut and Unrestricted Vertex Multicut. We have a proof that Vertex Multicut is NP-hard in bounded-degree trees. Unrestricted Vertex Multicut is easier: it is polynomially solvable in trees, but it becomes NP-hard in bounded-degree series-parallel graphs.

The tree-width notion (first introduced by Robertson and Seymour [22,23]) seems to often capture a property of the input graph which makes hard problems easy. Various NP-hard problems, like Clique or Coloring, have a polynomial-time algorithm (linear-time in fact) if the input graph has bounded tree-width (see for example [2]). We will present the formal definition of tree-width in Section 2.

Bounded tree-width can also be used to obtain good approximation algorithms for those problems that remain NP-hard even if restricted to graphs of bounded tree-width. In our case, Unrestricted Vertex Multicut is NP-hard in graphs of tree-width at most two, since this class of graphs coincides with the series-parallel graphs (see for example [25]). A PTAS consists of, for each $\epsilon > 0$, a polynomial-time algorithm for the problem with a performance ratio of at most $1+\epsilon$. We give a straightforward PTAS for Unrestricted Vertex Multicut in graphs of bounded tree-width.

We present an approximation-ratio preserving reduction from Edge Multicut to Unrestricted Vertex Multicut. If the Edge Multicut instance graph has bounded degree and bounded tree-width, the Unrestricted Vertex Multicut instance graph obtained by the reduction has bounded tree-width. Combining the reduction with the PTAS for Unrestricted Vertex Multicut in graphs of bounded tree-width, we obtain a PTAS for Unweighted Edge Multicut in graphs with bounded degree and bounded tree-width. This is the main result of the paper. Note that, according to [7, Theorem 6.8, p. 140], an FPTAS cannot exist for this problem, unless P = NP. (An FPTAS consists of, for each $\epsilon > 0$, an algorithm for the problem with a performance ratio of at most $1+\epsilon$ and running-time polynomial in the size of the input and $1/\epsilon$.)

We also have a linear-time implementation of our PTAS for Edge Multicut in bounded-degree trees. The running time of our implementation is $O((n+k)\lceil 1/\epsilon \rceil d^{d\lceil 1/\epsilon \rceil + 2})$, where n is the number of vertices of the tree, k is the number of $\{s_i, t_i\}$ pairs, d is the maximum degree of the tree and $1+\epsilon$ is the desired approximation ratio of the algorithm. The size of the input is $\Theta(n+k)$.

Most of our results hold also for the following version of multicut in directed graphs (here called *digraphs*), introduced by Klein et al. [16]. We call *Directed Edge Multicut* the following problem: given a digraph D and a collection of k pairs of distinct vertices $\{s_i, t_i\}$ of D, find a minimum set of edges whose removal ensures that none of the strongly connected components includes one of the k pairs of vertices. In other words, for all i, either all paths from s_i to t_i *or* all paths from t_i to s_i must be broken.

Klein et al. [16] present an $O(\log^2 k)$-approximation algorithm for edge-weighted Directed Edge Multicut in general digraphs. We give a PTAS for Directed Edge Multicut in unweighted digraphs with bounded degree and bounded tree-width. It is worth mentioning that we mean here the concept of tree-width in digraphs, introduced by Johnson et al. [14], which differs from the one in undirected graphs. For example, the class of digraphs of tree-width zero consists of all acyclic digraphs.

From the hardness point of view, we show that Edge Multicut is still NP-hard for binary (degree bounded by three) trees. Thus on the class of graphs of bounded degree and bounded tree-width, which contains binary trees, Edge Multicut is easier (there is a PTAS) than on general graphs, yet still NP-hard. Identifying this class is the main theoretical result of this paper.

Also, we show that Directed Edge Multicut is NP-hard in digraphs with tree-width one and maximum in and out degree three. Other hardness results indicate why we cannot eliminate any of the three restrictions—unweighted, bounded degree and bounded tree-width—on the input graph and still obtain a PTAS. It is known [1] that for a Max SNP-hard problem, unless P = NP, no PTAS exists. We already mentioned that Unweighted Edge Multicut is Max SNP-hard in stars [10], so letting the input graph have unbounded degree makes the problem harder. We show that Weighted Edge Multicut is Max SNP-hard in binary trees, therefore letting the input graph be weighted makes the problem harder. Finally, we show that Unweighted Edge Multicut is Max SNP-hard if the input graphs are walls. Walls, to be formally defined in Section 6, have degree at most three and unbounded tree-width. We conclude that letting the input graph have unbounded tree-width makes the problem significantly harder.

In Section 2 we present the polynomial-time algorithm for Unrestricted Vertex Multicut in trees and the polynomial-time approximation scheme for Unrestricted Vertex Multicut in bounded-tree-width graphs. In Section 3, we show the approximation-preserving reduction from Edge Multicut to Unrestricted Vertex Multicut. In Section 4 we present the linear-time implementation of the polynomial-time approximation scheme for bounded-degree trees. Section 5 treats the directed case. Finally, in Section 6 we present our hardness results.

2. Algorithms for unrestricted vertex multicut

In this section we concentrate on Unrestricted Vertex Multicut. We present a polynomial-time algorithm for trees and a PTAS for graphs with bounded tree-width. Let us start defining tree-width.

Let G be a graph and Θ be a pair $(T, (X_w)_{w\in V(T)})$, which consists of a tree T and a multiset whose elements X_w, indexed by the vertices of T, are subsets of $V(G)$. For a vertex v of G, we denote by F_v the subgraph of T induced by those vertices w of T for which X_w contains v. Then Θ is called a *tree decomposition of* G if it satisfies the two conditions below:

(1) for every edge $e = xy$ of G, there is a vertex w of T such that $\{x, y\} \subseteq X_w$;
(2) for every vertex v of G, the subgraph F_v of T is a tree.

The *width of* Θ is the maximum, over all vertices w of T, of $|X_w| - 1$, and the *tree-width of* G, denoted by $tw(G)$, is the minimum of the widths of all tree decompositions of G.

Consider an instance of Unrestricted Vertex Multicut, that is, a graph $G = (V, E)$ and a set $\mathcal{C}$ of pairs $\{s_i, t_i\}$ of vertices of G. We say a pair $\{s_i, t_i\}$ in $\mathcal{C}$ is *disconnected by a set* $S \subseteq V$ if s_i or t_i is in S or s_i is disconnected from t_i in the subgraph of G induced by $V \setminus S$. A set S is a *solution for* G if S disconnects all pairs $\{s_i, t_i\}$ in $\mathcal{C}$. If S has minimum size (i.e., minimum number of vertices), then S is an *optimal solution for* G.

Now, let us describe the polynomial-time algorithm for trees. The input of the algorithm is a tree T and a set $\mathcal{C}$ of pairs $\{s_i, t_i\}$ of vertices of T.

Consider the tree T rooted at an arbitrary vertex and consider also an arbitrary ordering of the children of each vertex (so that we can talk about postorder). For a set S of vertices, call a pair $\{s_i, t_i\}$ in $\mathcal{C}$ *active* if it is not disconnected by S.

Algorithm UVMulticut-in-trees $(T, \mathcal{C})$.

```
$S := \emptyset$;
for each vertex $v$ in $T$ in postorder do
    if $v$ is the least common ancestor of some active pair in $\mathcal{C}$
        then $S := S \cup \{v\}$;
output $S$.
```

Clearly the following invariant holds: all nonactive pairs in $\mathcal{C}$ are disconnected by S. A pair in $\mathcal{C}$ that becomes nonactive does not go back to active since we never remove vertices from S. At the end of the algorithm, no pair in $\mathcal{C}$ is active, meaning that S is a solution for the problem. Besides, it is not hard to see that the algorithm can be implemented to run in polynomial time. The solution produced is optimal. Indeed, we associate to each vertex v included in S an active pair $\{s_i, t_i\}$ having v as least common ancestor, and the paths connecting these pairs in T are vertex disjoint. So any solution must contain a vertex in each of these vertex-disjoint paths.

2.1. Bounded-tree-width graphs

Next we present a PTAS for Unrestricted Vertex Multicut in graphs with bounded tree-width. Recall that a PTAS consists of, for each $\epsilon > 0$, a polynomial-time algorithm for the problem with a performance ratio of at most $1 + \epsilon$. Let us describe such an algorithm.

The input of our algorithm is a graph $G = (V, E)$, a tree decomposition $\Theta = (T, (X_w)_{w \in V(T)})$ of G, and a set $\mathcal{C}$ of pairs of vertices of G.

Given a subgraph G' of G, denote by $\mathcal{C}(G')$ the set of pairs in $\mathcal{C}$ whose two vertices are in G', and by $G - G'$ the subgraph of G induced by $V(G) \setminus V(G')$. For the description of the algorithm, all the instances we mention are on a subgraph G' of G and the set of pairs to be disconnected is $\mathcal{C}(G')$. So we drop $\mathcal{C}(G')$ of the notation and refer to an instance only by the graph G'. Denote by $\mathrm{opt}(G')$ the size (i.e., the number of vertices) of an optimal solution for G'.

Root the tree T (of the given tree decomposition) at a vertex r and consider an arbitrary ordering of the children of each vertex of T. For a vertex u of T, let $T(u)$ be the subtree of T rooted at u. Let $G(u)$ be the subgraph of G induced by the union of all X_w, $w \in V(T(u))$. Let $t := \lceil (tw(G)+1)/\epsilon \rceil$.

Here is a general description of the algorithm: label the vertices of T in postorder. Find the lowest labeled vertex u such that an optimal solution for $G(u)$ has at least t vertices. If there is no such vertex, let u be the root. Find an approximate solution S_u for $G(u)$ such that $|S_u| \leqslant (1+\epsilon)\mathrm{opt}(G(u))$ and $X_u \subseteq S_u$. If u is the root of T, then output S_u. Otherwise, let $G' := G - G(u)$ and let $\Theta' := (T', (X'_w)_{w \in V(T')})$ be the tree decomposition of G' where $T' := T - T(u)$ and $X'_w := X_w \setminus V(G(u))$, for all w in $V(T')$. Recursively get a solution S' for G'. Output $S := S' \cup S_u$.

Next we present a detailed description of the algorithm. It works in iterations. Iteration i starts with a subgraph G^{i-1} of G, a tree decomposition $\Theta^{i-1} = (T^{i-1}, (X_w^{i-1})_{w \in V(T^{i-1})})$ of G^{i-1} with T^{i-1} rooted at r, and a set S^{i-1} of vertices of G. Initially, $G^0 = G$, $\Theta^0 = \Theta$, $S^0 = \emptyset$, and $i = 1$. The algorithm halts when $G^{i-1} = \emptyset$. When G^{i-1} is nonempty, the algorithm starts calling a procedure $\mathrm{Get}(u, A)$, which returns a vertex u of T^{i-1} and a solution A for $G^{i-1}(u)$ such that $|A| \leqslant (1+\epsilon)\mathrm{opt}(G^{i-1}(u))$ and $X_u^{i-1} \subseteq A$. Then the algorithm starts a new iteration with $G^i = G^{i-1} - G^{i-1}(u)$, $\Theta^i = (T^i, (X_w^i)_{w \in V(T^i)})$, where $T^i = T^{i-1} - T^{i-1}(u)$ and $X_w^i = X_w^{i-1} \setminus V(G^{i-1}(u))$, for all w in $V(T^i)$, and $S^i = S^{i-1} \cup A$.

We postpone the description of $\mathrm{Get}(u, A)$ and, for now, assume that it works correctly and in polynomial time. The next lemma [23] states a property of tree decompositions that we use later.

Algorithm UVMulticut-PTAS-in-bd-tree-width-graphs $(G, \Theta, \mathcal{C})$.

```
$G^0 := G$;
$\Theta^0 := \Theta$;
$S^0 := \emptyset$;
$i := 1$;
while $G^{i-1} \neq \emptyset$ do
    $\mathrm{Get}(u_i, A^i)$;      /* $\|A^i\| \leqslant (1+\epsilon)\mathrm{opt}(G^{i-1}(u_i))$ and $X_{u_i}^{i-1} \subseteq A^i$ */
    $G^i := G^{i-1} - G^{i-1}(u_i)$;
    $T^i := T^{i-1} - T^{i-1}(u_i)$;
    for each $w$ in $V(T^i)$ do
        $X_w^i := X_w^{i-1} \setminus V(G^{i-1}(u_i))$;
    $S^i := S^{i-1} \cup A^i$;
    $i := i + 1$;
$f := i - 1$;
output $S^f$.
```

Lemma 1. *Consider a graph G and a tree decomposition $\Theta = (T, (X_w)_{w \in V(T)})$ of G. Let u, z and v be vertices of T with z on the path from u to v in T. Let x be a vertex in X_u and y be a vertex in X_v. Then any path in G from x to y contains a vertex of X_z.*

Next we prove that the output of the algorithm is in fact a solution.

Lemma 2. *S^f is a solution for G.*

Proof. Let $\{s, t\}$ be a pair in $\mathcal{C}$ and P be a path in G from s to t. We need to show that there is a vertex of P in S^f. Note that the vertex sets $V(G^{i-1}(u_i))$ define a partition of $V(G)$.

Let i be such that s is in $G^{i-1}(u_i)$ and j be such that t is in $G^{j-1}(u_j)$ and assume by symmetry that $i \leqslant j$. If $i = j$, then $\{s, t\} \in \mathcal{C}(G^{i-1}(u_i))$. Since S^f contains a solution for $G^{i-1}(u_i)$, we have that S^f must contain a vertex of P.

There is an X_u in $T^{i-1}(u_i)$ with s in X_u and an X_v in $T^{j-1}(u_j)$ with t in X_v. If $i < j$, then u_i is on the path from u to v in T^{i-1}. By Lemma 1, any path in G from s to t contains a vertex of $X_{u_i}^{i-1}$, and the algorithm insures that $X_{u_i}^{i-1} \subseteq S^f$, concluding the proof of the lemma. □

The next lemma proves that the performance ratio of the algorithm is at most $1 + \epsilon$.

Lemma 3. $|S^f| \leqslant (1 + \epsilon)\mathrm{opt}(G)$.

Proof. We have that

$$|S^f| = \sum_{i=1}^{f} |A^i| \leqslant \sum_{i=1}^{f} (1 + \epsilon)\mathrm{opt}\bigl(G^{i-1}(u_i)\bigr) = (1 + \epsilon) \sum_{i=1}^{f} \mathrm{opt}\bigl(G^{i-1}(u_i)\bigr)$$
$$\leqslant (1 + \epsilon)\mathrm{opt}(T),$$

because the subgraphs $G^{i-1}(u_i)$ are disjoint. □

Now we proceed with the description of a straightforward polynomial-time implementation of Get(u, A).

Search the vertices of the tree T^{i-1} in postorder. Stop if the vertex u being visited is either the root or is such that $\mathrm{opt}(G^{i-1}(u)) \geqslant t$. Let us show how we check whether $\mathrm{opt}(G^{i-1}(u)) \geqslant t$ in polynomial time.

If we are searching vertex u, it is because all children of u have been searched and have an optimal solution with less than t vertices. Compute an optimal solution for each child v of u. This can be done in $O(n^{t+1})$ time by brute force: check all subsets of $G(v)$ of size less than t. The time is polynomial, since $t = \lceil (tw(G) + 1)/\epsilon \rceil$ is fixed. Let s be the sum of the sizes of the solutions for the children of u.

Let us show that the optimum of $G^{i-1}(u)$ is at most $s + tw(G) + 1$. We do this by presenting a solution B for $G^{i-1}(u)$ of size at most $s + tw(G) + 1$. The set B is the union of X_u and an optimal solution for $G^{i-1}(v)$, for each child v of u. Thus $|B| \leqslant |X_u| + s \leqslant tw(G) + 1 + s$. Now we must prove that B is in fact a solution for $G^{i-1}(u)$. Let $\{s, t\}$ be a pair in $\mathcal{C}(G^{i-1}(u))$ and P be a path in $G^{i-1}(u)$ from s to t. We need to show that there is a vertex of P in B. If there is a vertex of P in X_u, then clearly B contains a vertex of P. If, on the other hand, P contains no vertex of X_u, we must have all vertices of P in the same $G^{i-1}(v)$, for some child v of u, by Lemma 1. But B contains a solution for $G^{i-1}(v)$. Therefore, B contains a vertex of P. This completes the proof that B is a solution for $G^{i-1}(u)$, and so the optimum of $G^{i-1}(u)$ is at most $s + tw(G) + 1$.

Now, let us proceed with the description of Get(u, A). If $s < t$, then $\text{opt}(G(u)) \leqslant s + tw(G) + 1 < t + tw(G) + 1$, and we can compute in polynomial time an optimal solution A' for $G(u)$ (by brute force: in $O(n^{t+tw(G)+2})$ time, which is polynomial since $t = \lceil (tw(G) + 1)/\epsilon \rceil$). If $|A'| < t$ then proceed to the next vertex in postorder. If $|A'| \geqslant t$, then we output u and the set $A := A' \cup X_u$. Note that in fact $\text{opt}(G^{i-1}(u)) = |A'| \geqslant t$, $X_u \subseteq A$, and $|A| \leqslant \text{opt}(G^{i-1}(u)) + (tw(G) + 1) \leqslant \text{opt}(G^{i-1}(u)) + t\epsilon \leqslant (1 + \epsilon)\text{opt}(G^{i-1}(u))$, as desired. On the other hand, if $s \geqslant t$, then $t \leqslant s \leqslant \text{opt}(G^{i-1}(u)) \leqslant s + tw(G) + 1 \leqslant s + t\epsilon \leqslant \text{opt}(G^{i-1}(u)) + \text{opt}(G^{i-1}(u))\epsilon = (1 + \epsilon)\text{opt}(G^{i-1}(u))$. Thus B is a solution for $G^{i-1}(u)$ of size at most $s + tw(G) + 1 \leqslant (1 + \epsilon)\text{opt}(G^{i-1}(u))$ that can be computed in polynomial time. Moreover, $X_u \subseteq B$. So in this case we output u and $A := B$. This finishes the description of Get(u, A).

3. Edge multicut

In this section we show that Edge Multicut can be reduced to Unrestricted Vertex Multicut by a reduction that preserves approximability.

The reduction has the following property. If the instance of Edge Multicut is a graph with bounded degree and bounded tree-width, then the corresponding instance of Unrestricted Vertex Multicut has bounded tree-width.

Given a graph $G = (V, E)$, the *line graph of* G is the graph whose vertex set is E and such that two of its vertices (edges of G) are adjacent if they share an endpoint in G. In other words, the line graph of G is the graph (E, L), where $L := \{ef\colon e, f \in E \text{ and } e \text{ and } f \text{ have a common endpoint}\}$.

Consider an instance of Edge Multicut, that is, a graph $G = (V, E)$ and a set $\mathcal{C}$ of pairs of distinct vertices of G. Let us describe the corresponding instance of Unrestricted Vertex Multicut. The input graph for Unrestricted Vertex Multicut is the line graph of G, denoted by G'. Now let us describe the set of pairs of vertices of G'. For each pair $\{s, t\}$ in $\mathcal{C}$, we have in $\mathcal{C}'$ all pairs $\{e, f\}$ such that e has s as endpoint and f has t as endpoint.

Clearly G' can be obtained from G in polynomial time. Note that $\mathcal{C}'$ has at most $k\Delta^2$ pairs, where $k = |\mathcal{C}|$ and Δ is the maximum degree of G. Also $\mathcal{C}'$ can be obtained from G and $\mathcal{C}$ in polynomial time.

The following theorem completes the reduction.

Theorem 4. *S is a solution for Edge Multicut in G if and only if S is a solution for Unrestricted Vertex Multicut in G'.*

Proof. Consider a solution S of Edge Multicut in G, that is, a set S of edges of G such that any pair in $\mathcal{C}$ is disconnected in $(V, E \setminus S)$. Note that $S \subseteq E(G) = V(G')$. Let us verify that the removal of S from G' disconnects all pairs in $\mathcal{C}'$. For any pair $\{e, f\}$ in $\mathcal{C}'$, there are s and t in $V(G)$ such that s is an endpoint of e, t is an endpoint of f and the pair $\{s, t\}$ is in $\mathcal{C}$. Moreover, a path P' in G' from e to f corresponds to a path P in G from s to t whose edges are a subset of the vertices in P' (which are edges of G). Since S is a solution of Edge Multicut in G, there must be an edge of P in S, which means that there is a vertex of P' in S. Hence S is a solution for Unrestricted Vertex Multicut in G'.

Conversely, let S be a solution for Unrestricted Vertex Multicut in G', that is, S is a set of edges of G whose removal from G' disconnects all pairs of vertices of G' in $\mathcal{C}'$. Let $\{s, t\}$ be a pair in $\mathcal{C}$, and P a path in G from s to t. (Recall that, by the description of Edge Multicut, $s \neq t$.) Let e be the first edge of P and f the last one. Clearly s is incident to e, and t to f. Thus $\{e, f\}$ is a pair in $\mathcal{C}'$. Corresponding to P, there is a path P' in G' from e to f containing as vertices all edges of P. Since S is a solution for Unrestricted Vertex Multicut in G' and $\{e, f\}$ is in $\mathcal{C}'$, S must contain a vertex of P'. Therefore there is an edge of P in S, which implies that S is a solution of Edge Multicut in G. □

The next lemma shows the previously mentioned property of this reduction.

Lemma 5. *If G has bounded degree and bounded tree-width, then the line graph of G has bounded tree-width.*

Proof. Denote by G' the line graph of G. Let us present a tree decomposition of G' whose tree-width is at most $(tw(G) + 1)\Delta$, where Δ is the maximum degree of G.

Let $\Theta := (T, (X_u)_{u \in V(T)})$ be a tree decomposition of G of width $tw(G)$. For each u in $V(T)$, let Y_u be the set of edges of G incident to some vertex in X_u. First let us prove that $\Theta' = (T, (Y_u)_{u \in V(T)})$ is a tree decomposition of G'. For this, given an edge e of G, denote by T_e the subgraph of T induced by those vertices in T for which Y_u contains e. We shall prove that (1) any edge h of G' has both endpoints in Y_u, for some u in $V(T)$; and (2) that T_e is a tree for any edge e of G'.

The endpoints of an edge h of G' are two edges e and f of G with a common endpoint, say, v. But $v \in X_u$ for some u in $V(T)$. This implies that both e and f belong to Y_u, proving (1). For (2), let e be a vertex of G', that is, an edge $e = xy$ of G. For any u such that $e \in Y_u$, we must have that either $x \in X_u$ or $y \in X_u$. Therefore $T_e = T_x \cup T_y$. We know that the subgraphs T_x and T_y of T are subtrees of T. Moreover, T_x and T_y have a vertex in common, because both x and y belong to the same X_u, for some u in $V(T)$. Hence T_e is a subtree of T. This completes the proof that Θ' is a tree decomposition of G'.

To verify that the width of Θ' is at most $(tw(G) + 1)\Delta$, just note that $|Y_u| \leqslant |X_u|\Delta$, for all u in $V(T)$. □

The next corollary is a consequence of the previous reduction and the PTAS given in Section 2.1.

Corollary 6. *There is a PTAS for Edge Multicut in bounded-degree graphs with bounded tree-width.*

4. Linear-time PTAS for edge multicut in bounded-degree trees

The PTAS for Unrestricted Vertex Multicut in bounded-tree-width graphs can be modified into a PTAS for Edge Multicut in bounded-degree trees in a straightforward way. In this section we describe this modified PTAS and present a linear-time implementation for it.

Let $T := (V, E)$ be a tree and $\mathcal{C}$ be a set of pairs $\{s_i, t_i\}$ of distinct vertices of T. We say a pair $\{s_i, t_i\}$ in $\mathcal{C}$ is *disconnected by a set* $S \subseteq E$ if s_i is disconnected from t_i in $(V, E \setminus S)$. A set S is a *solution for* T if S disconnects all pairs $\{s_i, t_i\}$ in $\mathcal{C}$. If S has minimum size (i.e., minimum number of edges), then S is an *optimal solution for* T.

Now, for each fixed $\epsilon > 0$, let us describe the polynomial-time $(1 + \epsilon)$-approximation algorithm for bounded-degree trees. The input of the algorithm is a bounded-degree tree T and a set $\mathcal{C}$ of pairs $\{s_i, t_i\}$ of vertices of T. Consider the tree T rooted at an arbitrary vertex r and consider also an arbitrary ordering of the children of each vertex (so that we can talk about postorder and preorder).

As before, given a subtree T' of T, denote by $\mathcal{C}(T')$ the set of pairs in $\mathcal{C}$ whose two vertices are in T'. For the description of the algorithm, all the instances we mention are on a tree T' and the set of pairs to be disconnected is $\mathcal{C}(T')$, where T' is a subtree of T. So we drop $\mathcal{C}(T')$ of the notation and refer to an instance only by the tree T'. Denote by $\text{opt}(T')$ the size (i.e., the number of edges) of an optimal solution for T'. For a vertex u of T, let $T(u)$ be the subtree of T rooted at u. Denote by $T - T(u)$ the subtree of T obtained from T after removing all vertices in $T(u)$. Let $t := \lceil 1/\epsilon \rceil$.

The algorithm is similar to the one for Unrestricted Vertex Multicut. It works in iterations. Iteration i starts with a subtree T^{i-1} of T also rooted at r, and a set S^{i-1} of edges of T. Initially, $T^0 := T$, $S^0 := \emptyset$, and $i = 1$. The algorithm halts when $T^{i-1} = \emptyset$. When T^{i-1} is nonempty, the algorithm starts calling a procedure $\text{Get}(u, A)$, which returns a vertex u and an optimal solution A for $T^{i-1}(u)$ where either $u = r$ or $\text{opt}(T^{i-1}(u)) \geqslant t$. Unless u is the root (in which case the algorithm outputs $S^{i-1} \cup A$ and terminates) iteration i starts with $T^i := T^{i-1} - T^{i-1}(u)$ and $S^i := S^{i-1} \cup A \cup \{uv\}$, where v is the parent of u.

The proof that this algorithm is a polynomial-time $(1 + \epsilon)$-approximation is analogous to the proof presented for the PTAS for Unrestricted Vertex Multicut in bounded-tree-width graphs.

In the following we describe a $O((n + k)\lceil 1/\epsilon \rceil d^{d\lceil 1/\epsilon \rceil + 2})$ implementation of the algorithm described above. Recall that n is the number of vertices of the tree, k is the number of $\{s_i, t_i\}$ pairs, d is the maximum degree of the tree and $1 + \epsilon$ is the desired approximation ratio of the algorithm. The size of the input is $\Theta(n + k)$.

As in the straightforward implementation of $\text{Get}(u, A)$ for Unrestricted Vertex Multicut, the procedure $\text{Get}(u, A)$ finds the first qualified u in postorder. By qualified we mean that either $u = r$ or $\text{opt}(T^{i-1}(u)) \geqslant t = \lceil 1/\epsilon \rceil$. Also, while traversing T^{i-1} in postorder, we compute $\text{opt}(T^{i-1}(u))$ for each encountered vertex u.

The main idea to obtain this linear-time implementation is to compute $\text{opt}(T^{i-1}(u))$ faster. Since we compute these values in postorder, it is natural to use the values computed for the children of u. In fact, we compute a set of "relevant" optimal solutions for $T(u)$ (the technical definition of relevant appears in the next subsection) using the sets of relevant optimal solutions for the children of u. Note that the main algorithm works no matter what optimal solution Get(u, A) returns. So, in this linear-time implementation, Get(u, A) returns only relevant optimal solutions. We will prove that the set of relevant optimal solutions is nonempty and show how to compute it for $T(u)$ using the sets of relevant optimal solutions for the children of u. We also show that the number of relevant optimal solutions is small. One can use this idea alone to obtain an $O(kn^3 d^{d\lceil 1/\epsilon\rceil+1})$-time implementation. This time is better than the time for the straightforward implementation since the degree of the polynomial in $n+k$ is fixed, that is, does not depend on ϵ or d. To further decrease the time dependence on $n+k$ we will show how to avoid recomputing optimum values and relevant optimal solutions when we switch from T^{i-1} to T^i and how to check faster if a set of edges is a solution.

4.1. Relevant solutions

We start with some definitions. Let T and C be an instance of the Multicut problem. Let v be a vertex of T and A a subset of edges of $T(v)$. We say that a pair $\{s, t\}$ in C is *open with respect to* v *and* A if exactly one of s and t is in $T(v)$ and the unique path in T from s to t does not contain any edge of A. We denote by $O_v(A)$ (T and C will be understood from the context) the set of open pairs with respect to v and A. Let v be a vertex of T. We say that an edge set B is a *relevant optimal solution for* $T(v)$ if, for any vertex u of $T(v)$, $B \cap E(T(u))$ is an optimal solution for $T(u)$. It is not obvious that a relevant optimal solution exists. In fact, this follows from the following lemma.

Lemma 7. *Let B be an optimal solution for $T(v)$. Then there is a relevant optimal solution A for $T(v)$ such that $O_v(A) \subseteq O_v(B)$.*

Proof. By induction on the number of vertices in $T(v)$. If $T(v)$ has only one vertex (vertex v) then $B = \emptyset$ and we can take $A := \emptyset$.

Suppose now that $T(v)$ has more than one vertex and let $u_1, u_2, \ldots, u_r$ be the children of v. Let $B_j = B \cap E(T(u_j))$, for $j = 1, 2, \ldots, r$. We construct $A = \bigcup_{j=1}^{r} A_j$ in the following way: if $|B_j| = \text{opt}(T(u_j))$, by induction, we let A'_j be a relevant optimal solution for $T(u_j)$ such that $O_{u_j}(A'_j) \subseteq O_{u_j}(B_j)$. If also $u_j v \in B$, we let $A_j := A'_j \cup \{u_j v\}$, otherwise $A_j = A'_j$. If $|B_j| > \text{opt}(T(u_j))$, let A'_j be any relevant optimal solution for $T(u_j)$. Such a relevant optimal solution exists by induction. Let $A_j := A'_j \cup \{u_j v\}$.

We now show that A is a solution for $T(v)$. Let $\{s, t\}$ be a pair with both s and t in $T(v)$. A first case is when there is j in $\{1, 2, \ldots, r\}$ such that both s and t are in $T(u_j)$. Then, since A'_j is a solution for $T(u_j)$, s and t are disconnected by A.

A second case is when s is in $T(u_j)$ and $t = v$. The case when t is in $T(u_j)$ and $s = v$ is symmetric. If $u_j v \in A_j$, then s and t are disconnected by A. The only case in which $u_j v \notin A_j$ is when $u_j v \notin B$ and $|B_j| = \text{opt}(T(u_j))$. Since B is a solution for $T(v)$ and $u_j v \notin B$, there is an edge in B_j disconnecting s from u_j. This means that the pair $\{s, t\}$ is not in $O_{u_j}(B_j)$, and since we selected A'_j such that $O_{u_j}(A'_j) \subseteq O_{u_j}(B_j)$, $\{s, t\}$ is not in $O_{u_j}(A'_j)$. Thus there is an edge in A'_j disconnecting s from u_j.

The third and last case is when s is in $T(u_j)$ and t is in $T(u_l)$, for some $l \neq j$. In this case there is an edge in B disconnecting s from v or there is an edge in B disconnecting t

from v. We may assume without loss of generality that there is an edge in B disconnecting s from v. Using the same reasoning as in the second case, it follows that, because there is an edge in B disconnecting s from v, there is an edge in A disconnecting s from v. In conclusion, there is an edge in A disconnecting s from t.

We note that $|A| \leqslant |B|$. Since B is an optimal solution, we have $|A| = |B|$. We now show that A is a relevant optimal solution. Let u be a vertex in $T(v)$. If $u = v$, $A \cap E(T(v)) = A$ is an optimal solution for $T(v)$. If $u \neq v$, let j in $\{1, 2, \ldots, r\}$ be such that u is a vertex in $T(u_j)$. Then $A \cap E(T(u)) = A'_j \cap E(T(u))$ is an optimal solution for $T(u)$ since A'_j is a relevant optimal solution for $T(u_j)$.

Finally we show that $O_v(A) \subseteq O_v(B)$. Let $\{s, t\}$ be a pair in $O_v(A)$ and assume without loss of generality that s is a vertex in $T(v)$ and t is not a vertex in $T(v)$. If $s = v$, then, since $B \subseteq E(T(v))$, the pair $\{s, t\}$ is in $O_v(B)$. If $s \neq v$, let j in $\{1, 2, \ldots, r\}$ be such that s is a vertex in $T(u_j)$. Then $\{s, t\}$ is a pair in $O_{u_j}(A'_j)$ (because $A'_j = A \cap E(T(u_j))$). Since $\{s, t\}$ is a pair in $O_v(A)$, $u_j v \notin A$ and therefore $u_j v \notin A_j$. The only case in which $u_j v \notin A_j$ is when $u_j v \notin B$ and $|B_j| = \text{opt}(T(u_j))$. In this case, by the construction of A, $O_{u_j}(A'_j) \subseteq O_{u_j}(B_j)$ and therefore the pair $\{s, t\}$ is in $O_{u_j}(B_j)$. Using the fact that $u_j v \notin B$, we obtain that the pair $\{s, t\}$ is in $O_v(B)$. □

In the following we describe a *generating procedure* which computes all relevant optimal solutions for $T(v)$. If v is a leaf, then we have exactly one relevant optimal solution, namely the empty set of edges. Otherwise, let $u_1, u_2, \ldots, u_r$ be the children of v. We assume that for all j, we have all the relevant optimal solutions for $T(u_j)$. The procedure is as follows: start with $l = 0$. Repeat the following process: try the union of all combinations of l edges incident to v and relevant optimal solutions for $T(u_1), T(u_2), \ldots, T(u_r)$. If some combination is a solution for $T(v)$, then write it down as a relevant optimal solution for $T(v)$. Obtain all such combinations for this same value of l and then halt. If none of the combinations leads to a solution for $T(v)$, we increment l and resume the process.

We now show that the above procedure correctly computes all relevant optimal solutions for $T(v)$. Any relevant optimal solutions for $T(v)$, if restricted to $T(u_j)$, is a relevant optimal solution for $T(u_j)$. Let A be any relevant optimal solution for $T(v)$. So A consists of a combination of relevant optimal solutions for $T(u_j)$, $j = 1, 2, \ldots, r$, and a set of q edges incident to v. All relevant optimal solutions have the same number of edges, and since any two have exactly the same number of edges in $E(T(u_j))$, for $j = 1, 2, \ldots, r$, it follows that all relevant optimal solutions have the same number of edges, q, incident to v. Also, for $l < q$, there cannot be any edge set B, which is the union of a combination of relevant optimal solutions for $T(u_1), T(u_2), \ldots, T(u_r)$ and l edges incident to v, and such that B is a solution for $T(v)$, since in this case $|B| < |A|$ and therefore A would not be an optimal solution. The correctness of the procedure above is now clear, as for $l < q$ the procedure will try all combinations but will not find any solution, and then, for $l = q$, it will output all relevant optimal solutions with q edges incident to v, and since all relevant optimal solutions have exactly q edges incident to v, it will output all relevant optimal solutions for $T(v)$.

From the procedure above, it follows by induction on the number of vertices in $T(v)$ that there are at most $d^{\text{opt}(T(v))}$ relevant optimal solutions for $T(v)$. Indeed, if v is a leaf, there is only one relevant optimal solution for $T(v)$. If v is not a leaf, using the notation from the paragraph above, $\text{opt}(T(v)) = q + \text{opt}(T(u_1)) + \text{opt}(T(u_2)) + \cdots + \text{opt}(T(u_r))$. By induction, there are at most $d^{\mathrm{opt}(T(u_j))}$ relevant optimal solutions for $T(u_j)$. In conclusion, we need to try at most $\binom{r}{q}\prod_{j=1}^{r}\mathrm{opt}(T(u_j))$ combinations and, as $r \leqslant d$, this number is at most $d^q\prod_{j=1}^{r} d^{\mathrm{opt}(T(u_j))} = d^{\mathrm{opt}(T(v))}$.

We are left with the implementation details on how to avoid recomputing values and relevant optimal solutions when we switch from T^{i-1} to T^i and how to check efficiently if a combination is a solution.

4.2. Implementation details

We represent the current tree T^i in the following way: we mark all vertices of T not in T^i. So when executing $T^i \leftarrow T^{i-1} - T^{i-1}(u_i)$ in the algorithm, in fact we traverse $T^{i-1}(u_i)$ in preorder and mark each vertex. To traverse in preorder $T^{i-1}(u_i)$, we in fact traverse in preorder $T(u_i)$, but we stop and return immediately each time we meet an already marked vertex. The time for the traversal (and marking) of $T^{i-1}(u_i)$ is proportional to the number of vertices we mark plus the number of already marked vertices we encounter. Note that whenever we encounter an already marked vertex v during the traversal of $T^{i-1}(u_i)$, its parent u was not marked before the traversal of $T^{i-1}(u_i)$, but u is marked at the end of the traversal of $T^{i-1}(u_i)$. So the total number of times, over all i, we encounter an already marked vertex is $O(n)$. Since we mark a vertex exactly once, the total time, over all i, for these marking traversals is $O(n)$.

To compute the relevant optimal solutions for vertex v in T^{i-1}, we simply ignore the children of v which are marked. Note that if, for a vertex v in T^{i-1}, the set of relevant optimal solutions is computed, and v is in T^i, then, in T^i, v has exactly the same set of relevant optimal solutions. Indeed, if v's set of relevant optimal solutions is computed in T^{i-1}, none of its descendant in T^{i-1} has optimum value at least t. In conclusion all descendants of v in T^{i-1}, including v itself, appear in T^i and therefore $T^{i-1}(v) = T^i(v)$. Since the set of relevant optimal solutions depends only on $T^{i-1}(v) = T^i(v)$, it follows that, in T^i, v has exactly the same set of relevant optimal solutions. Also note that $\mathrm{opt}(T^{i-1}(v)) < t$, since v appears in T^i. In conclusion it is unnecessary to recompute an already computed set of relevant optimal solutions.

Note that, since T^i is a subtree of T, postorder in T^i equals postorder in T restricted to the vertices of T^i. It remains to show how to find quickly the next vertex of T^i in postorder whose set of relevant optimal solutions is not computed. The first such vertex of T^i (unless of course $T^i = \emptyset$) is the successor in postorder in T of u_i, and in general the next such vertex of T^i is actually the next vertex of T in postorder. This fact—that the next vertex of T^i in postorder, whose set of relevant optimal solutions is not computed, is simply the next vertex of T in postorder—follows from and is used to maintain the following invariant: the vertices of T for which the set of relevant optimal solutions (in some T^i) is computed is always the set of first j vertices of T in postorder, for some $0 \leqslant j \leqslant n$.

4.3. Deciding feasibility

In this last subsection of Section 4 we describe how to check if a combination is a solution, thus finishing the description of the linear-time PTAS for Edge Multicut in bounded-degree trees. In a preprocessing phase, in $O(n)$ [11], construct a data structure which allows us to find in constant time the lowest common ancestor (lca) of any two vertices u and v in T. Using this data structure, in time $O(n+k)$, construct for each vertex v, the list of $\{s,t\}$ pairs such that $v = \mathrm{lca}(s,t)$. Let k_v be the size of this list.

So, as above, let v be the vertex whose set of relevant optimal solutions we are now computing in T^i. Let $u_1, u_2, \ldots, u_r$ be the children of v in T^i. To obtain these children of v in T^i, we go through the list of the children of v in T and ignore the marked ones. We need to check whether an edge set B is a solution for $T^i(v)$, where B consists of some edges incident to v and $B \cap E(T^i(u_j))$ is a relevant optimal solution for $T^i(u_j)$. In conclusion, it is enough to check if B disconnects those pairs $\{s,t\}$ with both s and t in T^i and having v as lowest common ancestor (all the other pairs are disconnected by $B \cap E(T^i(u_j))$, for $j = 1, 2, \ldots, r$). To do this, we go through the list of pairs having v as lowest common ancestor. For each pair $\{s,t\}$ in the list, we first check if both s and t are unmarked. If one of them is marked we ignore this pair. Then we check for each edge of B if that edge is on the unique path from s to v. This can be accomplished in constant time per edge of B in the following way: for an edge of B with a deeper (in T) endpoint u, we check if $\mathrm{lca}(u,s) = u$, and if so, we conclude that the edge disconnects s from v and therefore s from t. Similarly, we check for each edge of B if that edge is on the unique path from t to v. If no such edge is found, we conclude that B does not disconnect s from t. The total time spent for edge set B is $O(|B|(1+k_v))$.

Now let us prove that if the search reaches a vertex u, then $\mathrm{opt}(T^{i-1}(u)) \leqslant dt$. We prove this by presenting a solution Y for $T^{i-1}(u)$ of size at most dt. The set Y is the union of the set of edges incident to u in T^{i-1} and an optimal solution for $T^{i-1}(v)$, for each child v of u. More specifically, let Y_0 be the set of edges incident to u in T^{i-1}, and let Y_j be an optimal solution for $T^{i-1}(v_j)$, where v_j is the jth child of u. Because T^{i-1} has maximum degree at most d, $|Y_0| \leqslant d$ and u has at most d children. All of the at most d children of u were visited before u (because of the postorder). So for any child v_j of u, $|Y_j| = \mathrm{opt}(T^{i-1}(v_j)) \leqslant t-1$ (otherwise the search would have stopped before it reaches u). Thus the set

$$Y = \bigcup_{i=0}^{|Y_0|} Y_j$$

has size

$$|Y| = \sum_{i=0}^{|Y_0|} |Y_j| = |Y_0| + \sum_{i=1}^{|Y_0|} |Y_j| \leqslant d + d(t-1) = dt.$$

Therefore $|B| \leqslant dt$, and there are at most d^{dt} combinations we need to check for each l from the generating procedure described on page 344. Thus, the total time spent computing the set of relevant optimal solutions for v in T^i is $O((1+k_v)dtd^{dt+1})$. Computing these sets of relevant optimal solutions dominates the time, so the total time of the algorithm is $O((n+k)dtd^{dt+1})$, since $\sum_v (1+k_v) = n+k$. Since $t = \lceil 1/\epsilon \rceil$, we obtain that the total time of the algorithm is $O((n+k)\lceil 1/\epsilon \rceil d^{d\lceil 1/\epsilon \rceil + 2})$, which is linear in the size of the problem for any fixed $\epsilon > 0$.

5. Multicuts in digraphs

Using the same techniques we applied for undirected graphs, we can get a PTAS for Directed Edge Multicut in digraphs with bounded degree and bounded tree-width. Johnson et al. [14] presented the following definition of tree-width in digraphs.

An *arborescence* is a digraph R such that R has a vertex r_0, called the root of R, with the property that for every vertex r in $V(R)$ there is a unique directed path from r_0 to r. Thus every arborescence arises from a tree by selecting a root and directing all edges away from the root. If $r, r' \in V(R)$, we write $r' > r$ if $r' \neq r$ and there exists a directed path in R with initial vertex r and terminal vertex r'. If $e \in E(R)$, we write $r' > e$ if either $r' = r$ or $r' > r$, where r is the head of e. We also write $e \sim r$ when e is incident with r.

Let D be a digraph and let $Z \subseteq V(D)$. The digraph obtained from D by deleting Z will be denoted by $D - Z$. We say that a set $S \subseteq V(D)$ is Z-*normal* if there is no directed walk in $D - Z$ with first and last vertices in S that uses a vertex of $D - (Z \cup S)$. In other words, any directed path in $D - Z$ which starts and ends in S is entirely contained in S.

An *arboreal decomposition of a digraph* D is a triple (R, X, W), where R is an arborescence, and $X = (X_e \subseteq V(D)\colon e \in E(R))$ and $W = (W_r \subseteq V(D)\colon r \in V(R))$ satisfy

(1) $(W_r\colon r \in V(R))$ is a partition of $V(D)$, and
(2) if $e \in E(R)$, then $\bigcup\{W_r\colon r \in V(R), r > e\}$ is X_e-normal.

The *width of* (R, X, W) is the least integer w such that $|(\bigcup_{e\sim r} X_e) \cup W_r| \leqslant w + 1$ for all r in $V(R)$. The *tree-width of* D, denoted by $dtw(D)$, is the least integer w such that D has an arboreal decomposition of width w. An example of a arboreal decomposition of a digraph is given in Fig. 1.

Consider the *Directed Unrestricted Vertex Multicut* problem: given a digraph D and a collection of k pairs of vertices $\{s_i, t_i\}$ of D, called *terminals*, find a minimum set of vertices whose removal ensures that none of the strongly connected components includes one of the k pairs of vertices.

As in the undirected case, we have an approximation-ratio preserving reduction from Directed Edge Multicut to Directed Unrestricted Vertex Multicut. If the digraph of the Directed Edge Multicut instance has bounded degree and bounded tree-width, the digraph of the Directed Unrestricted Vertex Multicut instance obtained by the reduction has bounded tree-width. Combining the reduction with a PTAS for Directed Unrestricted

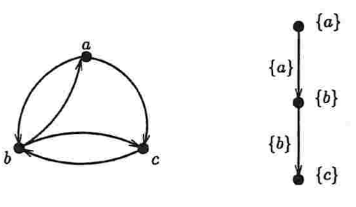


Fig. 1. On the right we have an arboreal decomposition of the digraph on the left. Next to each vertex or arc of the arborescence we find the sets W_r and X_e, which all have one element in this decomposition of width one.

Vertex Multicut in digraphs of bounded tree-width, we obtain a PTAS for Directed Unweighted Edge Multicut in digraphs with bounded degree and bounded tree-width.

The PTAS for Unrestricted Vertex Multicut in bounded-tree-width graphs can be modified into a PTAS for Directed Unrestricted Vertex Multicut in bounded-tree-width digraphs in a straightforward way. Using the same notation modified to the directed setup, the following is a short description of the modification.

Label the vertices of R in postorder. Find the lowest labeled vertex r such that an optimal solution for $D(r)$—the subdigraph of D induced by the set $Z := \bigcup\{W_{r'}\colon r' \in V(R), r' \geqslant r\}$—has at least t vertices. If there is no such vertex, let r be the root of R. Let e be the edge of R entering r (if r is the root, let e be undefined). As before, using the already computed optimal solutions for the children of r, find an approximate solution S_r for $D(r)$ such that $|S_r| \leqslant (1+\epsilon)\mathrm{opt}(D(r))$ and $X_e \cup W_r \subseteq S_r$. If r is the root of R, then output S_r. Otherwise, let $D' := D - (D(r) \cup X_e)$ and let $\Theta' := (R', X', W')$ be the arboreal decomposition of D' where $R' := R - R(r)$ and $X'_f := X_f \setminus (X_e \cup V(D(r)))$, for all f in $E(R')$, and $W'_s := W_s \setminus X_e$, for all s in $V(R')$. Recursively get a solution S' for D'. Output $S := S' \cup S_r$.

The set S produced is a solution for the Directed Unrestricted Vertex Multicut in D. Indeed if C is a directed circuit either entirely in $D(r) \cup X_e$ or entirely in D' and containing s and t for some $\{s,t\}$ in $\mathcal{C}$, then $V(C) \cap S \neq \emptyset$ (C either intersects S_r or S'). If C is a directed circuit with vertices in $D(r)$ (that is, in Z) and in D', containing s and t for some $\{s,t\}$ in $\mathcal{C}$, then C has a vertex in $X_e \subseteq S_r \subseteq S$ because Z is X_e-normal.

5.1. Reducing directed edge multicut to directed unrestricted vertex multicut

In this subsection we proceed to the reduction, which is very different from the undirected reduction. Consider an instance of Directed Edge Multicut, that is, a digraph $D = (V, E)$ and a set $\mathcal{C}$ of pairs of distinct vertices of D. Let us describe the corresponding instance of Directed Unrestricted Vertex Multicut.

The input graph for Directed Unrestricted Vertex Multicut is the following digraph D'. (To avoid confusion, we will refer to vertices of D' as nodes and to edges of D' as arcs.) The set of nodes of D' is E plus m copies $u_1, \ldots, u_m$ of each vertex u in V, where m is the number of edges in D. There is an arc in D' from a copy of u in V to e in E if u is the tail of e, and there is an arc in D' from e in E to a copy of u in V if u is the head of e. These are all the arcs in D'. Please refer to Fig. 2 for an example. Note that D' is a bipartite digraph (edges of D in one side and copies of vertices of D in the other). Clearly D' can be obtained from D in polynomial time.

Now let us describe the set of pairs of vertices of D'. For each pair $\{s,t\}$ in $\mathcal{C}$, we have in $\mathcal{C}'$ pairs $\{s_i, t_j\}$ for all $1 \leqslant i, j \leqslant m$. Note that $\mathcal{C}'$ has km^2 pairs (recall that $k = |\mathcal{C}|$). Also $\mathcal{C}'$ can be obtained from D and $\mathcal{C}$ in polynomial time.

Lemma 8. *For any solution S' for the Directed Unrestricted Vertex Multicut in D' there is a solution S in D' such that $S \subseteq E(D)$ and $|S| \leqslant |S'|$.*

Proof. We may assume S' is a minimal solution for the Directed Unrestricted Vertex Multicut in D'. If $S' \subseteq E(D)$ then we take $S := S'$. If not, let u in $V(D)$ and $1 \leqslant i \leqslant m$ be

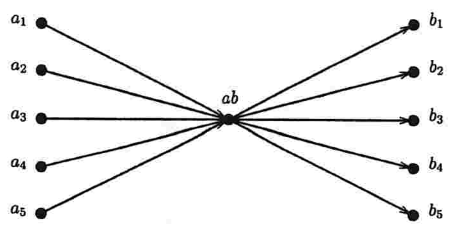


Fig. 2. The node ab of D' and its incident arcs, constructed from the directed edge ab of graph D from Fig. 1.

such that $u_i \in S'$. Then $S' \setminus \{u_i\}$ is not a solution in D', which implies that there is a pair $\{s,t\}$ in C' and a directed circuit Q in $D' - (S' \setminus \{u_i\})$ containing s and t (because s and t are in the same strongly connected component). If $u_j \notin S'$ for some $1 \leqslant j \leqslant m$, the circuit obtained from Q by substituting u_i by u_j is a directed circuit in $D' - S'$ and contains a pair in C', a contradiction. Thus $u_j \in S'$ for all $1 \leqslant j \leqslant m$. In this case, $|S'| \geqslant m = |E(D)|$ and we take $S := E(D)$. In both cases S is a solution in D' and $|S| \leqslant |S'|$, which completes the proof of the lemma. □

Lemma 8 together with the following theorem, whose proof is similar to the proof of Theorem 4, completes the reduction.

Theorem 9. *A set $S \subseteq E(D)$ is a solution for Directed Edge Multicut in D if and only if S is a solution for Directed Unrestricted Vertex Multicut in D'.*

The next lemma shows that, if the instance of Directed Edge Multicut is a digraph with bounded degree and bounded tree-width, then the corresponding instance of Directed Unrestricted Vertex Multicut has bounded tree-width. For this, we need some notation. For a vertex v in a digraph D, we denote by $\delta_D^+(v)$ the set of edges in D with v as tail. For a set S of vertices of D, $\delta_D^+(S)$ is the set of edges with tail in S and head not in S.

Lemma 10. *If D has bounded degree and bounded tree-width, then D' has bounded tree-width.*

Proof. Let us present an arboreal decomposition of D' whose tree-width is at most $(dtw(D)+1)(\Delta+1)$, where Δ is the maximum outdegree of D.

Let $\Theta := (R, X, W)$ be an arboreal decomposition of D of width $dtw(D)$. Let $\bar{r}$ be the root of R and R' be an arborescence obtained from R by substituting each edge e in $E(R)$ from a vertex r' to a vertex r in R by a directed path $P_e := \langle u_0, \ldots, u_m\rangle$ of length m, starting at $u_0 := r'$ and ending at $u_m := r$, where $u_1, \ldots, u_{m-1}$ are new vertices. Let f be the edge in $E(P_e)$ from u_{i-1} to u_i, $1 \leqslant i \leqslant m$. If $i = 1$, then let $Y_f := \delta_D^+(X_e)$, else let $Y_f := \delta_D^+(X_e \cup W_r)$. For $i = 1, \ldots, m$, let $W_r^i \subseteq V(D')$ be the set containing the ith copy in D' of each vertex in W_r. Set $Z_{u_1} := W_r^1 \cup \delta_D^+(W_r)$ and, for $i = 2, \ldots, m$, set $Z_{u_i} := W_r^i$. For an example, see Fig. 3. In addition, add to R' a path $P_{\text{root}} := \langle u_1, \ldots, u_m\rangle$, where $u_1, \ldots, u_{m-1}$ are new vertices, $u_m := \bar{r}$ and let u_1 be the root of R'. For any edge f on this path, let $Y_f := \delta_D^+(W_r)$. Set $Z_{u_1} := W_{\bar{r}}^1 \cup \delta_D^+(W_{\bar{r}})$ and, for $i = 2, \ldots, m$, set $Z_{u_i} := W_{\bar{r}}^i$. Let us prove that $\Theta' := (R', Y, Z)$ is an arboreal decomposition of D'.

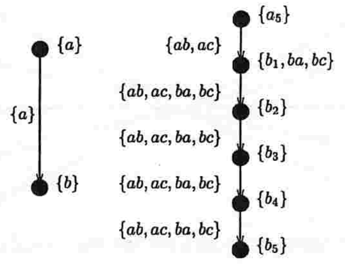


Fig. 3. From the arc $r'r$ of the arboreal decomposition of digraph D from Fig. 1, on the left of the picture (with $W_{r'} = \{a\}$, $W_r = \{b\}$, and $X_{r'r} = \{a\}$), we obtain a path of length 5 (the number of edges in D) with the sets Z (one for each vertex) given on the right of the path and the sets Y (one for each edge) given on the left of the path.

First note that $(Z_u\colon u \in V(R'))$ is a partition of $V(D')$. Now let f be an edge of R'. We need to show that $Z' := \bigcup\{Z_u\colon u \in V(R'), u > f\}$ is Y_f-normal. This property is immediate if f is in P_{root}, because any edge leaving $D' - (Z' \cup Y_f)$ is in Y_f. Let e in $E(R)$ be such that $f \in E(P_e)$. Let x, y in Z' be such that there is a directed path Q' in D' from x to y which uses a node in $V(D') \setminus Z'$. Choose x, y in Z' so that Q' has minimum length. We need to show that $V(Q') \cap Y_f \neq \emptyset$. We analyze two cases.

If $x \notin E(D)$ then let x' in $V(D')$ be the node following x in Q'. Note that $x' \in E(D) \setminus Z'$, because of the bipartition of D' and the choice of x and y. This implies that f is not the first edge of the path P_e. Thus x is a copy of a vertex in W_r and $x' \in \delta_D^+(W_r) \subseteq Y_f$.

If $x \in E(D)$ then let Q be the path in D which starts at the tail of x and visits the vertices of D in the same order in which they (their copies in fact) appear in Q'. Let $Z \subseteq V(D)$ be the set of vertices with at least one copy in Z'. Note that $Z = \bigcup\{W_r\colon r \in V(R), r > e\}$. Both endpoints of Q are in Z. Indeed, the tail of any edge of D in Z' is also in Z' (because, by construction, if $a \in E(D) \cap Z_u$ then one copy of its tail is also in Z_u) so the first vertex of Q is in Z. Also the last vertex of Q' is not in $E(D)$ (for the same reason), and this implies that the last vertex of Q is in Z. Thus Q is a path in D between two vertices of Z. Since Θ is an arboreal decomposition of D, there is a vertex of X_e in Q. But this implies Q' has a node in Y_f, because $Y_f \supseteq \delta_D^+(X_e)$. This completes the proof that Θ' is an arboreal decomposition of D'.

Now we just need to verify that the width of Θ' is at most $(dtw(D) + 1)(\Delta + 1)$. For that, let $r' \in V(R')$. If r' is a vertex of R, then $|(\bigcup_{f\sim r'} Y_f) \cup Z_{r'}| \leqslant |W_{r'}| + |\delta_D^+((\bigcup_{e\sim r'} X_e) \cup W_{r'})| \leqslant (dtw(D) + 1)(\Delta + 1)$. If r' was added when an edge e of R was substituted by a path between its endpoints, then, with r'' being the head of e, $|(\bigcup_{f\sim r'} Y_f) \cup Z_{r'}| \leqslant |W_{r''}| + |\delta_D^+(X_e \cup W_{r''})| \leqslant (dtw(D) + 1)(\Delta + 1)$. If r' is in P_{root} and $r' \neq \bar{r}$, then $|(\bigcup_{f\sim r'} Y_f) \cup Z_{r'}| \leqslant |\delta_D^+(W_r)| + |W_r| \leqslant (dtw(D) + 1)(\Delta + 1)$. $\square$

6. Complexity results

In this section, we examine the complexity of Edge, Vertex and Unrestricted Vertex Multicut in graphs and digraphs. First we prove that Edge and Vertex Multicut are NP-hard in bounded-degree trees, while Unrestricted Vertex Multicut is NP-hard in series-parallel graphs of bounded degree. Second, we show similar results for digraphs.

In Section 6.1 we show that the Weighted Edge Multicut is Max SNP-hard in binary tree and finally we prove that Edge, Vertex, and Unrestricted Vertex Multicut are Max SNP-hard in walls, defined later on, which have degree at most three and unbounded tree-width. Thus letting the input graph be weighted or have unbounded tree-width makes Edge Multicut harder.

Theorem 11. *Edge Multicut in binary trees is NP-hard.*

Proof. The reduction is from 3-SAT, a well-known NP-complete problem [7] defined as follows: given a set of *clauses*—disjunctions of three *literals* (Boolean variables or their negations)—decide if there exists a truth assignment for the Boolean variables that simultaneously satisfies all the clauses.

Consider an instance Φ of 3-SAT, that is, a set of m clauses $C_1, C_2, \dots, C_m$ on n variables $x_1, x_2, \dots, x_n$, each clause with exactly three literals. Let us construct an instance of Edge Multicut: a binary tree T and a set of pairs of distinct vertices of T.

The tree T is built as follows. For each variable x_i, there is a gadget as depicted in Fig. 4(a). The gadget consists of a binary tree with three vertices: the root and two leaves, one labeled x_i and the other labeled $\bar{x}_i$. For each clause C_j, there is a gadget as depicted in Fig. 4(b). The gadget consists of a binary tree with five vertices: the root, one internal vertex and three leaves, each one labeled by one of the literals in C_j. The tree T is built from these $n+m$ gadgets by arbitrarily connecting them using new vertices to get a binary tree. See Fig. 4(c) for an example.

Next, we give the set of pairs of vertices of T in our instance. For each variable x_i, there is a pair with the vertices labeled x_i and $\bar{x}_i$ in its gadget. For each clause C_j, there are two pairs in its gadget: one formed by the two leaves that are siblings and the other formed by the third leaf (with no sibling leaf) and its sibling internal vertex. Finally, each vertex labeled $\tilde{x}_i$ in the gadget for a clause is paired to the vertex labeled $\tilde{x}_i$ in the gadget for the variable x_i, where $\tilde{x}_i \in \{x_i, \bar{x}_i\}$. This ends the construction of the instance for Edge Multicut. Note that all this can be done in polynomial time in the size of Φ.

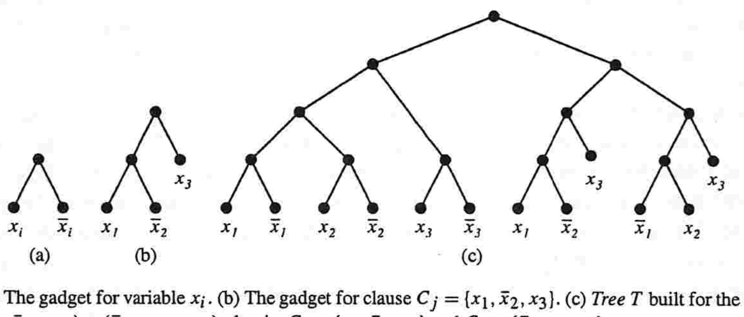


Fig. 4. (a) The gadget for variable x_i. (b) The gadget for clause $C_j = \{x_1, \bar{x}_2, x_3\}$. (c) *Tree* T built for the instance $\Phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$, that is, $C_1 = \{x_1, \bar{x}_2, x_3\}$ and $C_2 = \{\bar{x}_1, x_2, x_3\}$.

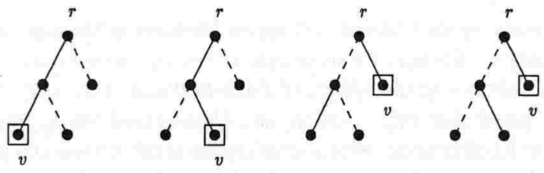


Fig. 5. Possible choices of two edges, the dashed edges, in the gadget for a clause that leave exactly one leaf (the marked leaf v) connected to the root r.

Lemma 12. *Φ is satisfiable if and only if there is a solution for T of size exactly $n + 2m$.*

Proof. Assume Φ is satisfiable and that A is a satisfying assignment. Let us present a solution S for T of size exactly $n + 2m$. The edge set S consists of two types of edges:

(1) For each variable x_i, S contains the edge in the gadget for x_i incident to the leaf labeled x_i if $x_i = \text{TRUE}$ in A or to the leaf labeled $\bar{x}_i$ if $x_i = \text{FALSE}$ in A.
(2) For each clause C_j, S contains two distinct edges in the gadget for C_j. These edges are such that (1) they disconnect the two pairs in the gadget, and (2) the only leaf that is still connected to the root of the gadget is a leaf with a label $\tilde{x}_i$ in C_j such that $\tilde{x}_i = \text{TRUE}$ in A. (The four possible choices for the two edges are shown in Fig. 5.)

Clearly such set S has exactly $n + 2m$ edges. Let us prove that S is in fact a solution for T. It is easy to see that S disconnects the pairs inside the gadget of every clause and of every variable. The remaining pairs consist of two vertices labeled by a literal $\tilde{x}_i$, one in the variable gadget for x_i and the other in a clause gadget. If $\tilde{x}_i = \text{TRUE}$ in A, then the edge in the variable gadget incident to the vertex labeled $\tilde{x}_i$ is in S, guaranteeing that the pair is disconnected. If $\tilde{x}_i = \text{FALSE}$ in A, then the vertex labeled $\tilde{x}_i$ in the clause gadget is disconnected from the root of this gadget and, therefore, from the gadget for x_i. Thus S is a solution for T, and it has exactly $n + 2m$ edges.

Let us prove the inverse implication. Assume there is a solution S for T with exactly $n + 2m$ edges: one per variable and two per clause (one for each of the "disjoint" pairs). More specifically, S has exactly two edges in each clause gadget in one of the configurations of Fig. 5. Construct a truth assignment A' by setting $x_i := \text{TRUE}$ if the edge of S in the gadget for x_i is incident to the vertex labeled x_i and by setting $x_i := \text{FALSE}$ otherwise.

For each clause C_j, there is exactly one leaf v in the gadget for C_j that is connected to the root r of the gadget. Let $\tilde{x}_i$ in $\{x_i, \bar{x}_i\}$ be the label for this leaf. There is a pair formed by this leaf v and the leaf in the gadget for x_i whose label is $\tilde{x}_i$. In S, there must be an edge e in the path between these two leaves. Since leaf v is connected to the root r of the gadget for C_j and all edges in S are either in a variable gadget or in a clause gadget, this edge e has to be in the variable gadget. This means e is the edge incident to the leaf labeled $\tilde{x}_i$ in the gadget for x_i. Hence $\tilde{x}_i = \text{TRUE}$ in A', and the clause is satisfied. Since this holds for all the clauses, the truth assignment A' satisfies Φ, implying that Φ is satisfiable. □

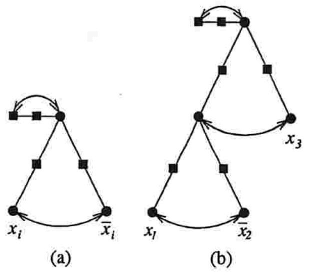


Fig. 6. (a) The new gadget for variable x_i. (b) The new gadget for clause $C_j = \{x_1, \bar{x}_2, x_3\}$. Squares indicate new vertices. Bidirected curved arcs indicate the pairs of vertices which must be disconnected inside each gadget.

Note that any solution for T must have at least $n + 2m$ edges (there are $n + 2m$ "disjoint" pairs: one in each variable gadget and two in each clause gadget). Therefore Φ is satisfiable if and only if the optimal solution for T has at most $n + 2m$ edges. This completes the proof of Theorem 11. □

Theorem 13. *Vertex Multicut in trees with maximum degree at most four is NP-hard.*

Proof. Recall that in Vertex Multicut we are not allowed to remove terminals. The reduction is also from 3-SAT, and very similar to the previous one. We just modify the variable and clause gadgets as shown in Fig. 6, (a) and (b). Each edge of the gadget is subdivided, with the new nodes taking the role of edges in the previous reduction. To prevent the vertices of the "original" gadgets (from the previous reduction) being picked in a vertex multicut solution, we make them terminals as follows: for each root v of an original gadget we add two new vertices u and w, the edges uw and wv and the pair $\{u, v\}$.

By similar arguments, the formula is satisfiable if and only if there is a solution for the constructed instance of Vertex Multicut of size at most $2n + 4m$. □

Theorem 14. *Unrestricted Vertex Multicut in series-parallel graphs with maximum degree at most three is NP-hard.*

Proof. We present a reduction from 3-SAT to Unrestricted Vertex Multicut. The reduction is a modification of the previous reduction. Consider an instance Φ of 3-SAT, that is, a set of m clauses $C_1, C_2, \ldots, C_m$ on n variables $x_1, x_2, \ldots, x_n$, each clause with exactly three literals. Let us construct an instance of the Unrestricted Vertex Multicut problem: a series-parallel graph G and a set of pairs of vertices of G.

The graph G is built as follows. For each variable x_i, there is a gadget which consists of an edge with endpoints one labeled x_i, the other $\bar{x}_i$ (see Fig. 7(a)). For each clause C_j, there is a gadget as depicted in Fig. 7(b), with three vertices labeled by the literals in C_j, as shown in the picture. The graph G is obtained from the gadgets by connecting them in parallel, adding new vertices as necessary, to get a series-parallel graph with maximum degree at most three. See Fig. 7(c) for an example. Vertices in a gadget are called *internal vertices*. Let s and t be the extremes of the series-parallel graph G, as in Fig. 7(c).

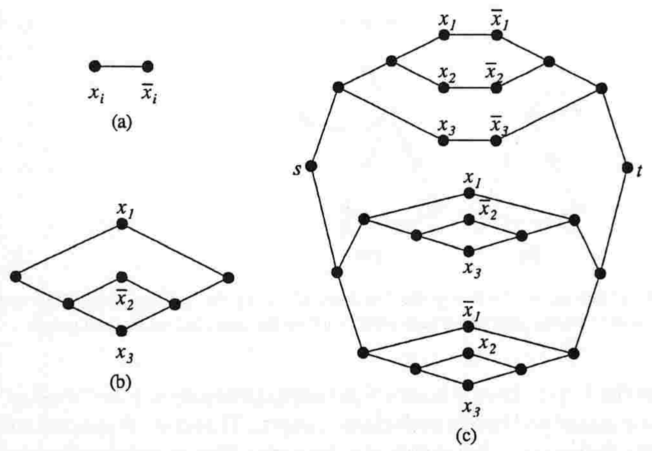


Fig. 7. (a) The gadget for variable x_i. (b) The gadget for clause $C_j = \{x_1, \bar{x}_2, x_3\}$. (c) Graph G built for the instance $\Phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$, that is, $C_1 = \{x_1, \bar{x}_2, x_3\}$ and $C_2 = \{\bar{x}_1, x_2, x_3\}$.

Next, we give the set of pairs of vertices of G in our instance. For each variable x_i, there is a pair with the vertices labeled x_i and $\bar{x}_i$ in its gadget. For each clause C_j, there are three pairs, one for each pair of literals in C_j, formed by the two vertices in the clause gadget labeled by these two literals. Finally, each vertex labeled $\tilde{x}_i$ in a clause gadget is paired to the vertex labeled $\tilde{x}_i$ in the gadget for the variable x_i, where $\tilde{x}_i \in \{x_i, \bar{x}_i\}$. This ends the construction of the instance for Unrestricted Vertex Multicut. Note that all this can be done in polynomial time in the size of Φ.

Lemma 15. *Φ is satisfiable if an only if there is a solution for G of size exactly $n + 2m$.*

Proof. Assume Φ is satisfiable and that A is a satisfying assignment. Let us present a solution S for G of size exactly $n + 2m$. The vertex set S consists of two types of vertices:

(1) For each variable x_i, S contains the vertex in the gadget for x_i labeled x_i if $x_i = \text{TRUE}$ in A or the vertex labeled $\bar{x}_i$ if $x_i = \text{FALSE}$ in A.
(2) For each clause C_j, S contains two distinct vertices in the gadget for C_j. These vertices are two of the three labeled vertices in the gadget, chosen so that the one left out has a label $\tilde{x}_i$ such that $\tilde{x}_i = \text{TRUE}$.

Clearly such set S has exactly $n + 2m$ vertices. Let us prove that S is in fact a solution for G. It is easy to see that S disconnects the pairs for the variables and the pairs for the clauses. The remaining pairs consist of two vertices labeled by a literal $\tilde{x}_i$, one in the variable gadget for x_i and the other in a clause gadget. If $\tilde{x}_i = \text{TRUE}$ in A, then the vertex labeled $\tilde{x}_i$ in the variable gadget is in S, guaranteeing that the pair is disconnected. If $\tilde{x}_i = \text{FALSE}$ in A, then the vertex labeled $\tilde{x}_i$ in the clause gadget is disconnected from s and t, and therefore, from the gadget for x_i. Thus S is a solution for G, and it has exactly $n + 2m$ vertices.

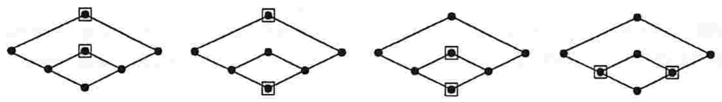


Fig. 8. Possible choices of two vertices, the vertices marked by a square, in the gadget for a clause that leave exactly one internal labeled vertex connected to the extremes s and t.

Let us prove the inverse implication. Assume there is a solution S for G with exactly $n + 2m$ vertices: one per variable and two per clause (one for each of the "disjoint" pairs). More specifically, S has exactly one vertex in each variable gadget, and exactly two vertices in each clause gadget in one of the configurations of Fig. 8. Construct a truth assignment A' by setting $x_i :=$ TRUE if the vertex of S in the gadget for x_i is labeled x_i and by setting $x_i :=$ FALSE otherwise.

For each clause C_j, there is exactly one internal labeled vertex $v \notin S$ which is connected in $G - S$ to both s and t. Let $\tilde{x}_i$ in $\{x_i, \bar{x}_i\}$ be the label of v, and let u be the vertex in the gadget for x_i such that the label of u is also $\tilde{x}_i$. In $G - (S \setminus \{u\})$, there is a path between u and either s or t. In conclusion, in $G - (S \setminus \{u\})$, there is a path between u and v. As $\{u, v\}$ is a pair which must be disconnected by S, it follows that $u \in S$. Hence $\tilde{x}_i =$ TRUE in A', and the clause is satisfied. Since this holds for all the clauses, the truth assignment A' satisfies Φ, implying that Φ is satisfiable. □

Again note that any solution for G must have at least $n + 2m$ vertices (in each variable gadget, at least one internal vertex must be removed and, in each clause gadget, at least two internal vertices must be removed, for a total of at least $n + 2m$ vertices). Therefore Φ is satisfiable if and only if the optimal solution for G has at most $n + 2m$ vertices. This completes the proof of Theorem 14. □

A *bidirected tree* is a digraph which has as underlying graph a tree and two opposite directed edges for each edge in its underlying graph. Similarly, a *bidirected series-parallel digraph* has as underlying graph a series-parallel graph, and two opposite directed edges for each edge in its underlying graph. Bidirected trees have tree-width one and bidirected series-parallel digraphs have tree-width at most two, by Proposition 2.1 in [14]. By a simple modification of the proofs shown above we can prove the following.

Theorem 16. *Directed Edge, Vertex, and Unrestricted Vertex-Multicut are NP-hard respectively in bidirected trees with maximum in and out degree three, bidirected trees with maximum in and out degree four and bidirected series-parallel digraphs with maximum in and out degree three.*

The modification consists of using, in each of the reductions, a digraph whose underlying graph is exactly the one described in the corresponding undirected case. Besides, the digraph has two opposite directed edges for each edge in its underlying graph. Note that any directed path in this digraph corresponds to a path in its underlying graph and *vice-versa*. For the edge version, we also note that any feasible solution can be converted in a feasible solution with edges directed away from the root, without increasing the size of the solution, by simply replacing any edge directed towards the root by the parallel edge directed away from the root. Thus the reductions work exactly as in the undirected case.

6.1. *Max SNP-hardness results*

The next theorem, combined with the celebrated result of Arora et al. [1], shows that a PTAS cannot be obtained for Weighted Edge Multicut in binary trees.

Theorem 17. *Weighted Edge Multicut is Max SNP-hard in binary trees.*

Proof. Recall that Unweighted Edge Multicut in stars is equivalent, including performance ratio, to the Max SNP-hard problem Minimum Vertex Cover. Let us reduce Edge Multicut in stars to Weighted Edge Multicut in binary trees. From an instance of the Unweighted Edge Multicut restricted to stars, we construct an instance of the Weighted Edge Multicut restricted to binary trees in the following way: for each leaf of the star S, there is a corresponding leaf in the binary tree T. The pairs are the same (we may assume there is no pair involving the root of the star). We connect the leaves of T arbitrarily into a binary tree, giving the edges in T incident to the leaves weight one and all other edges of T weight $2n + 1$, where n is the number of leaves in the star S (which is the same as the number of leaves in the tree T we construct). Any solution within twice the optimum for the Weighted Edge Multicut instance we constructed will contain only edges of T incident to the leaves, since any other edge is too heavy. Then it is easy to see that any optimal solution for the Weighted Edge Multicut instance we constructed corresponds to an optimal solution for the original Unweighted Multicut star instance, and *vice-versa*. Also approximability is preserved by this reduction. □

A *wall of height h* consists of $h + 1$ vertex-disjoint paths $R_0, \ldots, R_h$, which we call *rows*, and $h + 1$ vertex disjoint paths $L_0, \ldots, L_h$, which we call *columns*. A wall of height six is depicted in Fig. 9 (a). The formal definition is as follows. Each row is a path of $2h + 2$ vertices. Each column, a path with $2h + 1$ vertices. Column r contains the $(2r + 1)$st and the $(2r + 2)$nd vertices of all rows, as well as the edge between them. For $i < h$ and even, each L_r contains an edge between the $(2r + 2)$nd vertex of R_i and the $(2r + 2)$nd vertex of R_{i+1}. For $i < h$ and odd, each L_r contains an edge between the $(2r + 1)$st vertex of R_i and the $(2r + 1)$st vertex of R_{i+1}. These are all the edges of the wall.

We prove that Edge, Vertex, and Unrestricted Vertex Multicut are Max SNP-hard in walls. This means, by Arora et al. [1], that there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm for any of the three versions of Multicut with performance ratio at most $1 + \epsilon$ implies that P = NP.

As in [21], we use the concept of *L-reduction*, which is a special kind of reduction that preserves approximability. Let A and B be two optimization problems. We say *A L-reduces to B* if there are two polynomial-time algorithms f and g, and positive constants α and β, such that for each instance I of A,

(1) algorithm f produces an instance $I' = f(I)$ of B, such that the optima of I and I', of costs denoted $\mathrm{opt}_A(I)$ and $\mathrm{opt}_B(I')$, respectively, satisfy $\mathrm{opt}_B(I') \leqslant \alpha \cdot \mathrm{opt}_A(I)$, and

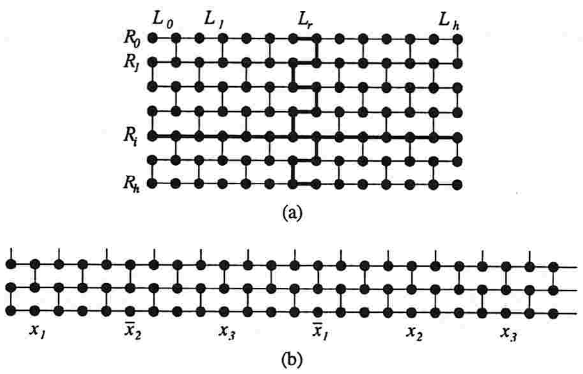


Fig. 9. (a) A wall of height six. The dark edges indicate row R_i and column L_r. (b) The labels are given according to the formula $\Phi = (x_1 \vee \bar{x}_2 \vee x_3)(\bar{x}_1 \vee x_2 \vee x_3)$.

(2) given any feasible solution of I' with cost c', algorithm g produces a solution of I with cost c such that $|c - \mathrm{opt}_A(I)| \leqslant \beta \cdot |c' - \mathrm{opt}_B(I')|$.

Theorem 18. *Edge, Vertex, and Unrestricted Vertex Multicut are Max SNP-hard in walls.*

Proof. The reduction is from the well-known Max SNP-hard problem MAX 3-SAT [21]. We only show the reduction for Unrestricted Vertex Multicut. The other two reductions are similar.

The first part of the L-reduction is the polynomial-time algorithm f and the constant α. Given any instance Φ of MAX 3-SAT, f produces an instance $W, \mathcal{C}$ of Unrestricted Vertex Multicut such that W is a wall. Also, the cost of the optimum of $W, \mathcal{C}$ in Unrestricted Vertex Multicut, denoted by $\mathrm{opt}_{\mathrm{MC}}(W, \mathcal{C})$, is at most α times the cost of the optimum of Φ in MAX 3-SAT, denoted by $\mathrm{opt}_{\mathrm{SAT}}(\Phi)$, i.e., $\mathrm{opt}_{\mathrm{MC}}(W, \mathcal{C}) \leqslant \alpha \cdot \mathrm{opt}_{\mathrm{SAT}}(\Phi)$.

Consider an instance Φ of MAX 3-SAT, that is, a collection of m clauses on n variables $x_1, \dots, x_n$, each consisting of exactly three literals. Let us describe the corresponding instance for Unrestricted Vertex Multicut. The wall W is a wall of height $3m$. To describe the collection $\mathcal{C}$ of pairs of vertices of W, consider the last row of W partitioned into m same length paths, each one associated to one of the clauses of Φ. Each path has length 6. Label the 2nd, 4th and 6th vertices in the jth path each with one of the literals in the jth clause. See Fig. 9(b) for an example. For each pair of vertices u, v in W, u labeled x_i and v labeled $\bar{x}_i$, include into $\mathcal{C}$ the pair $\{u, v\}$. For each clause, include three pairs: the three pairs formed by each pair of vertices labeled by its three literals. This ends the description of the instance of Unrestricted Vertex Multicut.

First note that W and $\mathcal{C}$ can be obtained in polynomial time in the size of Φ.

Lemma 19. $\mathrm{opt}_{\mathrm{MC}}(W, \mathcal{C}) \leqslant 6 \cdot \mathrm{opt}_{\mathrm{SAT}}(\Phi)$.

Proof. The set S of all labeled vertices of W is clearly a solution of Unrestricted Vertex Multicut for W and $\mathcal{C}$. Thus $\mathrm{opt}_{\mathrm{MC}}(W, \mathcal{C}) \leqslant |S| = 3m$. On the other hand, $\mathrm{opt}_{SAT}(\Phi) \geqslant m/2$ because either an assignment of TRUE for all variables in Φ satisfies at least half of the clauses of Φ or its negation satisfies at least half of the clauses of Φ. But then $\mathrm{opt}_{\mathrm{MC}}(W, \mathcal{C}) \leqslant 3m = 6 \cdot m/2 \leqslant 6 \cdot \mathrm{opt}_{\mathrm{SAT}}(\Phi)$. □

This finishes the first part of the L-reduction, since we can take $\alpha = 6$. The second part of the L-reduction is the constant β and the algorithm g. Given a vertex set S of W with s vertices, g produces in polynomial time a truth assignment for Φ which satisfies t clauses, where t is such that $|t - \mathrm{opt}_{\mathrm{SAT}}(\Phi)| \leqslant \beta |s - \mathrm{opt}_{\mathrm{MC}}(W, \mathcal{C})|$. We shall see that $\beta = 1$ suffices.

First note that $\mathrm{opt}_{\mathrm{MC}}(W, \mathcal{C}) \leqslant 3m - \mathrm{opt}_{\mathrm{SAT}}(\Phi)$. To verify this, it is enough to present a solution S' for W with $|S'| = 3m - \mathrm{opt}_{\mathrm{SAT}}(\Phi)$. Consider an assignment for Φ which satisfies $\mathrm{opt}_{\mathrm{SAT}}(\Phi)$ clauses of Φ. For each satisfied clause, include into S' two of its literals, leaving out a literal which is TRUE. For each nonsatisfied clause, include into S' its three literals. Clearly S' is a solution for W and has $3m - \mathrm{opt}_{\mathrm{SAT}}(\Phi)$ (one less vertex per satisfied clause). Now, note that

$$\begin{aligned}\bigl|s - \mathrm{opt}_{\mathrm{MC}}(W, \mathcal{C})\bigr| &= s - \mathrm{opt}_{\mathrm{MC}}(W, \mathcal{C}) \geqslant s - \bigl(3m - \mathrm{opt}_{\mathrm{SAT}}(\Phi)\bigr) \\ &= (s - 3m) + \mathrm{opt}_{\mathrm{SAT}}(\Phi) = \mathrm{opt}_{\mathrm{SAT}}(\Phi) - (3m - s).\end{aligned}$$

Let us present an assignment which satisfies at least $3m - s$ clauses. If $s \geqslant 3m$ then any assignment does it. If $s < 3m$ then there is a row R_{T} of W with no vertex in S (because W contains $3m + 1$ vertex-disjoint rows). Set $\tilde{x}_i$ to TRUE if and only if there is a path in $W - S$ from a vertex labeled $\tilde{x}_i$ to a vertex in row R_{T}. Such assignment is well-defined since a vertex labeled x_i and a vertex labeled $\bar{x}_i$ cannot be both connected to R_{T} in $W - S$, or S would not be a solution (because these two vertices form a pair).

Let us prove that this assignment satisfies at least $3m - s$ clauses. For this, we need some notation. Associate to each clause C_j a subgraph W_j of W induced by all vertices appearing in columns $L_{3j-3}, L_{3j-2}, L_{3j-1}$. Note that the graphs W_j are vertex disjoint.

In each W_j, we have the three vertices labeled by the literals in C_j. Recall that each two of these three vertices form a pair. One can verify that there must be at least two vertices of S in W_j, just because of these three pairs. Thus there are at least $3m - s$ subgraphs W_j which have exactly two vertices of S. But note that, if W_j has only two vertices of S, then at least one of the labeled vertices in W_j is connected to R_{T}. This means there is at least one literal in C_j which is assigned TRUE. Therefore this assignment satisfies at least $3m - s$ clauses and therefore

$$\bigl|s - \mathrm{opt}_{\mathrm{MC}}(W, \mathcal{C})\bigr| \geqslant \bigl|t - \mathrm{opt}_{\mathrm{SAT}}(\Phi)\bigr|,$$

where t is the number of clauses in Φ satisfied by the described assignment. This completes the proof of the second part of the L-reduction. □

Acknowledgment

We thank Howard Karloff for suggesting the problem, and for many helpful discussions. The first two authors also acknowledge the help of Robin Thomas in understanding the directed tree-width concept. We are grateful to the anonymous referee for comments leading to an improved presentation.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, Proof verification and hardness of approximation problems, in: Proc. 33rd IEEE Symposium on Foundations of Computer Science, 1992, pp. 14–23.
[2] S. Arnborg, J. Lagergren, Problems easy for tree-decomposable graphs, J. Algorithms 12 (2) (1991) 308–340.
[3] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, M. Yannakakis, The complexity of multi-terminal cuts, SIAM J. Comput. 23 (4) (1994) 864–894.
[4] E.A. Dinic, Algorithm for solution of a problem of maximum flow in networks with power estimation, Soviet Math. Dokl. 11 (1970) 1277–1280.
[5] L.R. Ford, D.R. Fulkerson, Maximal flow through a network, Canad. J. Math. 8 (1956) 399–404.
[6] L.R. Ford, D.R. Fulkerson, A suggested computation for maximal multicommodity network flows, Manage. Sci. 5 (1958) 97–101.
[7] M.R. Garey, D.S. Johnson, Computers and Intractability, Freeman, 1979.
[8] N. Garg, V. Vazirani, M. Yannakakis, Multiway cuts in directed and node weighted graphs, in: Proc. 21st International Colloquium on Automata, Languages, and Programming, in: Lecture Notes in Comput. Sci., Vol. 820, Springer-Verlag, 1994, pp. 487–498.
[9] N. Garg, V. Vazirani, M. Yannakakis, Approximate max-flow min-(multi)cut theorems and their applications, SIAM J. Comput. 25 (2) (1996) 235–251.
[10] N. Garg, V. Vazirani, M. Yannakakis, Primal-dual approximation algorithms for integral flow and multicut in trees, Algorithmica 18 (1) (1997) 3–20.
[11] D. Harel, R.E. Tarjan, Fast algorithms for finding nearest common ancestor, SIAM J. Comput. 13 (2) (1984) 338–355.
[12] T.C. Hu, Multicommodity network flows, Oper. Res. 9 (1963) 898–900.
[13] A. Itai, Two-commodity flow, J. ACM 25 (1978) 596–611.
[14] T. Johnson, N. Robertson, P.D. Seymour, R. Thomas, Directed tree-width, J. Combin. Theory Ser. B, to appear.
[15] P. Klein, S. Plotkin, S. Rao, Excluded minors, network decomposition, and multicommodity flow, in: Proc. 25th ACM Symposium on Theory of Computing, 1993, pp. 682–690.
[16] P. Klein, S. Plotkin, S. Rao, E. Tardos, Approximation algorithms for Steiner and directed multicuts, J. Algorithms 22 (2) (1997) 241–269.
[17] P. Klein, S. Rao, A. Agrawal, R. Ravi, An approximate max-flow min-cut relation for undirected multicommodity flow, with applications, Combinatorica 15 (2) (1995) 187–202.
[18] F.T. Leighton, S. Rao, Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms, J. ACM 46 (6) (1999) 787–832.
[19] D.B. Shmoys, Computing near-optimal solutions to combinatorial optimization problems, in: W. Cook, L. Lovász (Eds.), Combinatorial Optimization, in: DIMACS Ser. Discrete Math. Theoret. Comput. Sci., Vol. 20, Amer. Math. Society, 1995, pp. 355–397.
[20] S. Plotkin, E. Tardos, Improved bounds on the max-flow min-cut ratio for multicommodity flows, in: Proc. 25th Annual ACM Symp. on Theory of Computing, 1993, pp. 691–697.
[21] C.H. Papadimitriou, M. Yannakakis, Optimization, approximation, and complexity classes, J. Comput. System Sci. 43 (1991) 425–440.
[22] N. Robertson, P. Seymour, Graph minor II. Algorithmic aspects of tree-width, J. Algorithms 7 (1986) 309–322.
[23] N. Robertson, P. Seymour, Graph minor III. Planar tree-width, J. Combin. Theory Ser. B 36 (1984) 49–63.
[24] E. Tardos, V.V. Vazirani, Improved bounds for the max-flow min-multicut ratio for planar and $K_{r,r}$-free graphs, Inform. Process. Lett. 47 (2) (1993) 77–80.
[25] J. van Leeuwen, Graph Algorithms, Chapter 10, in: Handbook of Theoretical Computer Science, Vol. A, MIT Press, 1990, pp. 525–631.

Primal-Dual Approximation Algorithms for the Prize-Collecting Steiner Tree Problem

Paulo Feofiloff [*] Cristina G. Fernandes [*†] Carlos E. Ferreira [*‡] José Coelho de Pina [*]

Address for manuscript correspondence:

Cristina G. Fernandes

Instituto de Matemática e Estatística

Universidade de São Paulo

Rua do Matão 1010

05508-090 São Paulo/SP

Brazil

E-mail: `cris@ime.usp.br`

Tel: +55-11-3091-6301

Fax: +55-11-3091-6134

[*]Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, 05508-090 São Paulo/SP, Brazil. E-mail: `{pf,cris,cef,coelho}@ime.usp.br`. Research supported in part by PRONEX/CNPq 664107/1997-4 (Brazil).

[†]Research supported in part by CNPq Proc. 301174/97-0 (Brazil).

[‡]Research supported in part by CNPq Proc. 300752/94-6 (Brazil).

Abstract:

The primal-dual scheme has been used to provide approximation algorithms for many problems. Goemans and Williamson gave a $(2 - \frac{1}{n-1})$-approximation for the Prize-Collecting Steiner Tree Problem that runs in $O(n^3 \log n)$ time—it applies the primal-dual scheme once for each of the n vertices of the graph. We present a primal-dual algorithm that runs in $O(n^2 \log n)$, as it applies this scheme only once, and achieves the slightly better ratio of $(2 - \frac{2}{n})$. We also present a correct ratio and analysis of an algorithm by Johnson, Minkoff and Phillips that uses the primal-dual scheme only once. Tight examples are given for each of the algorithms.

Key words:

Approximation algorithms

Primal-dual scheme

Steiner trees

Prize collecting

1 Introduction

The Prize-Collecting Steiner Tree Problem is an extension of the Steiner Tree Problem where each vertex left out of the tree pays a penalty. The problem has applications in network design. Also, algorithms for it have been used for approximating other problems [1, 4].

The best approximation algorithms known for the Prize-Collecting Steiner Tree Problem are based on the primal-dual scheme. This scheme has been used to provide exact and approximation algorithms for many problems. Different linear formulation for a problem may lead to different algorithms. In this paper we present one such formulation for the Prize-Collecting Steiner Tree Problem and use it in the design of a new approximation algorithm.

Consider a graph $G = (V, E)$, a function c from E into $\mathbb{Q}_{\geq}$ and a function π from V into $\mathbb{Q}_{\geq}$. For any subset F of E and any subset W of V, let $c(F) := \sum_{e \in F} c_e$ and $\pi(W) := \sum_{w \in W} \pi_w$. The **Prize-Collecting Steiner Tree Problem** (PCST) consists of the following: given G, c, and π, find a tree T in G such that

$$c(E_T) + \pi(V \setminus V_T) \text{ is minimum.}$$

(V_H and E_H denote the vertex and edge sets of a graph H respectively.) The rooted variant of the problem requires T to contain a given root vertex.

Goemans and Williamson [5, 6] used a primal-dual scheme to derive a $(2 - \frac{1}{n-1})$-approximation, where $n := |V|$, for the rooted PCST. Trying all possible choices for the root, they obtained a $(2 - \frac{1}{n-1})$-approximation for the unrooted PCST. The resulting algorithm runs in time $O(n^3 \log n)$. Johnson, Minkoff and Phillips [7] proposed a modification of the algorithm that permits running the primal-dual scheme only once, resulting in a running-time of $O(n^2 \log n)$. They claimed the modification achieves the same approximation ratio that the original algorithm for the unrooted PCST. Unfortunately, their analysis uses a lower bound that does not necessarily hold. The algorithm has in fact a ratio of 2. The analysis that proves that, shown here, is very tight and is not a straightforward modification of their analysis. Cole et al. [2] stated the correct approximation ratio for Johnson, Minkoff and Phillips' algorithm, but did not present a complete proof.

This paper contains two results. First, we present a proof that Johnson, Minkoff and Phillips' algorithm is a 2-approximation and an example which assures that this analysis is tight. Second, we propose a modification of Goemans and Williamson's algorithm for the PCST based on a somewhat different linear program. We show that the new algorithm achieves a ratio of $2 - \frac{2}{n}$. It requires only one run of the primal-dual scheme, resulting in a running time of $O(n^2 \log n)$. Also, we present a family of graphs which proves that the analysis is tight. Though the improvement on the approximation ratio is small, the new algorithm seems interesting and might be useful for the design of a better algorithm for PCST. The behavior of the new algorithm is not far from the behaviour of Johnson, Minkoff and Phillips' algorithm. It somehow stops before that one does, and, for this, achieves a better ratio. The artifact that makes it stop earlier is suttle and is not obviously polynomially testable.

The paper is organized as follows. The next section introduces some notation and shows some preliminaries. Section 3 has the description of the new algorithm, while its analysis is given in Section 4. Section 5 discusses Johnson, Minkoff and Phillips' algorithm and a variant of it for the rooted PCST. The pruning phase is briefly discussed in Section 6.

2 Notation and preliminaries

For any collection $\mathcal{L}$ of subsets of V and any subset X of V, let $\overline{X} := V \setminus X$, $\mathcal{L}^X := \{L \in \mathcal{L} : L \subseteq X\}$ and $\mathcal{L}_X := \{L \in \mathcal{L} : L \supseteq X\}$. When $X = V_T$ or $X = \overline{V_T}$, we write T or $\overline{T}$ instead. For any e in E, let $\mathcal{L}(e) := \{L \in \mathcal{L} : e \in \delta_G L\}$, where $\delta_G L$ stands for the set of edges of G with one end in L and the other in $\overline{L}$. For any function y from $\mathcal{L}$ into $\mathbb{Q}_{\geq}$ and any subcollection $\mathcal{M}$ of $\mathcal{L}$, let $y(\mathcal{M}) := \sum_{L \in \mathcal{M}} y(L)$.

We say that y **respects** a function c defined on E (relatively to $\mathcal{L}$) if

$$y(\mathcal{L}(e)) \quad \leq \quad c_e \qquad \text{for each } e \text{ in } E. \tag{1}$$

An edge e is **tight for** y if equality holds in (1). The inequality in (1) is the usual restriction on edge e: the sum of y_R for all R in $\mathcal{L}$ that e "crosses" does not exceed c_e.

We say y **respects** a function π defined on V (relatively to $\mathcal{L}$) if

$$\begin{aligned} y(\mathcal{L}^L) + y(\mathcal{L}_{\overline{L}}) &\leq \pi(L) \quad \text{for each } L \text{ in } \mathcal{L} && (2) \\ \text{and} \quad y(\mathcal{L}^{\overline{L}}) + y(\mathcal{L}_L) &\leq \pi(\overline{L}) \quad \text{for each } L \text{ in } \mathcal{L}. && (3) \end{aligned}$$

An element L of $\mathcal{L}$ is **tight for** y if equality holds in (2). If equality holds in (3), we say $\overline{L}$ is **tight for** y. The inequality in (2) is slightly different than the usual one for PCST. The usual one says that the sum of y_R for all R in $\mathcal{L}$ contained in L does not exceed the sum of the penalties of all elements in L. In (2), we include in the sum the y_R for supersets R of $\overline{L}$ as well. This has the effect of making the algorithm stop earlier. The inequality in (3) is the same as in (2) for the complement of a set in $\mathcal{L}$.

An edge is **internal to** a partition $\mathcal{P}$ of V if both of its ends are in the same element of $\mathcal{P}$. All other edges are **external to** $\mathcal{P}$. For any external edge, there are two elements of $\mathcal{P}$ containing its ends. We call these two elements the **extremes** of the edge in $\mathcal{P}$.

A collection $\mathcal{L}$ of subsets of V is **laminar** if, for any two elements L_1 and L_2 of $\mathcal{L}$, either $L_1 \cap L_2 = \emptyset$ or $L_1 \subseteq L_2$ or $L_1 \supseteq L_2$. The collection of maximal elements of a laminar collection $\mathcal{L}$ will be denoted by $\mathcal{L}^*$. So, $\mathcal{L}^*$ is a disjoint collection.

We say a forest F in G is **connected in** a subset L of V if $F[V_F \cap L]$ is connected. For any laminar collection $\mathcal{S}$ of subsets of V, we say a tree T of G **has no bridges in** $\mathcal{S}$ if $|\delta_T S| \neq 1$ (therefore, either $\delta_T S = \emptyset$ or $|\delta_T S| \geq 2$) for all S in $\mathcal{S}$.

We denote by opt(PCST(G, c, π)) the minimum value of the expression $c(E_T) + \pi(\overline{V_T})$ when T is a tree in G.

The following lower bound serves as motivation for the new algorithm.

Lemma 2.1 *Given a connected subgraph T of G, a laminar collection $\mathcal{L}$ of subsets of V, and a function y from $\mathcal{L}$ into $\mathbb{Q}_{\geq}$ that respects c and π,*

$$y(\mathcal{L}) \leq c(E_T) + \pi(\overline{V_T}) .$$

Proof. If $\mathcal{L}_T = \emptyset$ then let $S := V$, else let S be a minimal set in $\mathcal{L}_T$. Now, consider the partition of $\mathcal{L}$ into the following three sets: $\mathcal{B} := \{L \in \mathcal{L} : \delta_T L \neq \emptyset\}$, $\mathcal{C} := \mathcal{L}^{\overline{S}} \cup \mathcal{L}_S$, and $\mathcal{D} := \mathcal{L}^{S \setminus V_T}$. We have that

$$\begin{aligned}
y(\mathcal{B}) &= \textstyle\sum_{L\in\mathcal{B}} y_L \;\le\; \sum_{L\in\mathcal{B}} |\delta_T L| y_L \;=\; \sum_{e\in E_T} y(\mathcal{L}(e)) \;\le\; \sum_{e\in E_T} c_e \;=\; c(E_T)\,, \\
y(\mathcal{C}) &= y(\mathcal{L}^{\overline{S}}) + y(\mathcal{L}_S) \;\le\; \pi(\overline{S})\,, \\
\text{and}\quad y(\mathcal{D}) &= \textstyle\sum_{L\in\mathcal{D}^*} y(\mathcal{L}_L) \;\le\; \sum_{L\in\mathcal{D}^*} \pi(L) \;\le\; \pi(S \setminus V_T)\,.
\end{aligned}$$

The lemma follows from the three inequalities. ■

Corollary 2.2 *For any laminar collection $\mathcal{L}$ of subsets of V and any function y from $\mathcal{L}$ into $\mathbb{Q}_{\geq}$ that respects c and π,*

$$y(\mathcal{L}) \leq \mathrm{opt}(\textsc{Pcst}(G, c, \pi))\ . \ ■$$

The proposed algorithm relies on Corollary 2.2. The above lower bound on $\mathrm{opt}(\textsc{Pcst}(G, c, \pi))$ can also be obtained from the following linear program: find vectors x and z that

$$\begin{aligned}
&\text{minimize} && \textstyle\sum_{e\in E} c_e x_e + \sum_{L\in\mathcal{R}} \pi(L) z_L \\
&\text{subject to} && \textstyle\sum_{e\in\delta_G S} x_e + \sum_{L\supseteq S} z_L + \sum_{L\supseteq \overline{S}} z_L \;\ge\; 1 \quad \text{for each } S \text{ in } \mathcal{R}\,, \\
&&& x_e \;\ge\; 0 \quad \text{for each } e \text{ in } E\,, \\
&&& z_L \;\ge\; 0 \quad \text{for each } L \text{ in } \mathcal{R}\,,
\end{aligned} \tag{4}$$

where $\mathcal{R}$ denotes the collection of all subsets of V. Given a solution T for $\textsc{Pcst}(G, c, \pi)$, set $x_e := 1$ if $e \in E_T$ and $x_e := 0$ otherwise. Set $z_L := 1$ if $L = V \setminus V_T$ and $z_L := 0$ otherwise. The pair (x, z) is a feasible solution for (4) and its value is $c(E_T) + \pi(V \setminus V_T)$. Thus this program is a relaxation of $\textsc{Pcst}(G, c, \pi)$. Its dual consists of: find vector y that

$$\begin{aligned}
&\text{maximize} && y(\mathcal{R}) \\
&\text{subject to} && y(\mathcal{R}(e)) \;\le\; c_e \quad \text{for each } e \text{ in } E\,, \\
&&& y(\mathcal{R}^L) + y(\mathcal{R}_{\overline{L}}) \;\le\; \pi(L) \quad \text{for each } L \text{ in } \mathcal{R}\,, \\
&&& y_S \;\ge\; 0 \quad \text{for each } S \text{ in } \mathcal{R}\,.
\end{aligned} \tag{5}$$

Corollary 2.2 is stronger in a sense than the tradicional argument that any feasible solution for (5) gives a lower bound on $\text{opt}(\text{PCST}(G, c, \pi))$. Let $\mathcal{L}$ and y be as in Corollary 2.2. Set $y_L := 0$ for each L in $\mathcal{R} \setminus \mathcal{L}$. One has to prove that such a y is indeed a feasible solution to infer Corollary 2.2 without Lemma 2.1. (The proof of this fact is indeed very similar to the proof of Lemma 2.1.)

3 Unrooted growth clustering algorithm

Our algorithm, which we call GW-UNROOTED-GROWTH, receives G, c, π and returns a tree T in G such that

$$c(E_T) + \pi(\overline{V_T}) \leq (2 - \tfrac{2}{n})\, \text{opt}(\text{PCST}(G, c, \pi)) \,.$$

The main difference between the new algorithm and Goemans and Williamson's algorithm is that the complement of a set with nonnull y can become tight. When this happens, the algorithm stops. During most of the course of the algorithm, it behaves exactly as Johnson, Minkoff and Phillips' algorithm (though more conditions are being tested). By this, we mean that the dual variables grow in the same way, except possibly at the end, and the edges that enter the forest are the same, except for the last one. Indeed, if one looks at the inequalities in (1), (2) and (3), the first one is the same, in the second one, the second, different term, is zero except at the very end, when there are only two components left, and the third one is the different one, but once it becames tight, the algorithm stops because, after that, there is at most one component which is not contained in a tight set. Next we describe the algorithm in details.

Each iteration of the algorithm starts with a spanning forest F in G, a laminar collection $\mathcal{L}$ of subsets of V such that $\bigcup \mathcal{L} = V$, a subcollection $\mathcal{S}$ of $\mathcal{L}$, a subcollection $\mathcal{M}$ of $\mathcal{L}$, and a function y from $\mathcal{L}$ into $\mathbb{Q}_{\geq}$ such that the following invariants hold:

(i1) all edges of F are internal to $\mathcal{L}$;

(i2) F is connected in each element of $\mathcal{L}$;

(i3) y respects c and π relatively to $\mathcal{L}$;

(i4) each edge of F is tight for y;

(i5) each element of $\mathcal{S}$ is tight for y;

(i6) $|\mathcal{M}| \leq 1$ and, if $M \in \mathcal{M}$, then $\overline{M}$ is tight for y;

(i7) for any tree T in G, if T is connected in each element of $\mathcal{L}$ and has no bridges in $\mathcal{S}$, then

$$\textstyle\sum_{e \in E_T} y(\mathcal{L}(e)) + y(\mathcal{L}^{\overline{T}}) + y(\mathcal{L}_T) \leq \left(2 - \frac{2}{n}\right) y(\mathcal{L}) \,. \tag{6}$$

The first iteration starts with $F = (V, \emptyset)$, $\mathcal{L} = \{\{v\} : v \in V\}$, $\mathcal{S} = \mathcal{M} = \emptyset$, and $y = 0$. Each iteration consists of the following:

Case 1: $|\mathcal{L}^* \setminus \mathcal{S}| > 1$ **and** $\mathcal{M} = \emptyset$.

Let ε be the largest number in $\mathbb{Q}_{\geq}$ such that the function y' defined by

$$y'_L = \begin{cases} y_L + \varepsilon \,, & \text{if } L \in \mathcal{L}^* \setminus \mathcal{S} \\ y_L \,, & \text{otherwise} \end{cases}$$

respects c and π.

Subcase 1A: some edge e external to $\mathcal{L}^*$ is tight for y'.

Let L_1 and L_2 be the extremes of e in $\mathcal{L}^*$. Set $y'_{L_1 \cup L_2} := 0$ and start a new iteration with $F + e$, $\mathcal{L} \cup \{L_1 \cup L_2\}$, $\mathcal{S}$, $\emptyset$, y' in the roles of F, $\mathcal{L}$, $\mathcal{S}$, $\mathcal{M}$, y respectively.

Subcase 1B: some element L of $\mathcal{L}^* \setminus \mathcal{S}$ is tight for y'.

Start a new iteration with F, $\mathcal{L}$, $\mathcal{S} \cup \{L\}$, $\emptyset$, y' in the roles of F, $\mathcal{L}$, $\mathcal{S}$, $\mathcal{M}$, y respectively.

Subcase 1C: for some M in $\mathcal{L}$, the set $\overline{M}$ is tight for y'.

Start a new iteration with F, $\mathcal{L}$, $\mathcal{S}$, $\{M\}$, y' in the roles of F, $\mathcal{L}$, $\mathcal{S}$, $\mathcal{M}$, y respectively.

Case 2: $|\mathcal{L}^* \setminus \mathcal{S}| = 1$ or $\mathcal{M} \neq \emptyset$.

If $\mathcal{M} \neq \emptyset$, let M be the only element of $\mathcal{M}$; else, let M be the only element of $\mathcal{L}^* \setminus \mathcal{S}$. In either case, call subalgorithm GW-PRUNING with arguments $F \cap M$, $\mathcal{L}^M$, and $\mathcal{S}^M$. The subalgorithm will return a subcollection $\mathcal{Z}$ of $\mathcal{S}^M$. Return $T := (F \cap M) - \bigcup \mathcal{Z}$ and stop.

Subalgorithm GW-PRUNING receives a tree T_0, a laminar collection $\mathcal{L}$ of subsets of V_{T_0}, and a subcollection $\mathcal{S}$ of $\mathcal{L}$. Each iteration begins with a subcollection $\mathcal{Z}$ of $\mathcal{S}$ such that $T := T_0 - \bigcup \mathcal{Z}$ is a tree connected in each element of $\mathcal{L}$. The first iteration begins with $\mathcal{Z} = \emptyset$.

Case A: $|\delta_T S| = 1$ for some S in $\mathcal{S}$.

Start a new iteration with $\mathcal{Z} \cup \{S\}$ in place of $\mathcal{Z}$.

Case B: $|\delta_T S| \neq 1$ for each S in $\mathcal{S}$.

Return $\mathcal{Z}$ and stop.

The subcollection $\mathcal{Z}$ of $\mathcal{S}$ that subalgorithm GW-PRUNING returns is such that the forest $T_0 - \bigcup \mathcal{Z}$ is a tree, is connected in each element of $\mathcal{L}$, and has no bridges in $\mathcal{S}$.

4 Analysis of the algorithm

Note that, by the choice of ε in Case 1, one of the three subcases apply. At the beginning of each iteration of the GW-UNROOTED-GROWTH algorithm, invariants (i1) to (i6) hold trivially. Let us verify that invariant (i7) holds as well. It is clear that it holds at the beginning of the first iteration, because $y_L = 0$ for all L in $\mathcal{L}$. Now assume that invariant (i7) holds at the beginning of an iteration where Subcase 1A occurs. Let T be a tree in G, connected in each element of $\mathcal{L}' := \mathcal{L} \cup \{L_1 \cup L_2\}$, with no bridges in $\mathcal{S}$. We must show that (6) holds with $\mathcal{L}'$ and y' in the roles of $\mathcal{L}$ and y. Since $y'_{L_1 \cup L_2} = 0$, this is equivalent to

$$\textstyle\sum_{e \in E_T} y'(\mathcal{L}(e)) + y'(\mathcal{L}^{\overline{T}}) + y'(\mathcal{L}_T) \leq \left(2 - \frac{2}{n}\right) y'(\mathcal{L}) \,. \tag{7}$$

If $\varepsilon = 0$ then (7) is true because it is identical to (6). Now suppose $\varepsilon > 0$ and let $\mathcal{A} := \mathcal{L}^* \setminus \mathcal{S}$. Since y' differs from y only in $\mathcal{A}$, this is equivalent to

$$\textstyle\sum_{e\in E_T} |\mathcal{A}(e)| + |\mathcal{A}^{\overline{T}}| + |\mathcal{A}_T| \;\leq\; (2 - \frac{2}{n})\, |\mathcal{A}| \,. \tag{8}$$

Let $\mathcal{N} := \{L \in \mathcal{L}^* : \delta_T L = \emptyset\}$. Since T is connected, $\mathcal{A}^{\overline{T}} \cup \mathcal{A}_T = \mathcal{N} \cap \mathcal{A}$. As $\sum_{e\in E_T} |\mathcal{A}(e)| = \sum_{L\in\mathcal{A}} |\delta_T L|$ and $(2 - \frac{2}{n})\, |\mathcal{A}| \geq 2\,|\mathcal{A}| - 2$, the inequality (8) will follow from

$$\textstyle\sum_{L\in\mathcal{A}} |\delta_T L| + |\mathcal{N} \cap \mathcal{A}| \;\leq\; 2|\mathcal{A}| - 2 \,. \tag{9}$$

If $\mathcal{A} \subseteq \mathcal{N}$, then (9) holds because $\sum_{L\in\mathcal{A}} |\delta_T L| + |\mathcal{N} \cap \mathcal{A}| = |\mathcal{A}| \leq 2|\mathcal{A}| - 2$, since $|\mathcal{A}| > 1$ in Case 1. Now assume $\mathcal{A} \not\subseteq \mathcal{N}$ and consider the graph $H := (\mathcal{L}^*, E')$, where E' is the set of edges of T external to $\mathcal{L}^*$ and each element of E' is incident to its two extremes in $\mathcal{L}^*$. Since T is connected in each element of $\mathcal{L}$, this graph is a forest. It has one nontrivial component and $|\mathcal{N}|$ trivial components. Hence, $|E'| = |\mathcal{L}^*| - 1 - |\mathcal{N}|$ and

$$\begin{aligned}
\textstyle\sum_{L\in\mathcal{A}} |\delta_T L| \;&=\; \textstyle\sum_{L\in\mathcal{L}^*} |\delta_T L| - \sum_{L\in\mathcal{L}^*\cap\mathcal{S}} |\delta_T L| \\
&=\; 2|E'| - \textstyle\sum_{L\in\mathcal{L}^*\cap\mathcal{S}} |\delta_T L| \\
&=\; 2|\mathcal{L}^*| - 2 - 2|\mathcal{N}| - \textstyle\sum_{L\in\mathcal{L}^*\cap\mathcal{S}} |\delta_T L| \\
&\leq\; 2|\mathcal{L}^*| - 2 - 2|\mathcal{N}| - 2|(\mathcal{L}^* \cap \mathcal{S}) \setminus \mathcal{N}| \qquad (10) \\
&=\; 2|\mathcal{L}^*| - 2 - 2|\mathcal{N}| - 2|\mathcal{L}^* \cap \mathcal{S}| + 2|\mathcal{N} \cap \mathcal{S}| \\
&=\; 2|\mathcal{L}^* \setminus \mathcal{S}| - 2 - 2|\mathcal{N} \setminus \mathcal{S}| \\
&\leq\; 2|\mathcal{A}| - 2 - |\mathcal{N} \cap \mathcal{A}| \,,
\end{aligned}$$

where (10) holds because T has no bridges in $\mathcal{S}$. We have thus shown that invariant (i7) remains valid after an occurrence of Subcase 1A. The very same proof applies in Subcases 1B and 1C.

Having proved invariants (i1) to (i7), we are ready to analyze the last iteration. By virtue of invariant (i4), the tree T produced by the algorithm in Case 2 is such that

$$c(E_T) \;=\; \textstyle\sum_{e\in E_T} c_e \;=\; \sum_{e\in E_T} y(\mathcal{L}(e)) \,.$$

If $\mathcal{M} \neq \emptyset$, let $\mathcal{X} := \{\overline{M}\} \cup \mathcal{Z}^*$; else, let $\mathcal{X} := (\mathcal{L}^* \cap \mathcal{S}) \cup \mathcal{Z}^*$. In either case, the collection $\mathcal{X}$ is disjoint. Every element of $\mathcal{X}$ is tight for y, according to invariants (i5) and (i6). Hence,

$$\begin{aligned} \pi(\overline{V_T}) &= \textstyle\sum_{X \in \mathcal{X}} \pi(X) \\ &= \textstyle\sum_{X \in \mathcal{X}} \left(y(\mathcal{L}^X) + y(\mathcal{L}_{\overline{X}})\right) \\ &= \textstyle\sum_{X \in \mathcal{X}} y(\mathcal{L}^X) + \sum_{X \in \mathcal{X}} y(\mathcal{L}_{\overline{X}}) \\ &\leq y(\mathcal{L}^{\overline{T}}) + \textstyle\sum_{X \in \mathcal{X}} y(\mathcal{L}_{\overline{X}}) && (11) \\ &\leq y(\mathcal{L}^{\overline{T}}) + y(\mathcal{L}_T)\,. && (12) \end{aligned}$$

Inequality (11) holds since every element of $\mathcal{X}$ is disjoint from V_T. In order to explain inequality (12), we reason as follows. First, observe that $V_T \subseteq \overline{X}$ and therefore $\mathcal{L}_{\overline{X}} \subseteq \mathcal{L}_T$ for every X in $\mathcal{X}$. Next, $\mathcal{L}_{\overline{X}} \cap \mathcal{L}_{\overline{X'}} = \emptyset$ for any two different elements X and X' of $\mathcal{X}$, since $\mathcal{X}$ is disjoint.

The tree T is connected in each element L of $\mathcal{L}$. Indeed, for any x and y in $L \cap V_T$, the path from x to y in T is the same as in F and uses only vertices of L because of invariant (i2). In addition, subalgorithm GW-PRUNING makes sure T has no bridges in $\mathcal{S}$. Hence, invariant (i7) holds for T and therefore

$$c(E_T) + \pi(\overline{V_T}) \;\leq\; \textstyle\sum_{e \in E_T} y(\mathcal{L}(e)) + y(\mathcal{L}^{\overline{T}}) + y(\mathcal{L}_T) \;\leq\; (2 - \frac{2}{n})\, y(\mathcal{L})\,.$$

Finally, by virtue of invariant (i3) and Corollary 2.2,

$$c(E_T) + \pi(\overline{V_T}) \;\leq\; (2 - \tfrac{2}{n})\operatorname{opt}(\text{PCST}(G, c, \pi))\,.$$

(In fact, as in Goemans and Williamson's analysis, one can easily prove the stronger statement $c(E_T) + 2\,\pi(\overline{V_T}) \leq (2 - \frac{2}{n})\operatorname{opt}(\text{PCST}(G, c, \pi))$.) This completes the proof of the following theorem:

Theorem 4.1 *Algorithm* GW-UNROOTED-GROWTH *is a* $(2 - \frac{2}{n})$*-approximation for* PCST(G, c, π).

The approximation ratio stated in Theorem 4.1 is tight, as the example in Figure 1 shows. For the graph in this example, the algorithm increases y_L to 1 for each singleton set L. At this point, all the dark edges in Figure 1(a) become tight and enter the forest F one by one (in an arbitrary order). When all of them enter F, the algorithm stops (the GW-PRUNING algorithm does nothing in this example) and outputs the tree induced by the dark edges. The optimal tree, on the other hand, consists only of the dark edge depicted in Figure 1(b). The ratio between the costs of these two solutions approaches $2 - \frac{2}{n}$ as ε tends to zero.

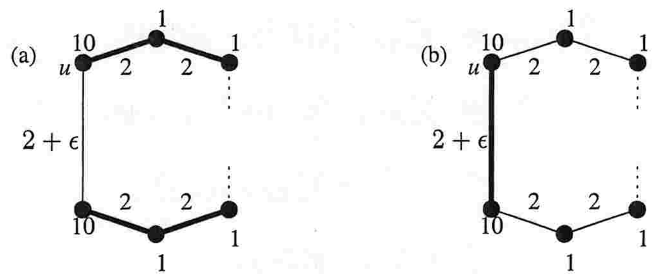


Figure 1: (a) The dark edges indicate the solution produced by the GW-UNROOTED-GROWTH algorithm, a solution of cost $2(n-1)$. (b) The only dark edge indicates the optimal solution, whose cost is $n + \varepsilon$.

Algorithm GW-UNROOTED-GROWTH can be implemented to run in $O(n^2 \log n)$ time. The details of such implementation are analogous to those of Goemans and Williamson's algorithm for the rooted PCST. There are a few differences, though, that are worth mentioning. One should carry, for each set L in $\mathcal{L}$, two values that we call $\Delta_1(L)$ and $\Delta_2(L)$, defined as

$$\Delta_1(L) := \Pi(L) - y(\mathcal{L}^L) - y(\mathcal{L}_{\overline{L}}) \quad \text{and} \quad \Delta_2(L) := \Pi(\overline{L}) - y(\mathcal{L}^{\overline{L}}) - y(\mathcal{L}_L).$$

In other words, each set L in $\mathcal{L}$ keeps the current slack in inequalities (2) and (3). During the algorithm, one has to keep such values up-dated. Using the notation in the algorithm, in each iteration, if Subcase 1A occurs, one has to decrease $\Delta_1(L)$ by ε, for every maximal active set L in $\mathcal{L}$, and one has to decrease $\Delta_2(L)$ by $t\varepsilon$, where t is the number of maximal active sets in the iteration, for every L in $\mathcal{L}$. Additionally, one has to set $\Delta_1(L_1 \cup L2)$ and $\Delta_2(L_1 \cup L2)$. Roughly, $\Delta_1(L_1 \cup L2) := \Delta_1(L_1) + \Delta_1(L_2)$ and $\Delta_2(L_1 \cup L2) := (\Delta_2(L_1) + \Delta_2(L_2) - \Delta_1(L_1) - \Delta_1(L_2))/2$. A few adjustments are needed in $\Delta_1(L_1 \cup L_2)$ when the current forest has few (two or three) compontents. The overall time required for these calculations is $O(n)$ per iteration, since $\mathcal{L}$ has $O(n)$ sets. Therefore, it is indeed possible to get an $O(n^2 \log n)$ implementation, as in the original Goemans and Williamson's algorithm.

Also, the ideas proposed by Klein [8] and by Gabow, Goemans and Williamson [3] can be used, resulting in implementations with running time $O(n\sqrt{m}\log n)$ and $O(n(n+\sqrt{m\log n}))$, respectively, where $m := |E|$. Finally, using the technique of Cole et al. [2], one can get a $(2 - \frac{2}{n} + \frac{1}{\mathrm{poly}(n)})$-approximation that runs in $O((n+m)\log^2 n)$-time.

5 Previous unrooted growth clustering algorithms

Johnson, Minkoff and Phillips [7] suggested a modification of Goemans and Williamson's algorithm, which differs from the one presented above only at the definition of "y respects π". For their algorithm, which we call JMP, (2) and (3) are replaced by

$$y(\mathcal{L}^L) \leq \pi(L) \quad \text{for each } L \text{ in } \mathcal{L}. \tag{13}$$

Set L is said to be tight if equality holds in (13). Of course, JMP has no Subcase 1C and no set $\mathcal{M}$. Algorithm JMP is based on the following LP: find vectors x and z that

$$\begin{array}{lrcll} \text{minimize} & \multicolumn{4}{l}{\sum_{e\in E} c_e x_e + \sum_{L\in\mathcal{R}} \pi(L) z_L} \\ \text{subject to} & \sum_{e\in\delta_G S} x_e + \sum_{L\supseteq S} z_L & \geq & 1 & \text{for each } S \text{ in } \mathcal{R}, \\ & x_e & \geq & 0 & \text{for each } e \text{ in } E, \\ & z_L & \geq & 0 & \text{for each } L \text{ in } \mathcal{R}, \end{array} \tag{14}$$

where, remember, $\mathcal{R}$ denotes the collection of all subsets of V. The dual of this linear program is: find vector y that

$$\begin{array}{lrcll} \text{maximize} & \multicolumn{4}{l}{y(\mathcal{R})} \\ \text{subject to} & y(\mathcal{R}(e)) & \leq & c_e & \text{for each } e \text{ in } E, \\ & y(\mathcal{R}^L) & \leq & \pi(L) & \text{for each } L \text{ in } \mathcal{R}, \\ & y_S & \geq & 0 & \text{for each } S \text{ in } \mathcal{R}. \end{array} \tag{15}$$

Unfortunately, (14) is not a relaxation of PCST, so its optimal value cannot be used as a lower bound for opt(PCST(G,c,π)). Algorithm JMP finds a feasible solution for (15) (see Minkoff's thesis [9] for the analysis). But the value of this feasible solution might be larger than opt(PCST(G,c,π)), as the example in Figure 2 shows. Indeed this example shows that the approximation ratio of the JMP algorithm can be arbitrarily close to 2, independent of the size of the graph. This contradicts Theorem 3.2 in [7].

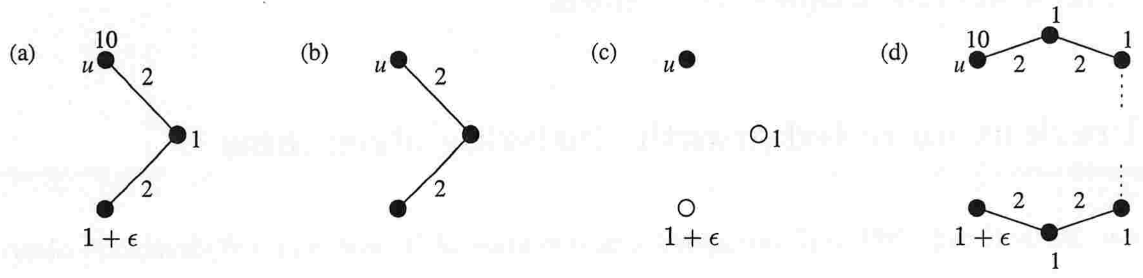


Figure 2: (a) An instance of the PCST. (b) The solution produced by the JMP algorithm when $\varepsilon > 0$. Its cost is 4. (c) The optimal solution, consisting of only vertex u, has cost $2 + \varepsilon$. (d) A similar instance of arbitrary size.

Minkoff [9] proposed a modification of the JMP algorithm, which we denote by MINKOFF, for the rooted PCST. The resulting algorithm is a 2-approximation, and not a $(2 - \frac{1}{n})$-approximation as claimed. The example in Figure 2 with u as root contradicts Theorem 2.6 in [9].

In fact, the analysis which shows that these two algorithms are 2-approximations is not straightforward: there are some non-trivial technical details. See the appendix for a precise description and a sketch of the analysis of the JMP algorithm.

6 Strong pruning

A dynamic programming approach can solve the PCST on trees in polynomial time. The GW-PRUNING algorithm can be replaced by this dynamic programming, which is usually called "strong pruning". Johnson, Minkoff and Phillips [7] and Minkoff [9] have already suggested this procedure.

Consider the modified JMP and MINKOFF algorithms that use strong pruning. It is easy to see that these are 2-approximations for the unrooted and rooted PCST: they produce trees at least as good as the ones produced by their variants with the ordinary pruning. However, the example shown in Figure 2 does not apply for these modified algorithms. The worst example we have for them is depicted in Figure 1 and the ratio achieved is $2 - \frac{2}{n}$. We conjecture that the JMP and MINKOFF algorithms with strong pruning achieve a ratio better than 2. The approximation ratio of the GW-UNROOTED-GROWTH algorithm does not improve by using strong pruning, as the example in Figure 1 shows.

7 Acknowledgments

We thank Márcia R. Cerioli, Orlando Lee, Flávio K. Miyazawa, Yoshiharu Kokayakawa and Yoshiko Wakabayashi for many helpful discussions.

References

[1] F. Chudak, T. Roughgarden, and D.P. Williamson. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. In *Proc. 8th Conference on Integer Programming and Combinatorial Optimization Conference (IPCO)*, volume 2081 of *Lecture Notes in Computer Science*, page 60 ff. Springer, 2001.

[2] R. Cole, R. Hariharan, M. Lewenstein, and E. Porat. A faster implementation of the Goemans-Williamson clustering algorithm. In *Symposium on Discrete Algorithms*, pages 17–25, 2001.

[3] H.N. Gabow, M.X. Goemans, and D.P. Williamson. An efficient approximation algorithm for the survivable network design problem. *Mathematical Programming*, 82(1-2, Ser. B):13–40, 1998.

[4] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 320–309, 1996.

[5] M.X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[6] D.S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, 1997.

[7] D.S. Johnson, M. Minkoff, and S. Phillips. The prize collecting Steiner tree problem: theory and practice. In *Symposium on Discrete Algorithms*, pages 760–769, 2000.

[8] P. Klein. A data structure for bicategories, with application to speeding up an approximation algorithm. *Information Processing Letters*, 52(6):303–307, 1994.

[9] M. Minkoff. The prize-collecting Steiner tree problem. Master's thesis, Massachusetts Institute of Technology, 2000.

A Johnson, Minkoff and Phillips' algorithm

We describe the JMP algorithm and show that it is a 2-approximation for the PCST. The algorithm receives G, c, π and returns a tree T in G such that $c(E_T) + \pi(\overline{V_T}) \leq 2\,\mathrm{opt}(\mathrm{PCST}(G, c, \pi))$. Each iteration starts with a spanning forest F in G, a laminar collection $\mathcal{L}$ of subsets of V with $\bigcup \mathcal{L} = V$, a subcollection $\mathcal{S}$ of $\mathcal{L}$, and a function y from $\mathcal{L}$ into $\mathbb{Q}_{\geq}$ such that invariants (i1) up to (i5) hold, (i6) does not apply, and, in the place of (i7), we have the following:

(i7') for any vertex u and any tree T in G, if T is connected in each L in $\mathcal{L}$ and has no bridges in $\mathcal{S}$, then

$$\textstyle\sum_{e \in E_T} y(\mathcal{L}(e)) + y(\mathcal{L}^{\overline{T}}) \leq 2y(\mathcal{L} \setminus \mathcal{L}_u) . \qquad (16)$$

(We write $\mathcal{L}_u$ and $\mathcal{L}^u$ instead of $\mathcal{L}_{\{u\}}$ and $\mathcal{L}^{\{u\}}$ respectively.) The first iteration starts with $F = (V, \emptyset)$, $\mathcal{L} = \{\{v\} : v \in V\}$, $\mathcal{S} = \emptyset$, and $y = 0$. Each iteration consists of the following:

Case 1: $|\mathcal{L}^* \setminus \mathcal{S}| > 1$.

Let ε be the largest number in $\mathbb{Q}_{\geq}$ such that the function y' defined by

$$y'_L = \begin{cases} y_L + \varepsilon\,, & \text{if } L \in \mathcal{L}^* \setminus \mathcal{S} \\ y_L\,, & \text{otherwise} \end{cases}$$

respects c and π.

Subcase 1A: some edge e external to $\mathcal{L}^*$ is tight for y'.

Let L_1 and L_2 be the extremes of e in $\mathcal{L}^*$. Set $y'_{L_1 \cup L_2} := 0$ and start a new iteration with $F+e$, $\mathcal{L} \cup \{L_1 \cup L_2\}$, $\mathcal{S}$, y' in the roles of F, $\mathcal{L}$, $\mathcal{S}$, y respectively.

Subcase 1B: some element L of $\mathcal{L}^* \setminus \mathcal{S}$ is tight for y'.

Start a new iteration with F, $\mathcal{L}$, $\mathcal{S} \cup \{L\}$, y' in the roles of F, $\mathcal{L}$, $\mathcal{S}$, y respectively.

Case 2: $|\mathcal{L}^* \setminus \mathcal{S}| = 1$.

Let M be the only element of $\mathcal{L}^* \setminus \mathcal{S}$. Call subalgorithm GW-PRUNING with arguments $F \cap M$, $\mathcal{L}^M$, and $\mathcal{S}^M$. The subalgorithm returns a subcollection $\mathcal{Z}$ of $\mathcal{S}^M$. Return $T := (F \cap M) - \bigcup \mathcal{Z}$ and stop.

The following lemma and its corollary establish the correct lower bound on $\mathrm{opt}(\textsc{Pcst}(G, c, \pi))$.

Lemma A.1 *Given a connected subgraph T of G, a laminar collection $\mathcal{L}$ of subsets of V and a function y from $\mathcal{L}$ into $\mathbb{Q}_{\geq}$ that respects c and π, $y(\mathcal{L} \setminus \mathcal{L}_T) \leq c(E_T) + \pi(\overline{V_T})$.*

Proof. The proof is analogous to the proof of Lemma 2.1, so we omit it. ■

Corollary A.2 *For any optimal solution T^* of* $\textsc{Pcst}(G, c, \pi)$*, $y(\mathcal{L} \setminus \mathcal{L}_{T^*}) \leq$* $\mathrm{opt}(\textsc{Pcst}(G, c, \pi))$.

The analysis of JMP differs from the one for the GW-UNROOTED-GROWTH algorithm in two points. First, while for the GW-UNROOTED-GROWTH we had to prove (8), for the JMP algorithm we have to prove

$$\textstyle\sum_{e \in E_T} |\mathcal{A}(e)| + |\mathcal{A}^{\overline{T}}| \;\leq\; 2\,|\mathcal{A} \setminus \mathcal{A}_u| \;\leq\; 2\,|\mathcal{A}| - 2\,|\mathcal{A}_u|\,. \tag{17}$$

But, since

$$\begin{aligned}\textstyle\sum_{e\in E_T} |\mathcal{A}(e)| + |\mathcal{A}^{\overline{T}}| &\leq \textstyle\sum_{e\in E_T} |\mathcal{A}(e)| + |\mathcal{A}^{\overline{T}}| + |\mathcal{A}_T| \\ &\leq 2|\mathcal{A}| - 2 \\ &\leq 2|\mathcal{A}| - 2|\mathcal{A}_u|\,,\end{aligned}$$

to show (17), it is enough to prove (9). Second, at the end of Case 2, $c(E_T) \leq \sum_{e\in E_T} y(\mathcal{L}(e))$, $\pi(\overline{V_T}) \leq y(\mathcal{L}^{\overline{T}})$ and, if T^* is an arbitrary optimal solution of $\text{P}\textsc{cst}(G,c,\pi)$ and u is an arbitrary vertex in T^*, then

$$\begin{aligned}c(E_T) + \pi(\overline{V_T}) &\leq \textstyle\sum_{e\in E_T} y(\mathcal{L}(e)) + y(\mathcal{L}^{\overline{T}}) \\ &\leq 2y(\mathcal{L} \setminus \mathcal{L}_u) \\ &\leq 2y(\mathcal{L} \setminus \mathcal{L}_{T^*}) \\ &\leq 2\text{opt}(\text{P}\textsc{cst}(G,c,\pi))\,.\end{aligned}$$

This concludes the proof of the following theorem.

Theorem A.3 *The* JMP *algorithm is a 2-approximation for the* PCST. ■

In the analysis of the GW-UNROOTED-GROWTH algorithm, the -2 in (9) corresponds to the $-\frac{2}{n}$ in the approximation ratio. In the analysis of the JMP algorithm, the -2 corresponds to the $-2|\mathcal{A}_u|$ term in (17).

A similar theorem holds for the MINKOFF algorithm [9].

Theorem A.4 *The* MINKOFF *algorithm is a 2-approximation for the rooted* PCST. ■

JOURNAL OF ALGORITHMS **28**, 105–124 (1998)
ARTICLE NO. AL980931

A Better Approximation Ratio for the Minimum Size k-Edge-Connected Spanning Subgraph Problem*

Cristina G. Fernandes[†]

Department of Computer Science, Universidade de Sao Paulo, Rua do Matao, 1010, 05508-900 Sao Paulo, Brazil

Received October 29, 1996; revised January 30, 1998

Consider the minimum size k-edge-connected spanning subgraph problem: given a positive integer k and a k-edge-connected graph G, find a k-edge-connected spanning subgraph of G with the minimum number of edges. This problem is known to be NP-complete. Khuller and Raghavachari presented the first algorithm which, for all k, achieves a performance ratio smaller than a constant which is less than two. They proved an upper bound of 1.85 for the performance ratio of their algorithm. Currently, the best known performance ratio for the problem is $1 + 2/(k + 1)$, achieved by a slower algorithm of Cheriyan and Thurimella. In this article, we improve Khuller and Raghavachari's analysis, proving that the performance ratio of their algorithm is smaller than 1.7 for large enough k, and that it is at most 1.75 for all k. Second, we show that the minimum size 2-edge-connected spanning subgraph problem is MAX SNP-hard. © 1998 Academic Press

1. INTRODUCTION

The study of connectivity in graph theory has important applications in the areas of network reliability and network design. Several approximation algorithms were developed for the problem of finding subgraphs satisfying certain connectivity requirements [2, 4, 6, 7, 8, 9, 12, 13, 16, 18]. In this article, we concentrate on the minimum size k-edge-connected spanning subgraph problem: given a positive integer k and given a k-edge-connected graph G, find a k-edge-connected spanning subgraph of G with the minimum number of edges.

*A preliminary version of this article appeared in the Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1997.
[†]E-mail: cris@ime.usp.br. Research supported in part by the CNPq (Brazil), under contract 200975/92-7.

This problem is known to be NP-complete [5], even for $k = 2$: if the graph G is Hamiltonian, a 2-edge-connected spanning subgraph of G with the minimum number of edges must be a Hamiltonian cycle. So the goal is to look for good polynomial-time approximation algorithms for the problem.

A quality of an approximation algorithm is frequently measured by its *approximation* or *performance ratio*. For the minimum size k-edge-connected spanning subgraph problem, the approximation ratio of an algorithm is the infimum, over all possible inputs, of the ratio between the number of edges in the output of the algorithm and the number of edges in an optimal solution.

For a long time, two was the best approximation ratio achieved for all k. This comes from the fact that every minimal k-edge-connected spanning subgraph has at most kn edges [14, 15] and that every k-edge-connected graph has minimum degree at least k.

Khuller and Raghavachari [10] presented the first algorithm which, for all k, achieves a performance ratio smaller than a constant which is less than two. They proved an upper bound of 1.85 for the performance ratio of their algorithm. In this article, we improve their analysis, proving that the performance ratio of their algorithm is smaller than 1.7 for large enough k, and that it is at most 1.75 for all k.

Karger [9] presented an algorithm with performance ratio $1 + O(\sqrt{(\log n)/k})$. This is smaller than 2 only when $k \gg \log n$. Also, there were algorithms with approximation ratio smaller than 2 for some particular values of k. An algorithm for $k = 2$ with a ratio of 1.5 was presented by Khuller and Vishkin [13]. As observed in [10], by combining the biconnectivity algorithm in [13] and the sparse certificate algorithm in [2], one can easily obtain a ratio of $2 - 1/k$.

The currently best known performance ratio for the minimum size k-edge-connected spanning subgraph problem is $1 + 2/(k + 1)$. It is achieved by a new algorithm developed by Cheriyan and Thurimella [3] concurrently with the research presented in this article. Let $G = (V, E)$ be the input graph. In a first step, Cheriyan and Thurimella's algorithm finds an edge set $M \subseteq E$ of minimum cardinality such that every vertex in V is incident to at least k edges of M. In a second step, their algorithm finds a minimal edge set $F \subseteq E \setminus M$ such that $(V, M \cup F)$, the output of their algorithm, is k-edge-connected. The running time of Cheriyan and Thurimella's algorithm is $O(k^3|V|^2 + |E|^{1.5}(\log|V|)^2)$, while the running time of Khuller and Raghavachari's algorithm is $O(k^2|V|^2)$, which is better. The reader is referred to [3] for more details.

The bound on the approximation ratio of Khuller and Raghavachari's algorithm given in [10] is actually better than 1.85 for small values of k: it is 1.5 for $k = 2$, 1.666... for $k = 3$, 1.75 for $k = 4$, 1.733... for $k = 5$, etc. In addition, these bounds are tight for $k = 2$ and 3. Our analysis improves Khuller and Raghavachari's bound for any fixed $k \geq 4$. In particular, we get 1.65 for $k = 4$, and we get 1.68 for $k = 5$.

Khuller and Vishkin [13] introduced the following concept: a *tree-carving in a graph* $G = (V, E)$ is a partition of the vertex set V into subsets $V_1, \dots, V_a$ with the following properties. Each subset constitutes a node of a tree Γ. For every vertex v in V_j, all the neighbors of v in G belong either to V_j itself or to V_i, where V_i is adjacent to V_j in the tree Γ. They used this to prove the 1.5 bound on the ratio of their algorithm for $k = 2$. We generalize the concept of tree-carving and, from this generalization, we prove that the performance ratio of Khuller and Raghavachari's algorithm is smaller than 1.7 for large enough k.

Finally, we show that the minimum size 2-edge-connected spanning subgraph problem is MAX SNP-hard. This implies that there is a constant $\varepsilon > 0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at most $1 + \varepsilon$ would imply that $P = NP$ [1].

In the next section, we review Khuller and Raghavachari's algorithm. In Section 3, we present a better analysis and we prove a new upper bound of 1.75 for all k on the performance ratio of the algorithm. Section 4 generalizes the concept of a tree-carving, and shows some properties which are then used to prove that the performance ratio of the algorithm is less than 1.7 for large enough k. The proof of MAX SNP-hardness appears in Section 5.

2. KHULLER AND RAGHAVACHARI'S ALGORITHM

Let us start with some notation. Consider a graph G, a depth-first search (DFS) forest F in G, and the corresponding DFS numbering of the vertices of G. For each edge e in G, call *top* (*bottom*) the endpoint of e with smaller (bigger) DFS number. It is known that any edge of G which is not in F is a back edge. In fact, this means that its top endpoint is an ancestor in F of its bottom endpoint. For each edge t in F, we call an edge e of G which is not in F a *back edge of* t if t lies on the unique path in (V, F) between the two endpoints of e. (See Fig. 1.)

The input for Khuller and Raghavachari's algorithm is a positive integer k and a k-edge-connected graph $G = (V, E)$. Their algorithm works in phases. It consists of $\lfloor k/2 \rfloor$ phases, plus an extra final phase when k is odd. For now, let us assume that k is even. Later we comment on the final phase for odd k.

During phase i, the algorithm selects a set S_i of edges of G which induces a $2i$-edge-connected spanning subgraph of G. Hence at the end of

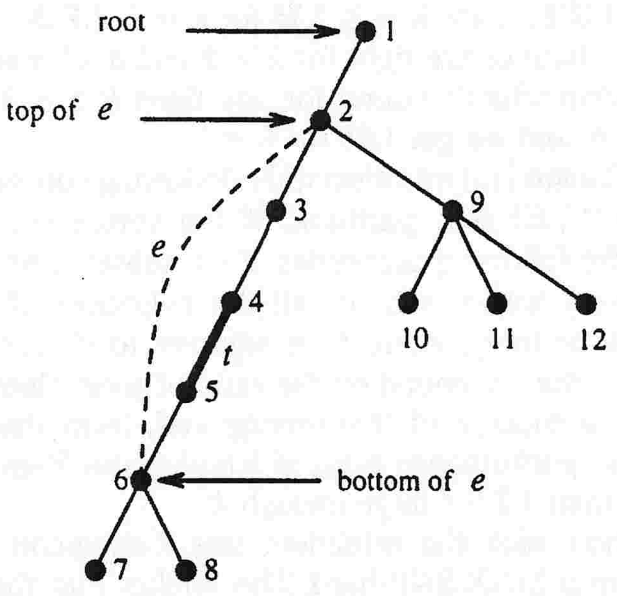


FIG. 1. A DFS forest F, plus an edge e which is a back edge of the edge t in F.

$k/2$ iterations, the current set $S_{k/2}$ induces a k-edge-connected spanning subgraph of G which is the output of their algorithm.

How does each phase work? Let $S_0 := \emptyset$. In phase $i \geq 1$, the algorithm chooses two disjoint edge sets F_i and B_i, both disjoint from S_{i-1}. The set F_i is simply the edge set of a DFS forest in the graph $(V, E - S_{i-1})$. The edge set B_i is built as follows. Initially $B_i = \emptyset$. Then the edges in F_i are scanned in the reverse of the DFS order (i.e., in the reverse of the order they were included in F_i). When edge $t \in F_i$ is scanned, the algorithm finds a minimum cut between the endpoints of edge t in the graph $(V, S_{i-1} \cup F_i \cup B_i)$. Denoting by X one of the sides of this minimum cut, let $(X, \overline{X})$ be the corresponding cut in G, that is, $(X, \overline{X}) = \{h \in E$: h has exactly one endpoint in $X\}$. If the cut $(X, \overline{X})$ in G contains exactly $2(i-1)+1 = 2i-1$ edges in $S_{i-1} \cup F_i \cup B_i$, then let $A := (E - (S_{i-1} \cup F_i \cup B_i)) \cap (X, \overline{X})$. The algorithm adds to B_i an edge $e \in A$ which has the smallest DFS-numbered top endpoint and sets $t_e := t$. Set $S_i := S_{i-1} \cup F_i \cup B_i$ at the end of phase i. This concludes the definition of the algorithm for even k. For odd k, the algorithm runs $\lfloor k/2 \rfloor$ phases, and in the end, it runs a "half" phase, computing $F_{\lceil k/2 \rceil}$ only (not $B_{\lceil k/2 \rceil}$). The final $S_{\lceil k/2 \rceil}$ is simply $S_{\lfloor k/2 \rfloor} \cup F_{\lceil k/2 \rceil}$.

The reader is referred to [10], where Khuller and Raghavachari give the proof of the correctness of this algorithm, as well as the upper bound of 1.85 on its performance ratio. In fact, for fixed k, Khuller and Raghavachari's analysis gives bounds smaller than 1.85 [10]. More specifically, it gives 1.5 for $k = 2$, 1.666... for $k = 3$, 1.75 for $k = 4$, and 1.733... for $k = 5$. These bounds are tight for $k = 2$ and 3.

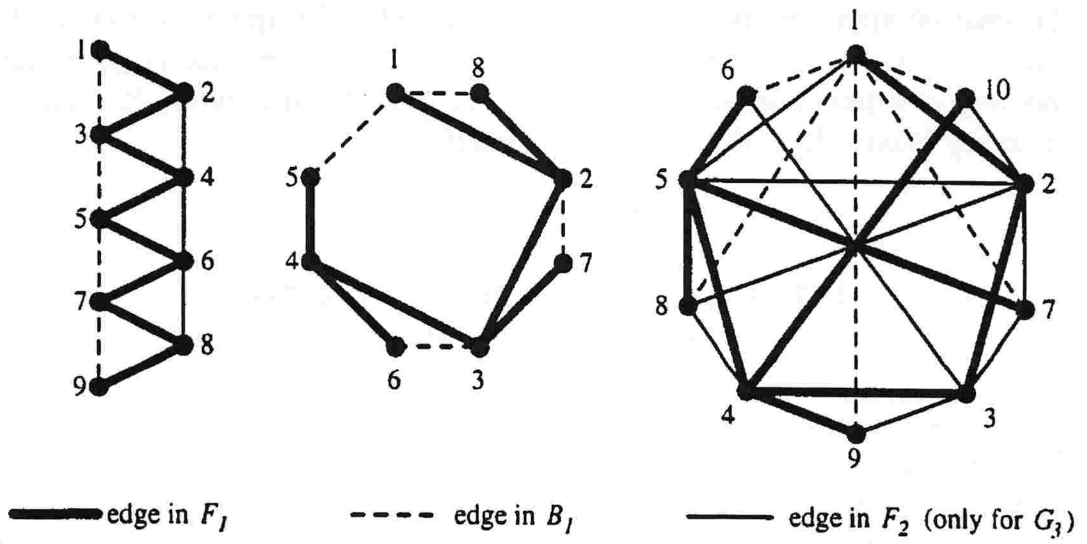


FIG. 2. Graphs G_1, G_2, and G_3, with the depth-first search numbering given by F_1.

In Fig. 2, we present our tight examples for $k = 2$ and 3. Our results originated from attempts at generalizing these examples for $k > 3$, and we think they can give some intuition. Let us denote by OPT an optimal solution and by opt the size of OPT. We give two essentially different types of examples for $k = 2$: G_1 and G_2. In both cases, opt $= n$, where n is the number of vertices in the graph. Also, if the algorithm chooses as F_1 the dark edges, then the output of the algorithm will have $n + (n - 1)/2$ edges (in both cases). As n grows, the ratio between the size of the output and opt approaches 1.5. Note that both these examples can be extended for large values of n. Graph G_3 is our tight example for $k = 3$ (only for even n). In this case, OPT can be the outer cycle and the chords from each vertex in the outer cycle to the opposite vertex in the outer cycle. We have that opt $= 3n/2$. Also, assume that the algorithm chooses as F_1 the dark edges. In general, F_1 is built as in G_2: it is a path starting at any vertex in the cycle, and passing by each other vertex in the cycle until it has visited half of the vertices. Then the remaining vertices are leaves of F_1. Note that F_1 is in fact a DFS tree. Consider that all edges from the leaves in F_1 to the root are present in G_3, so that these edges are chosen as B_1. In our example, the dashed edges would be B_1. Besides that, the remaining edges of G_3 should contain a tree. All the remaining edges would be F_2. The output of the algorithm will have $2(n - 1) + n/2$ edges. As n grows, the ratio between the size of the output and opt approaches $5/3 = 1.666\ldots$. This example can be extended, as suggested, for any large value of n such that $n = 2(2t + 1)$ for some positive integer t.

The intuition that we would like to get from these examples comes from the following reasoning. For Khuller and Raghavachari's bound to be tight, opt must be approximately $kn/2$, and the first $\lceil k/4 \rceil B_i$ must have approximately $\text{opt}/(k-2(i-1))$ edges. However, as we will see, for large k, the only way to expect that $|B_i| \approx \text{opt}/(k-2(i-1))$ for the initial B_is is if F_i has many leaves. But, this cannot happen if $\text{opt} \approx kn/2$.

3. THE 1.75 UPPER BOUND ON THE PERFORMANCE RATIO

In this section, we improve the analysis of Khuller and Raghavachari's algorithm by presenting a better upper bound on $|B_i|$.

The following definition is critical to understanding the improved analysis. For each e in B_i, recall that t_e is the edge in F_i which made e to be included in B_i.

DEFINITION 3.1. b_e is the bottom endpoint of t_e.

For each i, $1 \le i \le \lfloor k/2 \rfloor$, let l_i be the number of b_e which are leaves of F_i.

Our improvement is first based on the following stronger version of Lemma 3.6 in [10]:

LEMMA 3.2. $(k-2(i-1))|B_i| + (i-1)l_i \le \text{opt}$, *for all* i, $1 \le i \le \lfloor k/2 \rfloor$.

Proof. The number of pairs (u, h), where u is a vertex of G and h is an edge of OPT incident to u, is $2\,\text{opt}$. Next, we compute a lower bound for this number.

For each edge e in B_i, OPT has at least $k-2(i-1)$ edges in P_e ([10, p. 442]). So there are at least $2(k-2(i-1))$ pairs associated to e: two for each of the $k-2(i-1)$ edges of OPT in P_e. If b_e is a leaf of F_i, e can be associated to $(k-2(i-1))+k$ pairs. The first $k-2(i-1)$ pairs would be the ones mentioned previously whose first coordinate is not b_e. The remaining k pairs have b_e as first coordinate: because OPT is k-edge-connected, there are k edges of OPT incident to b_e.

Let us show that no two edges in B_i have an associated pair in common. Khuller and Raghavachari proved that the sets P_e are pairwise disjoint ([10, p. 442]), so the pairs coming from P_e are certainly distinct. The k pairs corresponding to b_e cannot be repeated also, because the only edges in P_f with b_e as endpoint are in P_e.

Therefore, we must have $l_i((k-2(i-1))+k) + (|B_i| - l_i)2(k-2(i-1)) \le 2\,\text{opt}$, which is equivalent to the statement of the lemma, concluding the proof. ▮

We will present a lower bound on l_i, for $1 \leq i \leq \lfloor k/2 \rfloor$, which will imply an improved upper bound on $|B_i|$. Let us fix i such that $1 \leq i \leq \lfloor k/2 \rfloor$. For such an i,

$$|P_e| \geq k - 2(i-1) \geq 2, \tag{1}$$

for any e in B_i. Denote by B the set $\{b_e : e \in B_i\}$. The following fact will be used in the next lemma.

FACT 3.3. *For any $e, f \in B_i$ such that b_f is a child of b_e, there is a child of b_f not in B.*

Proof. Consider $e, f \in B_i$ such that b_f is a child of b_e. Because $P_e \cap P_f = \emptyset$, all the edges in P_f must have b_e as their top endpoint. By (1), $|P_f| \geq 2$. Thus there is at least one edge g in $P_f - \{t_f\}$. The bottom endpoint of g is not b_f (otherwise g would have the same two endpoints of f). Because b_f is a child of b_e, we must have that f was inserted in B_i before e. At the moment f was included in B_i, there could not be another edge h in B_i such that t_h is on the path in F_i between the two endpoints of g. This is true because if, by contradiction, we assume there is an h in B_i such that t_h is on the path in F_i between the two endpoints of g, we must have that $g \in P_h$. But $g \in P_f$ too, contradicting $P_f \cap P_h = \emptyset$. So, the child of b_f on the path from b_f to the bottom endpoint of g is not in B. ▮

The lower bound on l_i is given by the following lemma.

LEMMA 3.4. *$l_i \geq 2|B_i| - n$, for any i, $1 \leq i < k/2$, and, for even k, $l_{k/2} \geq 3|B_{k/2}| - 2n$.*

Proof. Fix i such that $1 \leq i < k/2$. Denote by N the set of b_e in B which are not leaves in F_i. We will associate to each b_e in N a distinct vertex z_e in $V - B$. The association is made in two phases:

Phase 1. For each b_e in N which has a child in $V - B$, let z_e be a child of b_e in $V - B$.

Phase 2. For each b_e in N whose children are all in B, let $h \in B_i$ be such that b_h is one of b_e's children. By Fact 3.3, b_h has a child not in B.

Case 1. b_h has two or more children not in B. Let z_e be any child of b_h not in B and distinct of z_h.

Case 2. b_h has exactly one child not in B. Note that b_h satisfies the statement of Phase 1, which means that z_h was assigned in Phase 1. So this child is in fact z_h. But $i \leq (k-1)/2$, which implies that $|P_h| \geq k - 2(i-1) \geq 3$. This means that there is at least one edge g in P_h which has a descendant of z_h as its bottom endpoint. Let z_e be the child of z_h on the path from z_h to the bottom endpoint of g. Observe that z_e cannot be in B, otherwise the edge inserted in B_i by the edge whose bottom endpoint is z_e would forbid b_h to be in B. (See Fig. 3.)

Let us verify that distinct b_e are assigned distinct z_e. In the first phase, clearly no two distinct b_e are assigned the same z_e. Also no two z_e assigned in Phase 2, Case 1 coincide. The same for two z_e assigned in Phase 2, Case 2. We explicitly make sure that a z_e assigned in Phase 2, Case 1 does not coincide with a z_e assigned in Phase 1. A z_e assigned in Phase 2, Case 2 cannot coincide with a z_e assigned in Phase 1, because $z_h \notin B$. Finally, note that, in Phase 2, Case 1, z_e is a grandchild of b_e, while in Phase 2, Case 2 z_e is a great-grandchild of b_e. Thus, the only way of Phase 2, Case 1 and Phase 2, Case 2 select the same z_e is if b_h in Case 2 had its z_h assigned in Phase 2, Case 1. But, as we have said, b_h had its z_h assigned in Phase 1. This completes the proof that we can assign a different vertex in $V - B$ to each vertex in N.

Let $Z = \{z_e\colon e \in B_i \text{ and } b_e \in N\}$. We have that $|Z| = |N|$, and $|B| = |B_i|$. Moreover, $N \subseteq B$ and $Z \subseteq V - B$. Thus $l_i = |B - N| = |B| - |N| = |B| - |Z| \geq |B| - |V - B| = |B| - (n - |B|) = 2|B| - n = 2|B_i| - n$, completing the proof of the first part of Lemma 3.4.

The proof of the second part differs from the foregoing proof only at Phase 2, Case 2. For $i = k/2$ (even k), we just have $|P_h| \geq 2$. Therefore, z_h might not have a child to be selected as z_e. So, instead of associating to each b_e in N a *distinct* z_e in $V - B$, we associate a z_e which might have already been selected, but at most once. In other words, each $z \in Z = \{z_e\colon e \in B_i \text{ and } b_e \in N\}$ will correspond to at most two b_e in N. More specifically, in Phase 2, Case 2, we select $z_e = z_h$. This implies $|N| \leq 2|Z|$, and we have $l_{k/2} = |B| - |N| \geq |B| - 2|Z| \geq |B| - 2(n - |B|) = 3|B| - 2n = 3|B_{k/2}| - 2n$, completing the proof of the lemma. ∎

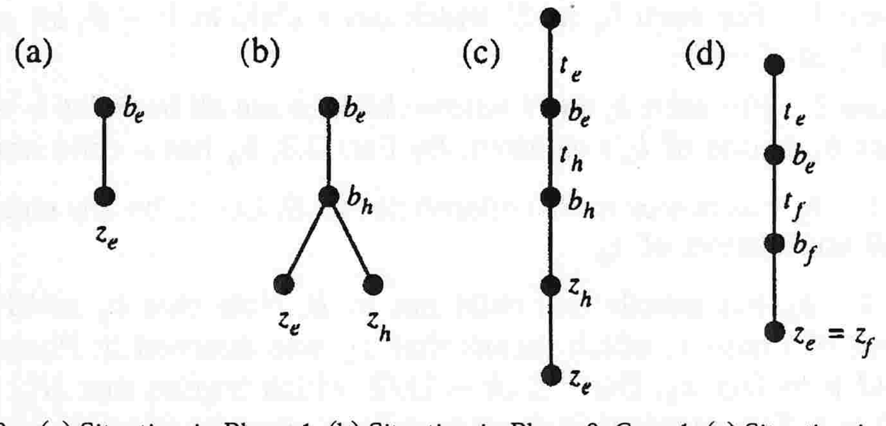


FIG. 3. (a) Situation in Phase 1. (b) Situation in Phase 2, Case 1. (c) Situation in Phase 2, Case 2. (d) Situation in the new Phase 2, Case 2 (used later).

Now, using Lemma 3.4 and $n \leq 2\,\mathrm{opt}/k$ in Lemma 3.2, a new upper bound on $|B_i|$ follows.

COROLLARY 3.5. *We have*

$$|B_i| \leq \left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right)\mathrm{opt}, \quad \textit{for any } i \textit{ such that } 1 \leq i < \frac{k}{2}.$$

$$|B_{k/2}| \leq \frac{2(3k-4)}{k(3k-2)}\mathrm{opt}.$$

This bound is not the best we can get using this technique, as we will show in the next section. But it implies the better upper bound of 1.75 on the performance ratio of Khuller and Raghavachari's algorithm, as stated in the following theorem.

THEOREM 3.6. *The performance ratio of Khuller and Raghavachari's algorithm is at most* 1.75 *for all k. More specifically, it is at most*

$$1.75 - \frac{3k^2-6k+8}{2k^2(3k-2)}, \quad \textit{for even } k, \quad \textit{and} \quad 1.75 - \frac{2k-3}{4k^2}, \quad \textit{for odd } k > 1.$$

Proof. For $k = 1$, the algorithm is optimum, i.e., the ratio is 1. For odd $k > 1$, we have (the inequality holding because $|F_i| < n$ and by Corollary 3.5),

$$\begin{aligned}
\frac{|S_{(k+1)/2}|}{\mathrm{opt}} &= \frac{\sum_{i=1}^{(k+1)/2}|F_i| + \sum_{i=1}^{(k-1)/2}|B_i|}{\mathrm{opt}} \\
&\leq \frac{(k+1)n/2}{kn/2} + \sum_{i=1}^{(k-1)/2}\left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right) \\
&= \left(1 + \frac{1}{k}\right) + \frac{k-1}{2k} + \frac{2}{k^2}\sum_{i=1}^{(k-1)/2}(i-1) \\
&= \frac{3}{2} + \frac{1}{2k} + \frac{(k-1)(k-3)}{4k^2} \\
&= \frac{3}{2} + \frac{k^2-2k+3}{4k^2} = \frac{7}{4} - \frac{2k-3}{4k^2}.
\end{aligned}$$

Observe that $2k - 3 > 0$ for all $k \geq 3$ and $4k^2 > 0$ for all positive k.

The accounting is similar for even k. Hence,

$$\begin{aligned}\frac{|S_{k/2}|}{\text{opt}} &= \frac{\sum_{i=1}^{k/2}|F_i| + \sum_{i=1}^{k/2}|B_i|}{\text{opt}} \\ &\le \frac{kn/2}{kn/2} + \sum_{i=1}^{k/2-1}\left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right) + \frac{2(3k-4)}{k(3k-2)} \\ &= 1 + \left(\frac{1}{2} - \frac{1}{k}\right) + \frac{2}{k^2}\sum_{i=1}^{k/2-1}(i-1) + \frac{2(3k-4)}{k(3k-2)} \\ &= \frac{3}{2} + \frac{k^2 - 10k + 8}{4k^2} + \frac{2(3k-4)}{k(3k-2)} \\ &= \frac{3}{2} + \frac{1}{4} - \frac{5k-4}{2k^2} + \frac{2(3k-4)}{k(3k-2)} = \frac{7}{4} - \frac{3k^2 - 6k + 8}{2k^2(3k-2)}.\end{aligned}$$

In this case, $3k^2 - 6k + 8 > 0$ always, and $2k^2(3k-2) > 0$ for all positive k. Hence the ratio is in fact always smaller than 1.75 for all values of k. ∎

4. GENERALIZED TREE-CARVINGS

In this section, we generalize some of the results from the previous section. In particular, we present stronger versions of both Lemmas 3.2 and 3.4. From these, we get an even better upper bound on the $|B_i|$, and, consequently, on the performance ratio of Khuller and Raghavachari's algorithm for large values of k.

We start by introducing a generalization of the tree-carving concept and by giving a generalization of some results given in [13].

Consider a positive integer k, a k-edge-connected graph $G = (V, E)$, a nonnegative integer $c < k$, and a subset S of E such that (V, S) is a c-edge-connected spanning subgraph of G.

A *c-tree-carving in G with respect to S* is a partition of the vertex set V into subsets $V_1, V_2, \dots, V_a$ with the following properties.

(1) Each subset consists of a node of a *rooted forest* Γ.

(2) For every edge $e = uv$ of $(V, E - S)$, the unique x and y such that $u \in V_x$ and $v \in V_y$ satisfy either $x = y$ or V_x is adjacent to V_y in Γ.

(3) For each arc e in Γ, there is a cut $(Y_e, \overline{Y}_e) = \{e\}$ in Γ such that, for $X_e = \bigcup\{V_j : V_j \in Y_e\}$, the cut $(X_e, \overline{X}_e)$ in G contains exactly c edges of S. Denote by P_e the set of edges in $(X_e, \overline{X}_e)$ not in S.

When $c = 0$, $S = \varnothing$ and Γ is a tree, this definition is the same as the tree-carving definition.

Khuller and Raghavachari's algorithm can be modified so that, at each phase, a $2(i-1)$-tree-carving in G with respect to S_{i-1} is computed. This is done as follows. At the end of phase i, let $T_i = \{t_e\colon e \in B_i\}$. Note that $T_i \subseteq F_i$ and $|T_i| = |B_i|$. Let Γ_i be the rooted forest obtained from F_i by contracting all the edges of $F_i - T_i$. Clearly, Γ_i has exactly $|T_i| = |B_i|$ arcs. Each node of Γ_i corresponds to a subset of the original graph $G = (V, E)$ that was collapsed into this node. So, the nodes of Γ_i are subsets $V_1, V_2, \dots, V_a$ of V which define a partition of V for some $a \le n$, where n is the number of vertices of G. Observe that this in fact defines a $2(i-1)$-tree-carving with exactly $|B_i|$ arcs.

Consider a c-tree-carving $V_1, \dots, V_a$ of G with respect to S, and let Γ, $(X_e, \overline{X}_e)$, and P_e be as in the preceding definition.

FACT 4.1. *Any k-edge-connected spanning subgraph of G has at least $k - c$ edges in each P_e.*

Proof. Let H be a k-edge-connected spanning subgraph of G. For each arc e in Γ, there must be at least k edges of H in $(X_e, \overline{X}_e)$. Exactly c edges in $(X_e, \overline{X}_e)$ are in S. Hence, there must be at least $k - c$ edges of H in $(X_e, \overline{X}_e)$ not in S, that is, in P_e. ▮

FACT 4.2. *For any distinct arcs e and f in Γ, the edge sets $P_e \subseteq E(G)$ and $P_f \subseteq E(G)$ are disjoint.*

Proof. Let $e = (V_i, V_j)$ and $f = (V_x, V_y)$ be two arcs in Γ. Suppose that $P_e \cap P_f \neq \varnothing$. Let us prove that $e = f$. Let $h \in P_e \cap P_f$. By the definition of P_e in (3), h has one endpoint in X_e and the other endpoint in $\overline{X}_e$ and $(Y_e, \overline{Y}_e) = \{e\} = \{(V_i, V_j)\}$. So, by (2), h has one endpoint in V_i and the other endpoint in V_j. Also, for the same reason, h has one endpoint in V_x and the other endpoint in V_y. But, by the unicity mentioned in (2), we must have ($V_i = V_x$ and $V_j = V_y$) or ($V_i = V_y$ and $V_j = V_x$). In both cases, $e = f$. ▮

Denote by t the number of nonroot nodes of Γ and denote by l the number of nonroot leaves of Γ. (Note that $t = |E(\Gamma)|$.) The following result is an extension of Lemma 3.2.

LEMMA 4.3. *Any k-edge-connected spanning subgraph of G has at least $(k - c)t + cl/2$ edges.*

Proof. Let H be a k-edge-connected spanning subgraph of G. Let us count the number of pairs (V_y, h), where h is an edge of H in $(V_y, \overline{V}_y)$.

For each nonroot V_j of Γ, let V_i be its parent in Γ, and let e be the arc (V_i, V_j). Recall that $P_e \subseteq E - S$. By Fact 4.1, H has at least $k - c$ edges in P_e. So there are at least $2(k - c)$ pairs associated to e: two for each of the $k - c$ edges of H in P_e (one with V_i as first coordinate, the other one with V_j).

In fact, if V_j is a leaf of Γ, arc e can be associated with $(k - c) + k = 2k - c$ pairs. The $k - c$ first pairs would be the ones mentioned previously whose first coordinate is V_i. The remaining k pairs have V_j as first coordinate: because H is k-edge-connected, there are k edges of H in $(V_j, \overline{V}_j)$.

Let us show that no two arcs have an associated pair in common. By Fact 4.2, the sets P_e are pairwise disjoint, so the pairs coming from P_e are certainly distinct. The k pairs corresponding to a leaf V_j cannot be repeated also because V_j is the endpoint of only one arc in Γ.

Denoting by m the number of edges in H, we can have at most $2m$ pairs. Therefore, $2(k - c)(t - l) + (2k - c)l \leq 2m$, which implies that $m \geq (k - c)t + cl/2$, completing the proof of the lemma. ▮

The generalization of Lemma 3.4 we want to present gives a lower bound on l. But before stating the generalization, we need to prove one more fact.

FACT 4.4. *For each arc $e = (V_i, V_j)$ in Γ, $|V_i| + |V_j| \geq 2\sqrt{k - c}$.*

Proof. Consider the cut $(X_e, \overline{X}_e)$ in G corresponding to e. Because G is k-edge-connected, and $(X_e, \overline{X}_e)$ contains exactly c edges of S, there must be at least $k - c$ edges of $E - S$ in $(X_e, \overline{X}_e)$. These edges must have one endpoint in V_i and the other endpoint in V_j. The maximum number of edges with one endpoint in V_i and the other endpoint in V_j is $|V_i| \cdot |V_j|$. Thus $|V_i| \cdot |V_j| \geq k - c$. This implies, that $|V_i| + |V_j| \geq 2\sqrt{k - c}$. ▮

Now we can prove the following lower bound on l, the number of leaves of the forest Γ.

LEMMA 4.5. *$l \geq (1 + 1/(\lfloor\sqrt{k - c}\rfloor - 1))t - (1/(\lfloor\sqrt{k - c}\rfloor - 1))n$.*

Proof. The proof of this lemma uses basically the same technique used in Lemma 3.4. We will associate a set Z_i of vertices of V with each nonroot V_i, where $Z_i \subseteq V_i \cup V_j$ for some V_j which is adjacent to V_i in Γ. The sets Z_i will be pairwise disjoint and will contain exactly $\lfloor\sqrt{k - c}\rfloor$ vertices if V_i is not a leaf of Γ, and will contain exactly one vertex if V_i is a leaf of Γ. The association is done in two phases. (Phase 1 is executed before Phase 2.)

Phase 1. For each nonroot V_i which is a leaf of Γ or such that $|V_i| \geq \lfloor \sqrt{k-c} \rfloor$: Let Z_i be a subset of V_i with exactly $\lfloor \sqrt{k-c} \rfloor$ vertices if V_i is not a leaf of Γ, and with one vertex if V_i is a leaf. (Note that $|V_i| \geq 1$ always.)

Phase 2. For each nonroot V_i which is not a leaf of Γ and such that $|V_i| < \lfloor \sqrt{k-c} \rfloor$: Let $e = (V_i, V_j)$ be an arc of Γ joining V_i to one of its children V_j. By Fact 4.4, $|V_i| + |V_j| \geq 2\sqrt{k-c}$. Thus we must have $|V_j| \geq 2\sqrt{k-c} - |V_i| \geq \sqrt{k-c}$, and so $|V_j| \geq \lfloor \sqrt{k-c} \rfloor$, which implies that V_j had its set Z_j assigned previously in Phase 1. Choose Z_i to be a subset of $V_i \cup (V_j - Z_j)$ with exactly $\lfloor \sqrt{k-c} \rfloor$ vertices. (There is such a subset because $|V_i \cup (V_j - Z_j)| = |V_i| + |V_j| - \lfloor \sqrt{k-c} \rfloor \geq \lfloor \sqrt{k-c} \rfloor$.)

In Phase 1, clearly the assigned sets are pairwise disjoint. It is also clear that a set assigned in Phase 2 does not intersect a set assigned in Phase 1. By Fact 4.4, two sets assigned in Phase 2 cannot be associated with adjacent sets V_y. So they are disjoint.

Because the union of the sets Z_i is a subset of V, we must have $(t-l)\lfloor \sqrt{k-c} \rfloor + l \leq n$, where, as before, t is the number of nonroot nodes of Γ. This means

$$\begin{aligned} l &\geq \frac{\lfloor \sqrt{k-c} \rfloor}{\lfloor \sqrt{k-c} \rfloor - 1} t - \frac{1}{\lfloor \sqrt{k-c} \rfloor - 1} n \\ &= \left(1 + \frac{1}{\lfloor \sqrt{k-c} \rfloor - 1}\right) t - \frac{1}{(\lfloor \sqrt{k-c} \rfloor - 1)} n, \end{aligned}$$

concluding the proof of the lemma. ∎

We showed how to modify the algorithm so that it outputs at each phase a $2(i-1)$-tree-carving, denoted simply by Γ_i, of G with respect to S_{i-1} and having $|B_i|$ arcs. Recall that l_i is the number of b_e which are leaves of F_i. Recall that b_e is the bottom endpoint of $t_e \in T_i$. If b_e is a leaf of F_i then t_e is the only edge of F_i incident to b_e, which means that $\{b_e\}$ is the vertex set of a connected component of $(V, F_i - T_i)$. In other words, there is an x such that $V_x = \{b_e\}$. Finally, note that V_x is a nonroot leaf of Γ_i. All this implies that Γ_i has at least l_i nonroot leaves.

Applying Lemma 4.5 to Γ_i, we get the following inequality,

$$l_i \geq \left(1 + \frac{1}{\lfloor \sqrt{k - 2(i-1)} \rfloor - 1}\right) |B_i| - \frac{1}{(\lfloor \sqrt{k - 2(i-1)} \rfloor - 1)} n, \quad (2)$$

because $t = |E(\Gamma_i)| = |B_i|$.

We also have, for $1 \leq i \leq j$,

$$l_i \geq \left(1 + \frac{1}{\lfloor \sqrt{k - 2(j-1)} \rfloor - 1}\right)|B_i| - \frac{1}{\left(\lfloor \sqrt{k - 2(j-1)} \rfloor - 1\right)} n. \quad (3)$$

Using (2), (3), and Lemma 4.3 for OPT, we conclude the new upper bounds on $|B_i|$.

COROLLARY 4.6. *We have*

$$|B_i| \leq \frac{k\left(\lfloor \sqrt{k - 2(i-1)} \rfloor - 1\right) + 2(i-1)}{k\left((k-(i-1))\left(\lfloor \sqrt{k-2(i-1)} \rfloor - 1\right) + (i-1)\right)} \text{opt}, \quad (4)$$

and, for any $1 \leq i \leq j$,

$$|B_i| \leq \frac{k\left(\lfloor \sqrt{k - 2(j-1)} \rfloor - 1\right) + 2(i-1)}{k\left((k-(i-1))\left(\lfloor \sqrt{k-2(j-1)} \rfloor - 1\right) + (i-1)\right)} \text{opt}. \quad (5)$$

Using these new upper bounds, we derive our final result.

THEOREM 4.7. *The performance ratio of Khuller and Raghavachari's algorithm, for large enough* k, *is at most* 1.7.

Proof. We will give the proof for even k. The proof for odd k is similar.

We start by using, for some fixed $r \geq 2$, bound (4) for $B_{k/2-r+1}, \ldots, B_{k/2}$. For the remaining sets B_i, we use bound (5) with $j = k/2 - r$. From this, we infer, for even k, that

$$\frac{|S_{k/2}|}{\text{opt}} = \frac{\sum_{i=1}^{k/2}(|F_i| + |B_i|)}{\text{opt}} \leq 1 + \sum_{i=1}^{j} \frac{|B_i|}{\text{opt}} + \sum_{i=j+1}^{k/2} \frac{|B_i|}{\text{opt}}$$

$$\leq 1 + \sum_{i=j+1}^{k/2} \frac{k\left(\lfloor \sqrt{k - 2(i-1)} \rfloor - 1\right) + 2(i-1)}{k\left((k-(i-1))\left(\lfloor \sqrt{k-2(i-1)} \rfloor - 1\right) + (i-1)\right)}$$

$$+ \sum_{i=1}^{j} \frac{k\left(\lfloor \sqrt{k - 2(j-1)} \rfloor - 1\right) + 2(i-1)}{k\left((k-(i-1))\left(\lfloor \sqrt{k-2(j-1)} \rfloor - 1\right) + (i-1)\right)}$$

$$\leq 1 + \sum_{i=j+1}^{k/2} \frac{k\lfloor \sqrt{2r} \rfloor - 2}{k\left((k-(i-1))\left(\lfloor \sqrt{k-2(i-1)} \rfloor - 1\right) + (i-1)\right)}$$

$$+ \sum_{i=1}^{j} \frac{k\left(\lfloor \sqrt{2(r+1)} \rfloor - 1\right) + 2(i-1)}{k\left((k-(i-1))\left(\lfloor \sqrt{2(r+1)} \rfloor - 1\right) + (i-1)\right)}$$

$$\leq 1 + \sum_{i=j+1}^{k/2} \frac{2(k\lfloor\sqrt{2r}\rfloor - 2)}{k(k-2r)}$$

$$+ \sum_{i=1}^{j} \frac{k\left(\left\lfloor\sqrt{2(r+1)}\right\rfloor - 1\right) + 2(i-1)}{k\left((k-(i-1))\left(\left\lfloor\sqrt{2(r+1)}\right\rfloor - 1\right) + (i-1)\right)}.$$

Denoting $\lfloor\sqrt{2(r+1)}\rfloor$ by C_r, just to simplify the formulas, we have

$$\frac{|S_{k/2}|}{\text{opt}} \leq 1 + r\frac{2(k\lfloor\sqrt{2r}\rfloor - 2)}{k(k-2r)}$$

$$+ \sum_{i=1}^{j} \frac{k(C_r - 1) + 2(i-1)}{k((k-(i-1))(C_r - 1) + (i-1))}$$

$$\leq 1 + \frac{2r(k\lfloor\sqrt{2r}\rfloor - 2)}{k(k-2r)} - \frac{2j}{k(C_r-2)}$$

$$+ \sum_{i=1}^{j}\left(\frac{2}{k(C_r-2)} + \frac{k(C_r - 1) + 2(i-1)}{k((k-(i-1))(C_r - 1) + (i-1))}\right)$$

$$\leq 1 + \frac{2r(k\lfloor\sqrt{2r}\rfloor - 2)}{k(k-2r)} - \frac{2j}{k(C_r-2)}$$

$$+ \frac{C_r(C_r-1)}{(C_r-2)^2}\left(\ln\frac{q'-1}{q'-1-j} + \varepsilon\right)$$

$$= 1 + \frac{2r(k\lfloor\sqrt{2r}\rfloor - 2)}{k(k-2r)} - \frac{k-2r}{k(C_r-2)}$$

$$+ \frac{C_r(C_r-1)}{(C_r-2)^2}\left(\ln\frac{q'-1}{q'-1-k/2+r} + \varepsilon\right),$$

where $q' = ((C_r - 1)/(C_r - 2))k$ and $\varepsilon > 0$. (For the last equality, we used that $j = k/2 - r$.) Note that C_r is a function solely of r so, as k goes to infinity, the bound on the ratio converges to

$$1 - \frac{1}{C_r - 2} + \frac{C_r(C_r-1)}{(C_r-2)^2}\ln\frac{(C_r-1)/(C_r-2)}{(C_r-1)/(C_r-2) - 1/2}$$

$$= 1 - \frac{1}{C_r - 2} + \frac{C_r(C_r-1)}{(C_r-2)^2}\ln\frac{2(C_r-1)}{C_r}.$$

Now, let us make r goes to infinity, which implies that C_r goes to infinity as well. The foregoing expression converges to $1 + \ln 2 < 1.7$. From this, we finally conclude that, for large enough k, the performance ratio of the algorithm is at most 1.7. ▮

5. THE COMPLEXITY OF THE PROBLEM

In this section, we show that the minimum size 2-edge-connected spanning subgraph problem is MAX SNP-hard. This means that there is a constant $\varepsilon > 0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at most $1 + \varepsilon$ implies that $P = NP$, by results of Arora *et al.* [1].

As in [17], we use the concept of L-reduction, which is a special kind of reduction that preserves approximability. Let A and B be two optimization problems. We say A *L-reduces to* B if there are two polynomial-time algorithms f and g, and positive constants α and β, such that for each instance I of A,

1. Algorithm f produces an instance $I' = f(I)$ of B, such that the optima of I and I', of costs denoted $\mathrm{opt}_A(I)$ and $\mathrm{opt}_B(I')$, respectively, satisfy $\mathrm{opt}_B(I') \le \alpha \cdot \mathrm{opt}_A(I)$, and
2. Given any feasible solution of I' with cost c', algorithm g produces a solution of I with cost c such that $|c - \mathrm{opt}_A(I)| \le \beta \cdot |c' - \mathrm{opt}_B(I')|$.

THEOREM 5.1. *The minimum size 2-edge-connected spanning subgraph problem is MAX SNP-hard.*

Proof. Denote by VC_7 the vertex cover problem restricted to graphs with maximum degree 7. Papadimitriou and Yannakakis [17] showed that VC_7 is MAX SNP-hard. We prove that VC_7 L-reduces to the minimum size 2-edge-connected spanning subgraph problem, here denoted by 2-MECSS. The reduction comes from [11], where a directed version of the 2-MECSS is proved to be MAX SNP-hard.

The first part of the L-reduction is a polynomial-time algorithm f and a constant α. Given any instance G of VC_7, f produces an instance H of the 2-MECSS such that the minimum number of edges in a 2-edge-connected spanning subgraph of H, denoted by $\mathrm{opt}_{2\text{-MECSS}}(H)$, is at most α times the minimum size of a vertex cover in G, denoted by $\mathrm{opt}_{VC_7}(G)$. In other words, $\mathrm{opt}_{2\text{-MECSS}}(H) \le \alpha \cdot \mathrm{opt}_{VC_7}(G)$.

Let us describe algorithm f. Consider an instance G of VC_7. G is a graph with maximum degree 7. Here is a procedure to construct an instance H of the 2-MECSS. Similarly to [11], start with a special vertex, the root. Each vertex in G will have a "current vertex," initially the root. For each edge uv, add a "cover-testing gadget" to H, as illustrated in Fig. 4(a). Specifically, add six new vertices $x_1, x_2, x_3, y_1, y_2, y_3$. Vertex x_2 is adjacent only to vertices x_1 and x_3. Analogously, vertex y_2 is adjacent only to vertices y_1 and y_3. There is an edge labeled u^- between x_1 and y_3, and an edge labeled v^- between y_1 and x_3. There is one more edge incident to x_1: an edge labeled u^+ between x_1 and the current vertex of u; and one more edge incident to y_1: an edge labeled v^+ between y_1 and the current vertex of v. Make y_3 the new current vertex of u, and make x_3 the new current vertex of v. Finally, after all edges of G have been considered, for each vertex v in G, add an edge labeled v^+ between its final current vertex and the root. The gadgets are implicitly numbered in the order we have added them. Clearly H can be obtained in polynomial time in the size of G. This completes the description of f.

The next fact will be used in the proof of the existence of a constant α. Let m be the number of edges in G, and let s be a positive integer.

FACT 5.2. *If G has a vertex cover with at most s vertices, then H has a 2-edge-connected spanning subgraph with at most $6m + s$ edges.*

Proof. Suppose G has a vertex cover S with at most s vertices. Consider the spanning subgraph H' of H with all edges incident to vertices of type x_2 and y_2, all edges labeled u^+ for $u \in S$, and all edges labeled u^- for $u \notin S$. Observe that all vertices in H', but the root, have degree 2. Besides, the only possible cycles in H' are of two types: either they are like $x_1x_2x_3y_1y_2y_3$, or they include the root. Because S is a vertex cover, no cycles of type $x_1x_2x_3y_1y_2y_3$ can occur. This implies that H is a collection of cycles intersecting only at the root, and spanning all vertices. Therefore, H' is a 2-edge-connected spanning subgraph of H. To count how many edges there are in H', just notice that H' has six edges per gadget plus one extra edge per vertex in S. Therefore, in total, H' has at most $6m + s$ edges. ∎

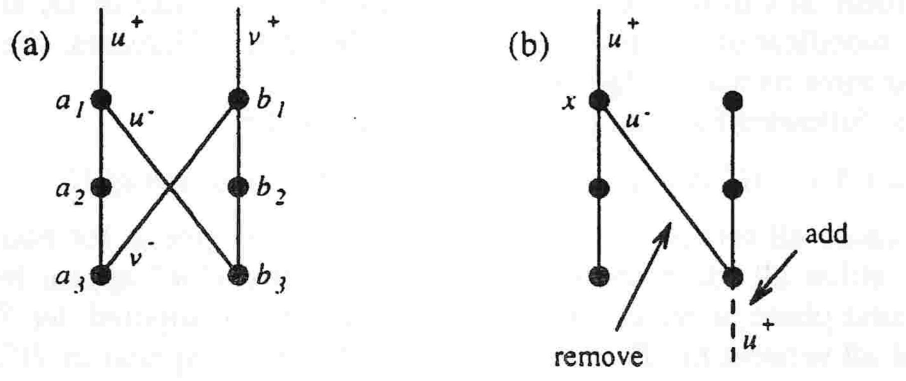


FIG. 4. (a) The gadget for edge uv. (b) The modification of the gadget for edge uv.

LEMMA 5.3. $\text{opt}_{\text{2-MECSS}}(H) \leq 43 \cdot \text{opt}_{VC_7}(G)$.

Proof. By Fact 5.2, $\text{opt}_{\text{2-MECSS}}(H) \leq 6m + \text{opt}_{VC_7}(G)$. Also, because G has maximum degree 7, $\text{opt}_{VC_7}(G) \geq m/7$. Putting these together, we get that

$$\begin{aligned}\text{opt}_{\text{2-MECSS}}(H) &\leq 6m + \text{opt}_{VC_7}(G) \leq (6 \cdot 7 + 1)\text{opt}_{VC_7}(G) \\ &= 43 \cdot \text{opt}_{VC_7}(G).\end{aligned}$$

▮

So, we can consider $\alpha = 43$.

The second part of the L-reduction is the constant β and the algorithm g. We may assume that $m \geq 1$. Note that H has $6m + 1$ vertices, therefore any 2-edge-connected spanning subgraph of H must have at least $6m + 1$ edges. Consider a 2-edge-connected spanning subgraph H' of H with $6m + s$ edges, where s is some positive integer. Algorithm g will produce in polynomial time a vertex cover of size at most s, from H'. From this, and from Fact 5.2, we will have that $\text{opt}_{\text{2-MECSS}}(H) = 6m + \text{opt}_{VC_7}(G)$. In addition, then $|6m + s - \text{opt}_{\text{2-MECSS}}(H)| = |6m + s - 6m - \text{opt}_{VC_7}(G)| = |s - \text{opt}_{VC_7}(G)|$, meaning that $\beta = 1$ suffices.

So let us see how algorithm g works. In a first phase, g produces another 2-edge-connected subgraph H'' of H with at most as many edges as H', and such that in H'' all vertices, but the root, have degree 2. To get H'', each gadget is checked for vertices of degree 3 (all vertices in H, but the root, have degree at most 3). If a vertex x has degree 3 in H', then the edges labeled u^+ and u^- incident to x appear in H'. Remove the edge labeled u^- incident to x and add (if not already there) the edge labeled u^+ incident to the vertex adjacent to x through the edge labeled u^-. See Fig. 4(b).

H'' is the graph obtained after applying this modification while there are vertices, other than the root, of degree 3. We can make sure the modification is applied to the lowest numbered gadget, and inside the gadget, to x_1, y_1, x_3, y_3, in this order. This procedure takes polynomial time: the modification can be done in polynomial time in the size of G, and after each modification the number of edges labeled u^- decreases. Clearly H'' has at most as many edges as H'.

The following fact can be verified by the reader.

FACT 5.4. *H'' is a 2-edge-connected spanning subgraph of H.*

Because all vertices in H'', but the root, have degree 2, for each vertex in G, either all edges labeled u^+ or all edges labeled u^- appear in H''. In a second phase, a vertex cover of size at most s is computed: let S be the set of all vertices in H'' whose all edges labeled u^+ appear in H''. In any gadget, there must be edges labeled u^+_i appearing in H'', otherwise H'' is not connected. Thus S is a vertex cover in G. Recall that H' had $6m + s$ edges. Hence H'' has at most $6m + s$ edges. But because all vertices but the root have degree 2, H'' has $6m + t$ edges, for $t \leq s$, and t is the number of vertices in S. This finishes the description of algorithm g, and the proof of Theorem 5.1. ▮

ACKNOWLEDGMENTS

I thank Vijay Vazirani for suggesting the problem, and S. Khuller for the main reference used on the MAX SNP-hardness proof. I am also grateful to Gruia Calinescu, Naveen Garg, Ion Mandoiu, and Robin Thomas for helpful discussions. I thank the anonymous referees and Howard Karloff for their comments and suggestions.

REFERENCES

1. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, Proof verification and hardness of approximation problems, *in* "Proceedings of the Thirty-Third IEEE Symposium on Foundations of Computer Science," 1992, pp. 14–23.
2. J. Cheriyan, M.-Y. Kao, and R. Thurimella, Algorithms for parallel k-vertex connectivity and sparse certificates, *SIAM J. Comput.* **22**(1) (1993), 157–174.
3. J. Cheriyan and R. Thurimella, Approximating minimum-size k-connected spanning subgraph via matching, *in* "Proceedings of the Thirty-Seventh IEEE Symposium on Foundations of Computer Science," Oct. 1996.
4. H. N. Gabow, M. X. Goemans, and D. P. Williamson, An efficient approximation algorithm for the survivable network design problem, *in* "Proceedings of the Third Integer Programming and Combinatorial Optimization Conference," 1993, pp. 57–74.
5. M. R. Garey and D. S. Johnson, "Computers and Intractability," Freeman, New York, 1979.
6. N. Garg, V. Santosh, and A. Singla, Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques, *in* "Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms," 1993, pp. 103–111.
7. M. X. Goemans, A. V. Goldberg, S. Plotkin, D. Shmoys, E. Tardos, and D. P. Williamson, Improved approximation algorithms for network design problems, *in* "Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms," 1994, pp. 223–232.
8. M. X. Goemans and D. P. Williamson, A general approximation technique for constrained forest problems, *SIAM J. Comput.* **24**(2) (1995), 296–317.
9. D. Karger, Random sampling in cut, flow, and network design problems, *in* "Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing," 1994, pp. 648–657.
10. S. Khuller and B. Raghavachari, Improved approximation algorithms for uniform connectivity problems, *J. Algorithms* **21** (1996), 434–450.
11. S. Khuller, B. Raghavachari, and N. Young, Approximating the minimum equivalent digraph, *SIAM J. Comput.* **24**(4) (1995), 859–872.
12. S. Khuller and R. Thurimella, Approximation algorithms for graph augmentation, *J. Algorithms* **14**(2) (1993), 214–225.

13. S. Khuller and U. Vishkin, Biconnectivity approximations and graph carvings, *J. Assoc. Comput. Mach.* **41**(2) (1994), 214–235.
14. W. Mader, Minimale n-fach kantenzusammenhängende Graphen, *Math. Ann.* (1971), 21–28.
15. W. Mader, Ecken vom Grad n in minimalen n-fach zusammenhängenden Graphen, *Arch. Math. (Basel)* **23** (1972), 219–224.
16. H. Nagamochi and T. Ibaraki, Linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph, *Algorithmica* **7**(5-6) (1992), 583–596.
17. C. H. Papadimitriou and M. Yannakakis, Optimization, approximation, and complexity classes, *J. Comput. Syst. Sci.* **43** (1991), 425–440.
18. R. Ravi and D. Williamson, An approximation algorithm for minimum-cost vertex-connectivity problems, *in* "Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms," 1995, pp. 332–341.

Available at
www.ElsevierMathematics.com
POWERED BY SCIENCE DIRECT

Journal of Combinatorial Theory, Series A 105 (2004) 185–206

Journal of Combinatorial Theory Series A

http://www.elsevier.com/locate/jcta

From monomials to words to graphs

Cristina G. Fernandes,[a,1] Edward L. Green,[b,2] and Arnaldo Mandel[a,3]

[a] *Departamento de Ciência da Computação, Universidade de São Paulo, São Paulo, SP, 05508-970, Brazil*
[b] *Mathematics Department, Virginia Tech University, USA*

Received 26 February 2003

Communicated by Bernd Sturmfels

Abstract

Given a finite alphabet X and an ordering $\prec$ on the letters, the map $\sigma_{\prec}$ sends each monomial on X to the word that is the ordered product of the letter powers in the monomial. Motivated by a question on Gröbner bases, we characterize ideals I in the free commutative monoid (in terms of a generating set) such that the ideal $\langle \sigma_{\prec}(I) \rangle$ generated by $\sigma_{\prec}(I)$ in the free monoid is finitely generated. Whether there exists an $\prec$ such that $\langle \sigma_{\prec}(I) \rangle$ is finitely generated turns out to be NP-complete. The latter problem is closely related to the recognition problem for comparability graphs.

Keywords: Gröbner bases; Polynomials; Comparability graphs; NP-complete; Blocking polyhedron

1. Introduction

An important structural difference between commutative and noncommutative free monoids is that in a finitely generated free commutative monoid all ideals are finitely generated (Dickson's Lemma), while this is not the case for free monoids (for instance, the ideal generated by $\{xy^nx \mid n \geqslant 0\}$). We will consider questions about

E-mail addresses: cris@ime.usp.br (C.G. Fernandes), green@math.vt.edu (E.L. Green), am@ime.usp.br (A. Mandel).

[1] Research MCT/CNPq through ProNEx Programme (Proc. CNPq 664107/1997–4).

[2] Partially supported by CNPq/NSF Grant no. 910123/99.

[3] Partially supported by CNPq/NSF Grant no. 910123/99 and MCT/CNPq through ProNEx Programme (Proc. CNPq 664107/1997–4).

whether some ideals of the free monoid are finitely generated. Those ideals will be described starting with an ideal in a free commutative monoid.

Let X be a finite alphabet (of *letters*). Denote by $[X]$ the free commutative monoid on X and by X^* the free monoid on X; we call the members of $[X]$ *monomials*, and the members of X^* *words*. When convenient, we assume $X = \{x_1, x_2, \dots, x_n\}$, so a monomial can be written multiplicatively as $x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$. We denote by π the canonical monoid epimorphism $X^* \to [X]$.

The most natural relation between ideals of $[X]$ and X^* is given by the canonical map, so the first question is:

Problem 1. Given an ideal I of $[X]$, is $\pi^{-1}(I)$ finitely-generated?

This first question, while quite natural, seems to be of limited interest, as ideals are not such big players in the structure theory of monoids.

The next questions were motivated by the study of noncommutative presentations of affine algebras and their Gröbner bases. In fact, the problems studied in this paper were motivated by a question posed by Bernd Sturmfels on the finite generation of monomial ideals. We postpone the discussion until Section 2, as the questions can be completely understood within the context of monoids.

For each ordering $\prec$ of the letters, we define a section $\sigma_\prec$ of π as follows. We say that a word is *sorted* if its letters occur in it in increasing order; if $x_1 \prec x_2 \prec \cdots \prec x_n$, such a word can be uniquely written as $x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$. Let $\sigma_\prec : [X] \to X^*$ be the function mapping each monomial m to the unique sorted word in $\pi^{-1}(m)$. So, $\pi\sigma_\prec$ is the identity map on $[X]$, and we define the *sorting map* $S_\prec$ on X^* by $S_\prec = \sigma_\prec \pi$. The subscript $\prec$ will be omitted when implicitly understood.

We will mainly be concerned with ideals of form $\mathscr{I}_\prec(I) = \langle \sigma_\prec(I) \rangle$, that is, the ideal generated by sorted words corresponding to a commutative ideal.

Problem 2. Given an ideal I of $[X]$ and an ordering $\prec$ of X, is $\mathscr{I}_\prec(I)$ finitely generated?

While it is convenient to assume an ordering on the letters so that one can write monomials, that ordering is not part of $[X]$. So, we also consider

Problem 3. Given an ideal I of $[X]$, is there an ordering $\prec$ of X so that $\mathscr{I}_\prec(I)$ is finitely generated?

These problems are very loosely posed, as it is not specified how each ideal is given. We are interested in specification by finite data, so that it makes sense to look for an algorithmic answer to each of the problems. We consider three forms of specifying an ideal of $[X]$: by a finite generating set, as the inverse image of an ideal under a morphism from $[X]$, and as the initial ideal of a polynomial ideal given by its generators. The latter is explained in Section 2; our main results relate to the other two.

Most of this paper will be concerned with a finite set of monomials and the ideal it generates. Let us denote by $\langle M \rangle$ the monoid ideal generated by a set M. So, $\langle M \rangle = M[X]$ if $M \subseteq [X]$, and $\langle M \rangle = X^*MX^*$ if $M \subseteq X^*$. From now on, except where explicitly stated, in each of the problems above, *given an ideal I* is reinterpreted as *given a finite set M of monomials, let $I = \langle M \rangle$*. That is the first of the three forms mentioned above.

It is not clear a priori that either problem is even decidable. Section 3 shows that standard methods of Automata Theory suffice to decide each instance of the problems, when the ideal is given by generators. However, this is unsatisfying both mathematically and computationally. From a purely mathematical viewpoint, the automata decision process is so far removed from the initial data that one learns very little about the underlying structure. From the computational viewpoint, what goes wrong is the exponential complexity of the algorithms thus obtained. We heed the fact that a monomial can be given as a vector of exponents, so the run-time of algorithms for the corresponding decision problems should be measured relative to the bit size of the exponents. With that in mind, we summarize the main results:

Problem 1 is solved completely in Section 7. The characterization we obtain yields a polynomial algorithm.

Problem 2 is the central one here, and also has a definite answer. To describe it we need more notation. We suppose an ordering of X is given. For w in $[X]$, say that a letter is *extremal* in w if it is the smallest or the largest letter with a positive exponent there and say that a letter is *internal* to w if it lies strictly between the extremal letters. Notice that an internal letter is not required to occur in w; for instance, using the ordering implied by the indices, the internal letters of $x_2^3x_3x_5^2x_7$ are x_3, x_4, x_5, and x_6. Also denote by $w\backslash x$ the monomial resulting of evaluating x to 1 in the monomial w. In particular, $x^m\backslash x = 1$. A collection of monomials is an *antichain* if no one divides another; clearly, the (unique) minimal generating set of an ideal of $[X]$ is an antichain. Dickson's Lemma, quoted earlier, is equivalent to the statement that every antichain of monomials is finite. For convenience, we will shorten $\mathscr{I}(\langle M \rangle)$ to $\mathscr{I}(M)$ when M is an arbitrary set of monomials.

Theorem 1. *Let M be an antichain in $[X]$. Then, $\mathscr{I}(M)$ is finitely generated if and only if, for every w in M and x in X, there exists s in M such that x is extremal in s and $s\backslash x$ divides w.*

A proof is found in Section 4. The above result immediately yields a polynomial-time decision algorithm for Problem 2. It also implies that Problem 3 is in NP.

When M is square free, Problem 3 can be decided in polynomial time. And that is the end of good news. Problem 3 is shown in Section 6 to be NP-complete even when M consists only of quadratic monomials. So, while the automata-theoretic algorithms where unacceptable for Problems 1 and 2 because of high exponents in the data, NP-completeness of Problem 3 is not related to the possibility of writing numbers succinctly.

In Section 8, using a slightly homological flavor, we turn to another way of presenting an ideal of $[X]$: fix a homomorphism from $[X]$ to another free commutative monoid, and an ideal J in the target, and take the pre-image of that ideal. The homomorphism can be described by a matrix, and when J is given by its minimal generators the ideal of $[X]$ is described as a finite union of integer polyhedra, each one of them an ideal. The theory developed for handling generators is enough to show that Problems 1 and 2 become coNP-complete with this data, while Problem 3 is shown to be NP-hard.

2. Connections with Gröbner bases

Problems 2 and 3 stem from a connection between the commutative and noncommutative Gröbner bases theories. Let K denote a field, $K[X]$ the commutative polynomial ring on the finite set X, and $K\langle X\rangle$ the free associative algebra on the same set (we adhere to the terminology of the first section, and talk about *letters* instead of variables). The linear extension of the monoid morphism π is a ring morphism $K\langle X\rangle \to K[X]$, still denoted by π; its kernel is generated by the commutation relations $\mathscr{C} = \{xy - yx \mid x, y \in X\}$. Also, given an ordering of X, the linear extension of the maps σ and S will be denoted by the same symbols.

Throughout this section, I is an ideal of $K[X]$ and $J = \pi^{-1}(I)$. It is occasionally useful to lift a commutative ring presentation $K[X]/I$ to a noncommutative presentation $K\langle X\rangle/J$ through π. This has been used in [1,2,10,15] for homological computations.

Proposition 2. *Let $A \subseteq \langle X\rangle$ be a set of noncommutative polynomials. The following are equivalent*:

(i) *$\mathscr{C} \cup A$ generates J.*
(ii) *$\pi(A)$ generates I.*
(iii) *For some ordering $\prec$ of X, $\mathscr{C} \cup S_{\prec}(A)$ generates J.*
(iv) *For any ordering $\prec$ of X, $\mathscr{C} \cup S_{\prec}(A)$ generates J.*

In particular, for any ordering $\prec$ of X, $\mathscr{C} \cup \sigma_{\prec}(I)$ generates J.

Proof. Suppose that $\mathscr{C} \cup A$ generates J. Then, $\langle \pi(A)\rangle = \langle \pi(\mathscr{C} \cup A)\rangle = \pi\langle \mathscr{C} \cup A\rangle = \pi(J) = I$. Conversely, suppose that $\pi(A)$ generates I. Since $\ker\pi \subseteq \langle \mathscr{C} \cup A\rangle \subseteq \pi^{-1}(I)$ and $\pi\langle \mathscr{C} \cup A\rangle = I$, it follows that $\langle \mathscr{C} \cup A\rangle = J$.

The equivalence of conditions (iii) and (iv) to the previous ones follows from the observation that $\pi S_{\prec}(A) = \pi(A)$. The last observation is immediate, as $\pi\sigma_{\prec}(I) = I$. $\square$

In many applications, one wants to describe a Gröbner basis for J, preferably related to a Gröbner basis for I. Let us recall quickly what those bases are (see [9,13] for an introduction to the subject). We will say *term* to mean either "word" or "monomial", so we can treat commutative and noncommutative polynomials simultaneously.

A necessary ingredient for a Gröbner bases theory is to fix a *term order*: a total order on the terms, compatible with multiplication and with 1 as minimum, and with no infinite descending chains. In this case, every polynomial has an *initial term*, the maximum term in its support, and to every ideal I one associates its *initial ideal* $In(I)$, the set of all initial terms of polynomials in I. The initial ideal is an ideal of the monoid of terms. A *Gröbner basis* for I is a subset B of I such that the initial terms of the members of B generate $In(I)$.

Eisenbud, et al. [5] took the following approach to relate Gröbner bases for commutative polynomial ideals and their pre-images in the free algebra. Start with a term order $\prec$ on $[X]$, and define its *lexicographic extension*, still denoted here by $\prec$, to X^* by: $u \prec v$ if $\pi(u) \prec \pi(v)$ or $\pi(u) = \pi(v)$ and u precedes v lexicographically, according to the $\prec$ ordering on X.

Proposition 3. *The initial ideal $In(J)$ is generated by*

$$\{xy \mid x, y \in X,\ x \succ y\} \cup \sigma(In(I)).$$

Proof. Since the noncommutative order extends the commutative one, if $p \in K[X]$, the initial term of $\sigma(p)$ is $\sigma(m)$, where m is the initial term of p. Also, from the lexicography, if $x \succ y$ are letters, xy is the initial term of $xy - yx$. The result now follows from Proposition 2. $\square$

It follows from Dickson's Theorem that every ideal of $[X]$ is finitely generated, hence every ideal of $K[X]$ has a finite Gröbner basis. In contrast, not every ideal of X^* is finitely generated, so it is generally interesting to detect whether a given ideal of $K\langle X \rangle$ has a finite Gröbner basis. The preceding proposition implies that J has a finite Gröbner basis with respect to $\succ$ if and only if $\mathscr{I}(In(I))$ is finitely generated. This gives rise to Problem 2.

A word in an ideal of X^* is a minimal generator of that ideal if and only if the words obtained by erasing either the first or the last letter are not in the ideal. Combining it with Proposition 2, we get the next result; it is Theorem 2.1 of [5], stripped of the ring theoretic context (which is handled by Proposition 3):

Theorem 4. *If M is an antichain of monomials, then the minimal generating set of $\mathscr{I}(M)$ is*

$$\{\sigma(mu) | m \in M, u \in [X] \text{ is generated by letters internal to } m,$$
$$\text{and is such that, for each letter } x \text{ extremal to } mux^{-1} \notin \langle M \rangle\}.$$

There seems to be no immediate characterization from the above for when J has a finite Gröbner basis. A sufficient condition is provided in [5], and then the question is finessed: it is shown that, if K is infinite, then, for any I and $\prec$, J will have a finite Gröbner basis after a generic change of variables.

This is of limited computational use if high degree polynomials are being handled, since the supports can grow explosively after generic changes. So, it is still of some interest to detect whether $\pi^{-1}(I)$ has a finite Gröbner basis when I is still expressed in the given coordinates.

If I is generated by a set M of monomials (this is called a *monomial ideal* of $K[X]$), then $In(I) = \langle M \rangle$, irrespective of the term order $\prec$. It follows from Proposition 3 that:

Proposition 5. *Let $M \subseteq [X]$ and let I be the monomial ideal it generates. Then J has a finite Gröbner basis with respect to the lexicographic extension of a term order of $[X]$ if and only if, for the same ordering of X, $\mathscr{I}(M)$ is finitely generated.*

This gives rise to Problem 3.

A similar looking question is, in the notation above, what conditions must M satisfy, so that J has a finite Gröbner basis with respect to the lexicographic extension of any term order of $[X]$? Proposition 5 translates it to a problem of monomials and words, and the answer is in Section 5, Theorem 16.

As a final note, we point out that within this context another way of specifying an ideal of $[X]$ is relevant to Problems 2 and 3. Namely, suppose a term order is given on $[X]$; given a finite set M of polynomials, consider the initial ideal I of the polynomial ideal generated by M. That is the actual motivation for those problems, after all! The usual process of going from M to I is Buchberger's algorithm and its variants. These are all of high complexity, so the question remains whether Problem 2 can be solved efficiently from this data.

3. Using automata

It is not clear from the outset that either of the problems mentioned in the introduction is decidable. This can be shown to be the case by means of the traditional machinery of automata theory (we follow the notation and terminology of [12]). We do it here, mostly for completeness and to underline some of the complexity issues. This section can be skipped, with no loss in understanding of the remaining text.

Suppose J is an ideal of the free monoid X^*; then its unique minimal generating set is $T = J \backslash (XJ \cup JX)$. Hence, if J is a regular language, so is T. The problem of deciding whether T is finite, given a regular expression for J, can be whimsically, although not very accurately, related to two well-known Unix utility programs: given a pattern for a `grep` search, decide whether the same search can be made by `fgrep`.

Problems 1 and 2 refer to ideals that are regular languages. Consider Problem 1. It is a well-known (although nonconstructive) consequence of Higman's Theorem [11] that, for every subset A of $[X]$, $\pi^{-1}(A)$ is regular. For $\pi^{-1}\langle M \rangle$, one can construct a deterministic automaton directly: have n parallel counters, one for each letter and counting up to its maximum degree in M. Each state of the automaton corresponds to an n-tuple of values for the counters, and processing a word x leads to a state whose counters correspond to the exponent vector of $\pi(x)$. The final states are those that show that $\pi(x) \in \langle M \rangle$, and they can be colluded into a single state that is never left after being reached.

For Problem 2, we can write a simple regular expression for $\pi^{-1}\langle M \rangle$. If $w = x_1^{i_1} x_2^{i_2} \cdots x_{n-1}^{i_{n-1}} x_n^{i_n} \in [X]$, then $\mathscr{I}(w) = X^* x_1^{i_1} x_2^{i_2} x_2^* \cdots x_{n-1}^{i_{n-1}} x_{n-1}^* x_n^{i_n} X^*$; and $\mathscr{I}(M) = \bigcup_{w \in M} \mathscr{I}(w)$, so $\mathscr{I}(M)$ is a regular language. A deterministic automaton recognizing $\mathscr{I}(M)$ can also be constructed using parallel bounded counters, although the description would be more complicated than the previous one.

In both cases, there is only one final state, from which no transition leaves. This makes it easy to construct a deterministic automaton for the minimal generating set of each ideal, with direct products of three very similar automata. Then finiteness can be easily checked by a graph search. This approach shows now that Problem 2 is decidable.

From a complexity viewpoint, this does not work. Even though we have scrupulously avoided using nondeterministic automata, there remains a source of exponential complexity: in either case, the automaton described for each ideal has a number of states that is roughly the product N of the maximum degrees of letters in M. This is too large; since a monomial can be represented as a vector of exponents, a reasonable encoding for M would have only $O(n \log N)$ bits, where $n = |X|$. So, the automaton for the minimal generating set has exponentially many states, and the graph search is linear in the number of states.

The solutions we present in the following sections could perhaps be retro-fitted into an automata-theoretical framework. Actually, thinking of automata helped in the discovery of those results: the Pumping Lemma (see [12, Section 2.4]) was a starting point.

There is a rich literature on ideals of the free monoid, from the viewpoint of language theory, with a twist. Instead of concentrating on the ideal, the focus is on its complement. The complement of an ideal is said to be a *factorial language*, and the minimal generators of the ideal appear as *forbidden subwords* in this context. There are many algorithms for problems involving factorial languages (see [3,4]), but they usually take a deterministic automaton like the large ones we described as input, so they are of no use here.

4. When $\mathscr{I}(M)$ is finitely generated

We assume a fixed ordering $\prec$ on the alphabet X. The *support* of a monomial w, denoted $\underline{w}$, is the set of letters with nonzero exponent in w. So, $\min(\underline{w})$ and $\max(\underline{w})$ are the *extremal* letters of w, while the letters x such that $\min(\underline{w}) \prec x \prec \max(\underline{w})$ are *internal* to w. We will use the notation $u|v$ meaning *u divides v*, both in $[X]$ and X^*. So, for monomials $u = x_1^{i_1} \ldots x_n^{i_n}, v = x_1^{j_1} \ldots x_n^{j_n}$, $u \mid v$ means that $i_1 \leqslant j_1, \ldots, i_n \leqslant j_n$; for words u, v, $u \mid v$ means that there exist words w, z such that $v = wuz$.

Example 1. Take $X = \{a, b, c, \dots\}$ with the usual ordering. Then, for $w = b^2df$, $\min(\underline{w}) = b$, $\max(\underline{w}) = f$, and the internal letters are c, d, and e.

Lemma 6. *If u and v are sorted words such that $u \mid v$, then for any x in X, either $u \mid S(vx)$ or $S(ux) \mid S(vx)$.*

Proof. The first case occurs if $x \leqslant \min(\underline{u})$ or $x \geqslant \max(\underline{u})$, and the second case occurs if $\min(\underline{u}) \leqslant x \leqslant \max(\underline{u})$. $\square$

Lemma 7. *Let T be a set of sorted words. If, for every t in T and x in X, $S(tx)$ has a factor in T, then there exists an ideal I of $[X]$ such that $T \subseteq \sigma(I)$ and $\langle T \rangle = \mathscr{I}(I)$.*

Proof. Let $M = \pi(T)$ and $I = \langle M \rangle$; clearly $T \subseteq \sigma(I)$, and we will show that $\langle T \rangle = \mathscr{I}(I)$.

Since $\langle T \rangle \subseteq \mathscr{I}(I)$ is clear, it is enough to show that $\sigma(I) \subseteq \langle T \rangle$. Since any monomial in I can be written as su, with $s \in M$, $u \in [X]$, we will show that $\sigma(su)$ has a factor in T (so, it is in $\langle T \rangle$) by induction in the total degree of u.

There is nothing to prove if the degree is zero. So, we can write $u = vx$, with $v \in [X]$, $x \in X$. By the induction hypothesis, $t \mid \sigma(sv)$ for some $t \in T$. By hypothesis, $S(tx)$ has a factor $y \in T$. Noticing that $S(\sigma(sv)x) = \sigma(svx) = \sigma(su)$, Lemma 6 implies that either $t \mid \sigma(su)$, or $S(tx) \mid \sigma(su)$, in which case, $y \mid \sigma(su)$. In either case, the result follows. $\square$

From this, it easily follows:

Corollary 8. *Let M be a set of monomials. A sufficient condition for a set T of words to be such that $\langle T \rangle = \mathscr{I}(M)$ is that $\sigma(M) \subseteq T \subseteq \sigma\langle M \rangle$ and for every t in T and x in X, $S(tx)$ has a factor in T.*

We restate Theorem 1 with some additional precision, in order to prove it. First, we recall and introduce some notation.

If w is a monomial and $x \in X$, $w \backslash x$ denotes the monomial obtained from w by erasing the occurrences of x. We denote by $\imath(w)$ the set of internal letters of w, and by $\partial_x w$ the degree of x in w. Given a set M of monomials, let $r_x(M)$ denote the maximum degree x occurs with as an extremal letter in M. To avoid misunderstandings, $[\imath(w)]$ is simply the submonoid of $[X]$ generated by $\imath(w)$.

Example 2. Continuing the earlier example, $w \backslash b = df$ and $w \backslash f = b^2d$. If $M = \{c^3, a^2c^5f^2, cf^3g, a^2b^2c^2\}$, then $r_c(M) = 3$ and $r_f(M) = 2$.

Theorem 9. *Let M be an antichain in $[X]$. The following are equivalent:*

(i) *$\mathscr{I}(M)$ is finitely generated.*

(ii) *For every w in M and x in $\imath(w)$, there exists s in M such that x is extremal in s, and $s\backslash x$ divides w.*

(iii) *$\mathscr{I}(M)$ is generated by $\sigma\left(\bigcup_{w\in M}\{u\in w[\imath(w)] \mid \forall x\in\imath(w),\ \partial_x u<r_x(M)\}\right)$.*

Proof. Let T be the minimal generating set of $\mathscr{I}(M)$, and suppose it is finite. We shall prove condition (ii). Note that $\sigma(M)\subseteq T$. Indeed, if $w\in M$, $\sigma(w)$ has a factor in T, and this has the form $\sigma(su)$, for some $s\in M$ and $u\in[X]$. So, $\sigma(su)|\sigma(w)$, and this clearly implies $su \mid w$. So, $s|w$ and, since M is an antichain and $s\in M$, it follows that $s=w$, $u=1$, hence $\sigma(w)\in T$.

Let $w\in M$ and let x be an internal letter to w. We can uniquely write $\sigma(w)=ux^r v$, with u, v sorted, $\max(\underline{u})\prec x\prec\min(\underline{v})$. Hence, all words $ux^n v$, with $n\geqslant r$, are in $\mathscr{I}(M)$, so each has a factor in T; it follows that some $t\in T$ is a factor of infinitely many such words. If x is not in the support of t, it must happen that t is a factor of u or of v, hence a proper factor of $\sigma(w)$, but $\sigma(w)\in T$, so this cannot occur. Therefore, x is in the support of t, and necessarily is extremal. Without loss of generality, let us assume that $x=\min(\underline{t})$; so $t=x^k z$ with $x\prec\min(\underline{z})$, and note that z is a factor of $\sigma(w)$, so $\pi(z)|w$. Now, $t=\sigma(sy)$ for some $s\in M$ and $y\in[X]$. If $x\notin\underline{s}$, we would have $s\mid\pi(z)$, hence $s\mid w$, a contradiction. Hence $x=\min(\underline{s})$, so $s\backslash x$ is a factor of w.

Now, suppose that condition (ii) holds and let us prove (iii). Call T the generating set in that statement. Clearly $\sigma(M)\subseteq T\subseteq\sigma\langle M\rangle$. Now, let $t\in T$ and $x\in X$, and let us find a factor of $S(tx)$ in T as required by Corollary 8. Write $t=\sigma(wz)$, with $w\in M$ and $z\in[\imath(w)]$.

If $x\notin\imath(w)=\imath(\pi(t))$, it follows immediately that $t\mid S(tx)$. There remains the case where $x\in\imath(w)$ and $S(tx)\notin T$ (since the case $S(tx)\in T$ is trivial).

Now, $\sigma(wz)\in T$, but $\sigma(wzx)\notin T$, hence $\partial_x wz=r_x(M)-1$. We can write uniquely $wz=ux^{r_x(M)-1}v$, with $\max(\underline{u})\prec x\prec\min(\underline{v})$. By hypothesis, there exists $s\in M$, with x extremal in s (without loss, $x=\max(\underline{s})$) such that $s\backslash x$ divides w. Since $\partial_x s\leqslant r_x(M)$, $s|wzx$, and by maximality of x in $\underline{s}$, $s\mid ux^{r_x(M)}$. Let p result from raising the degree of each internal letter of s to its exponent in u. Then, $p\in s[\imath(s)]$ and its internal letters have small degree, so $\sigma(p)\in T$, and it is the factor of $S(tx)$ we sought after.

Clearly (iii) implies (i), and the theorem is proved. □

Example 3. Consider the set $M=\{ab^2c,a^3b\}$. The ordering $a\prec b\prec c$ does not satisfy the conditions above, since b is internal to ab^2c, and extremal only in a^3b, but $a^3b\backslash b=a^3$ does not divide ab^2c; indeed, $\sigma\langle M\rangle$ comprises all words $a^ib^{j+2}c^k$ and $a^{i+3}b^j$ with $i,j,k\geqslant 0$, and the set $\{ab^{j+2}c \mid j\geqslant 0\}$ cannot be generated as multiples of finitely many of those. Similarly, $a\prec c\prec b$ fails, since c is internal to a^3b, and extremal in none. However, $b\prec a\prec c$ is good: a is internal to b^2ac only, and $ba^3\backslash a=b$ divides b^2ac. In this case, $\mathscr{I}(M)$ is generated by $\{b^2ac,b^2a^2c,ba^3\}$.

Condition (iii) above is a fairly precise description of the minimal generating set of $\mathscr{I}(M)$. One gets a quick and dirty estimate for its size by forgetting most parameters:

Corollary 10. *Let M be a finite set of monomials in $[X]$. Then, if $\mathscr{I}(M)$ is finitely generated, it can be generated by a set of at most*

$$|M'| \prod_{x \in \imath(M)} r_x(M) + |M| - |M'|$$

elements, where $M' = \{w \in M | \imath(w) \neq \emptyset\}$ and $\imath(M) = \bigcup_{w \in M} \imath(w)$.

Even though it is a rough estimate, the result above is best possible. To see this, suppose $X = \{x_1, x_2, \dots, x_n\}$, ordered according to the indices. Choose positive integers $m, r_2, r_3, \dots, r_{n-1}$, and let $M = \{x_1^{i+1} x_n^{m-i} \mid 0 \leqslant i < m\} \cup \{x_i^{r_i} | i = 2, \dots, n-1\}$. Then, the minimal generating set for $\mathscr{I}(M)$ is precisely that described in Theorem 9(iii) and has size $m \prod r_i + n - 2$.

If we are given M as a collection of integer vectors, divisibility is just componentwise comparison, so it can be tested rapidly. A naive check of the condition on Theorem 1 would need at most $|M|^2|X|$ such comparisons, so Problem 2 can be solved by a polynomial-time algorithm.

We end this section with some constructions that will be useful later and some unexpected consequences of the theorem.

Given a collection M of monomials and an integer k, M_k will denote the subset of M consisting of monomials whose support has size at most k.

Proposition 11. *If an antichain M of monomials is such that $\mathscr{I}(M)$ is finitely generated, so is $\mathscr{I}(M_k)$ for each integer k.*

Proof. Let us show that M_k satisfies condition (ii) of Theorem 9. If $w \in M_k$, and x is internal to w, we know, since $\mathscr{I}(M)$ is finitely generated, that for some monomial $u \in M$, x is extremal in u and $u \backslash x$ is a factor of w. Since x is internal to w and extremal in u, the support of $u \backslash x$ is a proper subset of the support of w, so $|\underline{u}| \leqslant k$; that is, $u \in M_k$. □

Proposition 12. *Let $w \leftrightarrow \hat{w}$ be a bijection between sets M and $\hat{M}$ of monomials, such that:*

(i) *For every w in M, w and $\hat{w}$ have the same extremal letters.*
(ii) *For every u, v in M and x in X, if $\partial_x u \leqslant \partial_x v$, then $\partial_x \hat{u} \leqslant \partial_x \hat{v}$.*
If $\hat{M}$ is an antichain, and $\mathscr{I}(M)$ is finitely generated, then so is $\mathscr{I}\hat{M}$.

Proof. First notice that, from condition (ii), $u \mid v$ implies $\hat{u} \mid \hat{v}$, so M is an antichain. Thus, it must satisfy Theorem 9 (ii). Let $\hat{w} \in \hat{M}$, and $x \in X$ be internal to $\hat{w}$. From condition (i), x is internal to w, so there exists $u \in M$ such that x is extremal in u and $u \backslash x | w$. Again from (i), x is extremal in $\hat{u}$, and from condition (ii), $\hat{u} \backslash x | \hat{w}$. □

5. Cool orderings

Given an antichain M of monomials in $[X]$, we say that an ordering of X is *cool* for M if, for every w in M and letter x internal to w, there exists s in M such that x is extremal in s and $s\backslash x$ divides w.

As we will see in the next section, no good algorithm is forthcoming to decide whether a cool ordering exists. Before giving substance to this, we try to get a better understanding of such orderings. We begin with an immediate consequence of Propositions 11 and 12.

Proposition 13. *Let M be an antichain of monomials. Then, any cool ordering for M is also a cool ordering for*:

1. *M_k, for any integer k.*
2. *Any $\hat{M}$, obtained from M by changing each w to $\hat{w}$, so that*:

(a) *w and $\hat{w}$ have the same support, and*
(b) *For every u, v in M and x in X, if $\partial_x u \leqslant \partial_x v$, then $\partial_x \hat{u} \leqslant \partial_x \hat{v}$.*

This gives some necessary conditions for existence of cool orderings. We also get a kind of equivalence between sets of monomials:

Proposition 14. *Let $w \leftrightarrow \hat{w}$ be a bijection between set M and $\hat{M}$ of monomials, such that*:

(i) *For every w in M, $\underline{w} = \underline{\hat{w}}$, and*
(ii) *For every u, v in M and x in X, $\partial_x u < \partial_x v$ if and only if $\partial_x \hat{u} < \partial_x \hat{v}$.*

Then, an ordering of X is cool for M if and only if it is so for $\hat{M}$.

Proof. It is easily checked that $u \mid v$ if and only if $\hat{u} \mid \hat{v}$. So, the bijection maps minimal monomials to minimal monomials, antichains to antichains, and so on. It is just a matter of applying part 2 of Proposition 13 in both directions. □

From now to the end of the article, an ordering of the letters will not be given at the outset, and the following concept will be useful for the search of cool orderings. A monomial w is said to *help* a monomial m *with* the letter x if $x \in \underline{w}$, $w\backslash x \mid m$ and $\underline{w\backslash x} \subsetneq \underline{m\backslash x}$.

If $\prec$ is a cool ordering for a set M and, for some $Y \subset X$, every monomial in M has support included in Y or disjoint from Y, then $\prec$ is a cool ordering for those monomials with support included in Y. This can be extended to the following easily verified fact:

Proposition 15. *A cool ordering for M is also cool for any $N \subseteq M$ such that all members on M that help some member of N (with some letter) are in N.*

Next section will consider the problem of finding a cool ordering, given M. We close the section considering a question that is a sort of opposite of that: how must M look like if *every* ordering of X is cool? The answer is surprisingly simple.

Theorem 16. *Let M be an antichain of monomials over X. Then, every ordering of X is cool for M if and only if, for every m in M and x in X such that $|\underline{m\backslash x}| \geqslant 2$, there exists a u in M_2 such that $u\backslash x \mid m$.*

Proof. Suppose that every ordering of X is cool for M. Let m and x be given as in the statement. Then, there exists an ordering $\prec$ on X such that x is internal to m. Since $\prec$ is cool, there is a u in M that helps m with x. Choose u with minimal support, and let us show that $\underline{u}$ has size at most 2. If $|\underline{u}| > 2$, there is another ordering of X that makes x internal to u. Again by Theorem 9, there exists a v in M that helps u with x; clearly, v also helps m with x. Since $x \in \underline{u} \cap \underline{v}$, it follows that $\underline{v} \subsetneq \underline{u}$, so we have contradicted the minimality of $\underline{u}$.

Conversely, suppose the divisibility condition holds, and consider an arbitrary ordering of X. Pick an $m \in M$ and let $x \in \imath(m)$. Choose u in M_2 such that $u\backslash x \mid m$; clearly x is extremal in u. It follows that the ordering is cool. □

6. Finding cool orders is hard

Monoids generated by square-free monomials appear frequently in algebraic combinatorics (related to Stanley–Reisner rings of simplicial complexes), and have been studied in the current context by Peeva and Sturmfels, together with Eisenbud [5] and Reiner [15]. Propositions 18 and 20 tell the same as [5, Proposition 3.2] and [15, Lemma 3.1], although the different jargon may obscure this. After that we move to another direction.

Proposition 17. *Suppose that M is an antichain and the degree of the letter x in $w \in M$ is the largest degree it has in all monomials in M. Then, in any cool ordering for M, x cannot be internal to w.*

Proof. If there is a cool ordering for M where x is internal to w, there exists a $t \in M$ such that $t\backslash x | w$. But since $\partial_x w \geqslant \partial_x t$, it follows that $t \mid w$, a contradiction, as M is an antichain. □

The following is an immediate corollary:

Proposition 18. *If M consists only of square-free monomials and affords a cool ordering, then its monomials have total degree at most* 2.

Degree 1 monomials are trivially handled here, so the square-free sets M of interest consist only of quadratic monomials. Polynomial ideals whose initial ideals are generated by quadratic monomials (mostly square-free) were extensively studied in [15].

We leave now the square-free condition, and consider the case when M consists exclusively of quadratic monomials, that is, we allow monomials of form x^2. This seemingly trivial extension has deep consequences:

Proposition 19. *The problem of deciding, given a set of quadratic monomials M, whether there exists a cool ordering for M is:*

(a) *Solvable in polynomial time, if M is square-free.*
(b) *NP-complete, in general.*

Proof. Part (a) follows from Proposition 20 and part (b) from Proposition 21. □

With quadratic monomials, irrespective of the order of the letters, each letter is extremal in each monomial it occurs, so, an ordering $\prec$ on X is cool for M if and only if whenever $x \prec y \prec z$ and xz is in M, then at least one of y^2, xy, yz is in M.

At this point, it becomes convenient to encode the data and the problem by means of graphs, and it turns out to be convenient to use the complement of what comes naturally. The graph $G(M)$ will have the letters as vertices, xy is an edge if xy is not in M. Let T_M denote the set of letters whose square is *not* in M.

An orientation of a graph is said to be *transitive* at a vertex y if, whenever oriented edges $x \to y$ and $y \to z$ exist, then the edge $x \to z$ must also exist. A graph is a *comparability graph* if it admits an orientation that is transitive at all its vertices; such an orientation is always acyclic. Comparability graphs have been widely studied, and can be recognized efficiently [6] (or [14]), [8,16].

Proposition 20. *A set M of quadratic monomials admits a cool order if and only if $G(M)$ admits an acyclic orientation that is transitive at all vertices of T_M. In particular, if M is square-free, it admits a cool order if and only if $G(M)$ is a comparability graph.*

Proof. Suppose M has a cool ordering. Direct all edges of $G(M)$ from the smallest to the largest vertex. This orientation is trivially acyclic. If $y \in T_M$, and edges $x \to y$ and $y \to z$ exist, then the monomials y^2, xy and yz are not in M. By coolness, $xz \notin M$, so the edge xz is in $G(M)$, and is correctly oriented.

Conversely, suppose that $G(M)$ admits an acyclic orientation that is transitive at all vertices of T_M. With a "topological sort" order its vertices so that all directed edges point from the smaller to the bigger end. One readily verifies that this ordering is cool for M.

When M is square-free, T_M comprises all vertices, so an acyclic orientation that is transitive at all vertices of T_M says that $G(M)$ is a comparability graph. □

We refer the reader to the already classic text [7] as a general reference for NP-completeness, good algorithms and satisfiability. Since good algorithms for recognition of comparability graphs are known, one would expect that testing the condition of Proposition 20 would also be feasible.

Proposition 21. *The problem*:

> given a graph G and a set $T \subseteq VG$, is there an acyclic orientation of G that is transitive at all vertices in T?

is NP-complete.

Proof. Let us shorten "orientation transitive at T" to *T-orientation*. The proof will be by a reduction from NOT-ALL-EQUAL-3SAT [17]. The basic gadget is the graph in Fig. 1, where the vertices in T are black (and labeled a, $\bar{a}$, c).

Fact 1. *The orientation $a \to \bar{a}$ of the edge $a\bar{a}$ can be extended to a unique T-orientation of this graph. In this orientation, a is a source, $\bar{a}$ is a sink, and the bottom edge is directed from* r to l.

To see this, notice that since the edge $s\bar{a}$ does not exist, sa must be oriented as $a \to s$, because of transitivity at a. By a similar argument we check that all edges with an end in T can have only one orientation. Finally, since the orientations $r \to c$ and $c \to l$ are forced, $r \to l$ is forced by transitivity at c.

Now we construct the main gadget by gluing three copies of the top hat, identifying cyclically each t with the next s and each r with the next l. The result is in Fig. 2, where only important vertices are labeled.

Fact 2. *Consider an orientation of the edges $a_1\bar{a}_1$, $a_2\bar{a}_2$ and $a_3\bar{a}_3$. It extends to an acyclic T-orientation of the gadget if and only if they are not all directed the same way along the external cycle.*

Indeed, by looking at the top hats we see that any orientation of these edges extends uniquely to a T-orientation of the gadget. If they are all oriented the same way, the inner triangle becomes a directed cycle. Conversely, if they are not all the same way (by symmetry, there is only one case to check), the orientation of the gadget is acyclic.

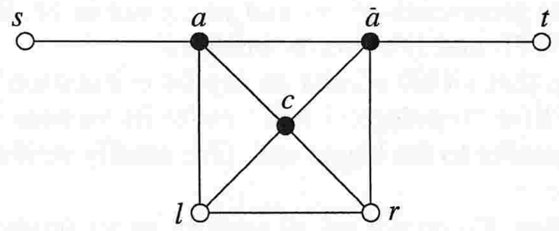


Fig. 1. The top hat.

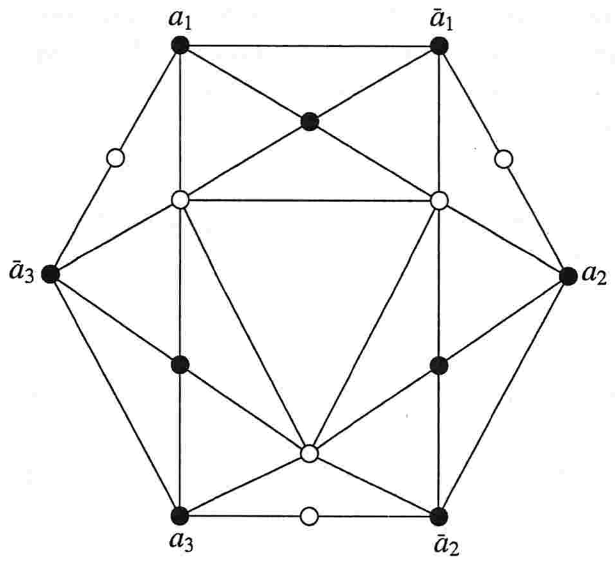


Fig. 2. The gadget.

Now we proceed to the reduction. A typical instance of NOT-ALL-EQUAL-3SAT consists of a set X of variables and a set C of clauses over X, where each clause has three literals, each of form x or $\bar{x}$, for some $x \in X$. The question is whether there exists a truth assignment to X, so that for each clause one literal gets value *true* and one gets value *false*.

For each clause, take a copy of the gadget and replace the labels a_1, a_2 and a_3 by the literals, and $\bar{a}_1$, $\bar{a}_2$ and $\bar{a}_3$ by the complements of the literals in the clause. Add to that a vertex v_x for each variable x, and join it to all vertices labeled x. Call the resulting graph G, and let T be formed by all v_x together with the union of all black vertices from the gadgets.

We will show a 1-1 correspondence between truth assignments for V that solve C and acyclic T-orientations of G. Start with a truth assignment. For each edge labeled $x\bar{x}$, orient it from x if x is assigned *true* and towards x otherwise. Consider a clause and its respective gadget. The three literals in the clause are not all *true* and not all *false*, so the three special edges are not all directed the same way. It follows from Fact 2 that one can orient (uniquely) all gadgets extending these orientations. In this orientation, all vertices labeled with the same literal are sources in their gadgets if that literal is *true*, and sinks otherwise. This T-orientation can now be extended to the whole G, directing all edges incident to v_x towards it if x is *true* and the opposite otherwise.

Conversely, suppose a T-orientation of G is given. Since all neighbors of v_x are pairwise nonadjacent, v_x is either a source or a sink. Assign x *true* if v_x is a sink, *false* otherwise. The fact that each gadget is acyclically T-oriented shows that in the corresponding clause the not-all-equal condition is satisfied. □

Since simple powers have originated the problem in Proposition 19, we tried to look at another extension of its first part, namely, allow only monomials with support size exactly 2. This was short lived, though:

Proposition 22. *The problem*:

> given a collection M of monomials, each with support of size 2, does there exist a cool ordering for M?

is NP-complete.

Proof. We reduce the quadratic case to this. Suppose that M is a collection of quadratic monomials, and let $M' = \{xy \mid xy \in M\} \cup \{x^2y \mid x^2 \in M \text{ and } xy \notin M\}$. It is easy to check that M and M' have precisely the same cool orderings. □

There is a lot of leeway in the reduction in the proof of Proposition 21. For instance, the v_i could be eliminated, and similarly labeled vertices could be merged. One could add irrelevant vertices of both types and show that existence of acyclic T-orientations is NP-complete even if $|T| = \frac{1}{2}|VG|$ (any constant between 0 and 1 would do). On the extremes, the problem can be solved:

When $T = VG$, that is recognition of comparability graphs. When T induces a bipartite graph, any acyclic orientation in which one side of T consists only of sources and the other (if it exists) only of sinks is a T-orientation. This takes care of $|T| \leqslant 2$; actually, for any fixed k, if $|T| \leqslant k$, one can restrict the search for a T-orientation to a polynomial number of acyclic orientations that can be systematically enumerated.

7. Lifting the ideal

Here we present the solution to Problem 1.

Theorem 23. *Given an antichain of monomials $M \subseteq [X]$, the following are equivalent*:

(i) *$\pi^{-1}\langle M \rangle$ is a finitely generated ideal of X^*.*
(ii) *For every m in M and any letters $x \neq z \neq y$ such that $xy|m$, there exists a monomial w in M such that $w \backslash z$ divides either mx^{-1} or my^{-1}. (Note that $x = y$ is included.)*
(iii) *For every m in M and any letter z such that no power of it is in M, if $m \backslash z$ has degree $\geqslant 2$, there exists a monomial $z^r t$ in M_2, such that $t \in \underset{\sim}{m}$.*
(iv) *$\pi^{-1}\langle M \rangle$ is generated by the inverse images of the monomials $m \in \langle M \rangle$ such that, for every letter x, $\partial_x m \leqslant \max_{u \in M_2} \partial_x u$.*

Proof. (i) implies (ii): Given m, x, y, and z, choose $u = xz^r vy \in \pi^{-1}(m)$, where $r \geqslant 0$. Now, for every $s \geqslant r$, $xz^s vy \in \pi^{-1}\langle M \rangle$, and since $\pi^{-1}\langle M \rangle$ is finitely generated, some minimal generator g divides infinitely many of these. This is only possible if g divides some xz^s or some $z^s vy$. So, $\pi(g)\backslash z$ divides either my^{-1} or mx^{-1}. The result follows by taking any w in M that divides $\pi(g)$.

(ii) implies (iii): Given m and z, choose letters x and y such that $xy \mid m\backslash z$. Let w be given by condition (ii), with minimal support. Since M is an antichain, $z \in \underline{w}$. Suppose, by contradiction, that $w\backslash z$ has degree $\geqslant 2$. We apply condition (ii) to w, obtaining a w'; that new monomial could also play the role of w with respect to m, so, by minimality of w, it cannot exist. Since no power of z is in M, $w = z^r t$ for some letter $t \neq z$. As $w\backslash z$ divides either mx^{-1} or my^{-1}, it follows that $t \in \underline{m}$.

(iii) implies (iv): Let W be the set claimed to generate $\pi^{-1}\langle M\rangle$. If this is false, then $\langle W\rangle \subsetneq \pi^{-1}\langle M\rangle$, since $W \subseteq \pi^{-1}\langle M\rangle$. So, there must exist a $w \in \pi^{-1}\langle M\rangle$, of minimum length, with no factor in W. So, for some letter z, $\partial_z \pi(w) > r = \max_{u\in M_2} \partial_z u$.

Suppose that $z^r \in M$. Clearly w has a proper factor u such that $z^r \mid \pi(u)$, so $u \in \pi^{-1}\langle M\rangle$. By minimality of w, u has a factor in W; then, so does w, a contradiction. So, $z^r \notin M$, and by the choice of r and as M is an antichain, no power of z lies in M. Now, let $m \in M$ be such that $m \mid \pi(w)$. By (iii), there exists a $z^s t \in M$ such that $t \in \underline{m}$. Since $s \leqslant r$, w has a proper factor u such that $z^s t \mid \pi(u)$. We get a contradiction again, that finishes the proof.

(iv) implies (i): We deserve the rest. $\square$

We briefly relate this result to the preceding ones. It is easy to check from the definitions that if $\pi^{-1}\langle M\rangle$ is finitely generated, then every ordering of X is cool for M. This can also be seen from the fact that if M satisfies the condition in Theorem 23(iii), then it also satisfies the condition of Theorem 16. The converse is not true; the simplest example is $M = \{a^2, bc\}$—here, every ordering is cool, but $\pi^{-1}\langle M\rangle$ is not finitely generated. Actually, if one starts with any M for which every ordering is cool and substitutes each letter for its square, this property is preserved. But now, $\pi^{-1}\langle M\rangle$ is not finitely generated.

The similarity between Theorems 23 and 9 may suggest that perhaps a restricted form of 9(ii) involving M_2 would hold. That is not likely, as suggested by $M = \{x_1x_2^2, x_1x_2x_3^2, x_1x_2x_3x_4^2, \dots\}$; the natural ordering of $x_1, x_2, \dots, x_n$ is cool for M, for any n, even though M_2 is quite skimpy.

8. Commutative ideals given by inequalities

Another way of giving an ideal of $[X]$ is as the pre-image of an ideal under a morphism from $[X]$ to another commutative monoid. This is useful only if there is a nice way of describing the morphisms and the ideals of the target. We will consider morphisms between free commutative monoids and lift ideals given by generators.

In this setting, it will be convenient to switch to an additive notation for $[X]$. We number the letters of X as $x_1, x_2, \dots, x_n$, and identify $[X]$ with $\mathbb{N}^n$ by the isomorphism given by $x_1^{i_1}x_2^{i_2}\cdots x_n^{i_n} \mapsto \mathbf{x} = (i_1, i_2, \dots, i_n)$. In this notation, a set $I \subseteq \mathbb{N}^n$ is an ideal if

$\mathbf{x} \geqslant \mathbf{y} \in I$ implies $\mathbf{x} \in I$ (as usual, $\mathbf{x} \geqslant \mathbf{y}$ means $\mathbf{x}_i \geqslant \mathbf{y}_i$ for every i). Other terms require translation: monomials become vectors, letters become indices or coordinates of vectors, and so on.

Consider a morphism $\varphi : [X] \to [Y]$, where an isomorphism $[Y] \to \mathbb{N}^m$ is already fixed. If $\varphi(x_j) = \prod_i y_i^{a_{ij}}$, then φ is the linear map $\mathbb{N}^n \to \mathbb{N}^m$ given by the matrix $A = (a_{ij})$, where each a_{ij} is a nonnegative integer. If J is an ideal of $\mathbb{N}^m$, then $I = \{\mathbf{x} \in \mathbb{N}^n | A\mathbf{x} \in J\}$ is an ideal of $[X]$; moreover, if J is generated by the finite set W, then that same ideal I can be described as:

$$I(A, W) = \{\mathbf{x} \in \mathbb{N}^n \mid A\mathbf{x} \geqslant w \text{ for some } w \in W\}.$$

When w is a vector in $\mathbb{N}^m$, we write $I(A, w)$ for $I(A, \{w\})$. Also, we stress that we only consider $I(A, w)$ when A is nonnegative.

Given a generating set M of an ideal of $[X]$, one can write down a description $\langle M \rangle = I(I_n, M)$, simply using $Y = X$ and the identity morphism as φ. From the complexity viewpoint, we notice that the new description has size bounded by a polynomial on the size of M; one interesting feature of the new type of description is that it can be much more compact. Just to give a trivial example, consider, for each k in $\mathbb{N}$, the ideal $\{\mathbf{x} \in \mathbb{N}^2 | x_1 + x_2 \geqslant k\}$. The size of this description is $O(\log k)$, while clearly it has $k + 1$ minimal generators.

Clearly, $I(A, w) = \bigcup_{w \in W} I(A, w)$, and this suggests the following definition: we say that an ideal of $[X]$ is *convex* if it is of form $I(A, w)$ for some integer matrix A and vector w. The name is motivated by the following fact, that follows from standard results in the theory of polyhedra (see [18] for terminology and facts about polyhedra that we use).

Proposition 24. *Let $I = \langle M \rangle$ be an ideal of $[X] = \mathbb{N}^n$, with M finite. The following are equivalent*:

(i) *I is convex.*
(ii) *I is the intersection of $\mathbb{N}^n$ with a convex set in $\mathbb{R}^n$.*
(iii) *$I = \mathbb{N}^n \cap (\mathrm{conv}(M) + \mathbb{R}^n_+)$ ($\mathrm{conv}(M)$ is the convex hull of M).*

So, $I(A, w)$ is a union of convex ideals. It turns out that any union of convex ideals can be expressed as an $I(A, w)$. As we see below, this can be done without wasting much space, so we can switch descriptions without penalty in the coarse complexity of the problems we will talk about.

Lemma 25. *Let $I_1 = I(A^{(1)}, w^{(1)})$, $I_2 = I(A^{(2)}, w^{(2)})$, $\ldots$, $I_r = I(A^{(r)}, w^{(r)})$ be ideals of $[X]$. Then there exist a matrix A and a set W of vectors, with total size polynomial in the total size of the descriptions $I(A^{(i)}, w^{(i)})$, such that $\bigcup_i I_i = I(A, w)$.*

Proof. Let A result from piling up the matrices $A^{(i)}$ on top of each other. For each i, let w^i result from extending $w^{(i)}$ with null entries corresponding to the inequalities of the other systems; so, $I(A, w^i) = I_i$. Finally, let $W = \{w^1, w^2, \ldots, w^r\}$.

For those of a more categorical persuasion, the proof is simply the substitution of a family of morphisms by its direct product. □

Now we consider what happens to the three guiding problems of the introduction when I is given in the form $I(A, w)$. Problem 3 is sort of hopeless, since a description $\langle M \rangle$ can be converted into a description $I(A, w)$ of size polynomial in the size of M, and Problem 3 is NP-complete when M is the given data. It follows that this problem with I given as $I(A, w)$ is NP-hard; to make things worse, we cannot even assert that it is in NP. At this point, we refer the reader again to [7] for a refresher on NP-completeness concepts, and, in particular, to the satisfiability problem, that will play an important role in the remainder of this section.

For the other two problems, our results are similarly bad and more definite. They will be shown to be coNP-complete. Indeed, we will add a new problem to the pack, that is completely trivial if the ideal is given by generators:

Problem 4. Given an ideal I of $[X]$, is it generated by monomials with support of size at most 2? That is, is there a set M such that $M = M_2$ and $\langle M \rangle = I$? □

We register two basic algorithms pertaining to these problems.

Lemma 26. *Given an ideal $I = I(A, w)$ and a vector $\mathbf{x}$, it can be decided in polynomial time whether $\mathbf{x} \in I$ and whether $\mathbf{x}$ is a minimal generator of I.*

Proof. Computing $A\mathbf{x}$ and comparing the result with each member of W, we quickly decide membership in I. To decide whether an $\mathbf{x} \in I$ is minimal, it is enough to verify that each vector obtained from $\mathbf{x}$ by subtracting 1 from a positive coordinate is not in I. □

In what follows, the proofs will be a bit sketchy, with some bare statements; filling in the details is routine handiwork.

Recall that a decision problem is in coNP if, the problem obtained by reversing the answer is in NP; in other words, **no**-instances have short certificates.

Proposition 27. *Problems* 1, 2 *and* 4 *are in coNP, when the ideal is given as $I(A, w)$.*

Proof. For each problem, when the answer to an instance is **no**, we will present a short certificate, verifiable in polynomial time. That will be a minimal generator of the ideal, and some additional information. Notice that any minimal generator has coordinates bounded by the maximum of all coordinates in members of W, so it can be part of a short certificate.

For Problem 4, a certificate is simply a minimal generator with support of size at least 3.

For Problem 1, a certificate is a minimal generator $\mathbf{m}$ and an index z such that item (iii) of Theorem 23 is violated. That amounts to the following:

- There is no vector in I whose support is $\{z\}$ (no power of z is in M). This happens if and only if, for each $w \in W$, there is an index i such that $w_i > 0$ and $a_{iz} = 0$.

- $\sum_{i \neq z} m_i \geqslant 2$ ($m \backslash z$ has degree $\geqslant 2$).
- There is no $\mathbf{x} \in I$ and index $t \neq z$ such that $m_t > 0$, $x_t = 1$ and $x_i = 0$ for every $i \neq t, z$. This is true, for each candidate t, if and only if for every $w \in W$ there exists an i such that $w_i > a_{it}$ and $a_{iz} = 0$.

For Problem 2, we assume, without loss of generality, that the ordering on the letters is that of the indexing. Now, a certificate consists of a minimal generator $\mathbf{m}$ and an index x, interior to $\mathbf{m}$ satisfying the condition: there is no minimal generator $\mathbf{s}$ whose first or last positive entry is in position x, and such that if $\mathbf{s}'$ results from $\mathbf{s}'$ by turning the x-component to 0, then $\mathbf{s}' \leqslant \mathbf{m}$. This condition can be checked as follows. Let $\mathbf{m}_{\leftarrow}$ ($\mathbf{m}_{\rightarrow}$) result from $\mathbf{m}$ by changing to zero all components with index bigger (smaller) than x. Let also A', W' result from eliminating all rows i such that $a_{ix} > 0$. Then, x satisfies the required condition if and only if neither $\mathbf{m}_{\leftarrow}$ nor $\mathbf{m}_{\rightarrow}$ is in $I(A', W')$.

Proposition 28. *Problems* 1, 2 *and* 4 *are coNP-complete, when the ideal is given as* $I(A, w)$.

Proof. We will reduce directly from SAT to the negative of each problem. The reductions will have a lot in common. From each instance of SAT, we will produce a family of convex ideals I_i, like in Lemma 25; instead of presenting them in matrix form, we write them as systems of linear inequalities.

Given an instance S of SAT on variables $x_1, x_2, \dots, x_n$, (we assume $n \geqslant 3$) our inequalities will involve the variables $x_1, \bar{x}_1, x_2, \bar{x}_2 \dots, x_n, \bar{x}_n$, in obvious correspondence to the literals. For each clause, the corresponding *clause inequality* is

sum of the literals in the clause $\geqslant 1$.

Let I_0 be defined by the clause inequalities, together with the *boolean inequalities* $x_i + \bar{x}_i \geqslant 1$ for $i = 1, 2, \dots, n$. The specific use of I_0 is the following: $\mathbf{x}$ *is a solution of* I_0 *in nonnegative integers such that each boolean inequality is satisfied as equality if and only if* $\mathbf{x}$ *is a boolean assignment satisfying* S.

We also define, for each $i = 1, 2, \dots, n$, the ideal I_i given by the single inequality $x_i + \bar{x}_i \geqslant 2$. Notice that I_i has three minimal generators: two with single support ($x_i = 2$ or $\bar{x}_i = 2$), the other with two-element support ($x_i = \bar{x}_i = 1$).

Reduction to Problem 4: The instance P of Problem 4 consists precisely of the systems $I_0, I_1, \dots, I_n$.

Suppose that S is satisfiable, and let $\mathbf{x}$ be a boolean assignment satisfying S. Since for each i, exactly one of x_i or $\bar{x}_i$ equals 1, the support of $\mathbf{x}$ has size $n \geqslant 3$, and $\mathbf{x}$ is not in any I_i, $i \geqslant 1$. On the other hand, clearly $\mathbf{x}$ is in I_0. Also, $\mathbf{x}$ is minimal, since zeroing any variable would violate the corresponding boolean inequality. So, P has a negative answer if S is satisfiable.

Conversely, suppose P has a negative answer, that is, the corresponding ideal has a minimal generator $\mathbf{x}$ whose support has size $\geqslant 3$. Clearly it cannot be in any I_i with $i \geqslant 1$, so it is in I_0, and $x_i + \bar{x}_i = 1$ for each i. Hence, S is satisfiable.

Reduction to Problem 2: We introduce two new variables, y and z, besides the ones we already have. The instance P of Problem 2 consists of the systems $I_1, \ldots, I_n$, together with I'_0, which is I_0 with the addition of the inequality $y \geqslant 1$. The variables of P will be ordered increasingly as $y, z, x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n$.

Suppose that S is satisfiable, define $\mathbf{x}$ as before, and extend it by setting $y = 1$ and $z = 0$. Then this is a minimal generator and is only in I'_0. Now, z is internal to this vector, but no minimal generator of the ideal has z in its support (let alone, as an extreme entry), so condition (ii) of Theorem 9 is violated, and P has a negative answer.

Conversely, if P has a negative answer, there exists a minimal generator and an internal variable such that condition (ii) of Theorem 9 is violated. This minimal generator cannot be in any of the $I_i, i \geqslant 1$, since those have no internal letters. So it is in I'_0, and must have $y = 1$, $z = 0$, and the other variables must be a boolean assignment that satisfies S. The problematic internal variable must be z, but who cares?

Reduction to Problem 1: We use just one new variable y. The instance P of Problem 1 consists of I_0, a new system I_*, with the single inequality $y \geqslant 2$, systems I'_i, each obtained from I_i by the addition of the inequality $y \geqslant 1$. By arguments similar to the preceding ones and the help of Theorem 23(iii), it can be shown that S is satisfiable if and only if P has a negative answer.

Proposition 24(iii) says that a convex ideal is the set of integer points of a *blocking polyhedron*. Such polyhedra, and mostly their integer points, have been the subject of a lot of attention in the context of combinatorial and integer programming. This, and perhaps sheer curiosity, justify asking what happens to Problems 1–4 if one restricts the questions to convex ideals (given in the form $I(A, w)$). No one of the definite results we presented so far applies to convex ideals; in particular, the proof of Theorem 19 constructs ideals that are not convex, so even Problem 3's status is undecided.

References

[1] D.J. Anick, On the homology of associative algebras, Trans. Amer. Math. Soc. 296 (1986) 641–659.

[2] D. Anick, G.-C. Rota, Higher-order syzygies for the bracket algebra and for the ring of coordinates of the Grassmannian, Proc. Nat. Acad. Sci. USA 88 (1991) 8087–8090.

[3] M.-P. Béal, F. Mignosi, A. Restivo, M. Sciortino, Forbidden words in symbolic dynamics, Adv. Appl. Math. 25 (2000) 163–193.

[4] M. Crochemore, F. Mignosi, A. Restivo, Automata and forbidden words, Inform. Process. Lett. 67 (1998) 111–117.

[5] D. Eisenbud, I. Peeva, B. Sturmfels, Non-commutative Gröbner bases for commutative algebras, Proc. Amer. Math. Soc. 126 (1998) 687–691.

[6] T. Gallai, Graphen mit triangulierbaren ungeraden Vielecken, Magyar Tud. Akad. Mat. Kutat Int. Közl. 7 (1962) 3–36.

[7] M. Garey, D.S. Johnson, Computers and Intractability. A guide to the theory of NP-Completeness, W.H. Freeman and Co., San Francisco, CA, 1979.

[8] M.C. Golumbic, Algorithmic Aspects of Perfect Graphs, North-Holland Mathematical Studies, Vol. 88, North-Holland, Amsterdam, 1984.
[9] E. Green, Noncommutative Gröbner bases and projective resolutions, Progress in Mathematics, Vol. 173, Birkhäuser Verlag, Basel/Switzerland, 1999, pp. 29–60 (Chapter 2).
[10] E. Green, R.Q. Huang, Projective resolutions of straightening closed algebras generated by minors, Adv. in Math. 110 (1995) 314–333.
[11] G. Higman, Ordering by divisibility in abstract algebras, Proc. London Math. Soc. 3 (1952) 326–336.
[12] H.R. Lewis, C.H. Papadimitiou, Elements in the Theory of Computation, 2nd Edition, Prentice-Hall, Englewood Cliffs, NJ, 1998.
[13] T. Mora, An introduction to commutative and noncommutative Gröbner bases, Theoret. Comput. Sci. 134 (1994) 131–173.
[14] F. Maffray, M. Preissmann, A translation of Gallai's paper: Transitiv Orientierbare Graphen, in: J.L. Ramírez-Alfonsín, B.A. Reed (Eds.), Perfect Graphs, Jossey-Bass, New York, 2001.
[15] I. Peeva, V. Reiner, B. Sturmfels, How to shell a monoid, Math. Ann. 310 (1998) 379–393.
[16] J.L. Ramírez-Alfonsín, B.A. Reed (Eds.), Perfect Graphs, Jossey-Bass, New York, 2001.
[17] T.J. Schaefer, The complexity of satisfiability problems, Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (San Diego, CA, 1978), ACM, New York, 1978, pp. 216–226.
[18] A. Schrijver, Theory of Linear and Integer Programming, Wiley-Interscience Series in Discrete Mathematics, Wiley, Chichester, 1986.

The Minimum Cycle Cover and the Chinese Postman Problems on Mixed Graphs with Bounded Tree-Width

Cristina G. Fernandes [*]
Universidade de São Paulo, Brazil
`cris@ime.usp.br`

Orlando Lee [†]
Universidade Estadual de Campinas, Brazil
`lee@ic.unicamp.br`

Yoshiko Wakabayashi [‡]
Universidade de São Paulo, Brazil
`yw@ime.usp.br`

November 25, 2003

Abstract

Let $M = (V, E, A)$ be a mixed graph with vertex set V, edge set E and arc set A. A cycle cover of M is a family $\mathcal{C} = \{C_1, \ldots, C_k\}$ of cycles of M such that each edge/arc of M belongs to at least one cycle in $\mathcal{C}$. The weight of $\mathcal{C}$ is $\sum_{i=1}^{k} |C_i|$. The *minimum cycle cover problem* is the following: given a strongly connected mixed graph M without bridges, find a cycle cover of M with weight as small as possible. The *Chinese postman problem* is: given a strongly connected mixed graph M, find a minimum length closed walk using all edges and arcs of M. These problems are NP-hard. We show that they can be solved in polynomial time if M has bounded tree-width.

Keywords: tree-width, polynomial algorithms, cycle cover, Chinese postman problem, mixed graphs.

1 Introduction

Mixed graphs generalize the notion of digraphs and graphs in the sense that they may contain (undirected) edges as well as (directed) arcs. A **mixed graph** is a triple $M = (V, E, A)$ where V

[*]Research partially supported by MCT/CNPq through ProNEx Programme (Proc. CNPq 664107/97-4) and by CNPq Proc. 301174/97-0 (Brazil).

[†]Research supported in part by CAPES (Proc. 3302006-0) and CNPq Proc. 200611/00-3 (Brazil).

[‡]Research partially supported by MCT/CNPq through ProNEx Programme (Proc. CNPq 664107/97-4) and by CNPq (Proc. 30452/89-0 and Proc. 470608/01-3) (Brazil).

is a finite set of vertices, E is a finite set of edges and A is a finite set of arcs. When $E = \emptyset$ we say that M is a digraph, and when $A = \emptyset$ we say that M is a graph. Note that these consist of *simple* mixed graphs, that is, loops and parallel arcs/edges are not allowed.

A **cycle cover** of a mixed graph $M = (V, E, A)$ is a family $\mathcal{C} = \{C_1, \ldots, C_k\}$ of cycles of M such that each edge or arc of M belongs to at least one cycle in $\mathcal{C}$. We define the **weight** of $\mathcal{C}$ as the sum of the lengths of the cycles in $\mathcal{C}$, that is, $\sum_{i=1}^{k} |C_i|$.

The MINIMUM CYCLE COVER PROBLEM (MCCP) consists of the following: given a bridgeless strongly connected mixed graph M, find a cycle cover of M of minimum weight.

This problem is NP-hard if M is an arbitrary planar mixed graph [13]. It is well-studied when M is a graph: Thomassen [16] showed that this case is NP-hard. Also, this case is related to the well-known cycle double cover conjecture [12] and the Chinese postman problem [9].

Let M be a strongly connected mixed graph. A **postman walk** in M is a closed walk that contains all edges and arcs of M. Any cycle cover of weight k can be converted into a postman walk of length k, but the converse is not true. (See Figure 1.)

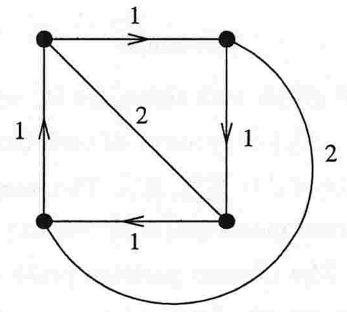


Figure 1: The numbers denote the number of times each edge/arc is used on a postman walk. This walk cannot be decomposed into a cycle cover of same weight. This postman walk has minimum length, while the weight of a minimum cycle cover is 9.

The CHINESE POSTMAN PROBLEM (CPP) in mixed graphs is a variant of MCCP: given a strongly connected mixed graph M, find a postman walk in M of minimum length.

We know that CPP can be solved in polynomial time in graphs [9] and in digraphs [10]. On the other hand, Papadimitriou [14] proved that CPP is NP-hard on planar mixed graphs.

In this paper, we study the complexity of MCCP and CPP on mixed graphs with a certain structural parameter—tree-width—bounded by a constant. The tree-width of a graph G can be defined using the notion of a tree-decomposition of G. The study of tree-decompositions started in the eighties and had several consequences in two areas: graph theory and design of algorithms. In the first, tree-decompositions were fundamental in the proof of Robertson and Seymour's result on graph minors: in every infinite sequence of graphs, there are two graphs such that one is a minor of the other. In the second, this concept were successfully explored in the design of polynomial-time algorithms for several NP-hard problems on graphs with bounded tree-width. We show a polynomial-time algorithm for MCCP and for CPP on mixed graphs with bounded tree-width. Many of the polynomial-time algorithms for graphs with bounded tree-width follow a straightforward dynamic programming approach. Although our algorithm is not an exception, it is not obvious.

Specifically, we prove the following theorems.

Theorem 1.1 CPP *is solvable in polynomial time on mixed graphs with bounded tree-width.*

Theorem 1.2 MCCP *is solvable in polynomial time on mixed graphs with bounded tree-width.*

The paper is organized as follows. Next we introduce some notation and give some definitions. In Section 2 we prove Theorem 1.1 and in Section 3 we prove Theorem 1.2.

1.1 Definitions and notation

Most of the concepts defined for graphs and digraphs (see [6]) can be extended in a natural way to mixed graphs. We assume the reader is familiar with them. We present here just a few concepts, to establish the notation.

A **walk** in a mixed graph is a sequence $W := (v_1, l_1, \ldots, v_{k-1}, l_{k-1}, v_k)$, where each v_i is a vertex and either $l_i = v_{i-1}v_i$ is an edge or $l_i = (v_{i-1}, v_i)$ is an arc. Sometimes W is denoted simply by $(v_1, \ldots, v_k)$. The **length** of W is $k-1$. We say that W is a walk **from** v_1 **to** v_k or that W **starts** at v_1 and **ends** at v_k. Also, we say that v_1 and v_k are the **ends** of W. If a walk starts and ends at the same vertex, we say it is **closed**. A **cycle** is a closed walk $(v_1, \ldots, v_k)$, where $v_i \neq v_j$, for all $1 \leq i < j < k$. A **path** is a walk $(v_1, \ldots, v_k)$, where $v_i \neq v_j$, for all $1 \leq i < j \leq k$.

A mixed graph is **strongly connected** if for any ordered pair of vertices (u, v), there is a path from u to v.

Given a mixed graph M, the **underlying undirected graph** of M is the graph obtained from M by replacing every arc in M by an edge with the same ends.

Let G be a graph and $\Theta := \big(T, (W_t)_{t \in V(T)}\big)$ be a pair consisting of a tree T and a multiset whose elements W_t, indexed by the vertices of T, are subsets of $V(G)$. For a vertex v of G, we denote by F_v the subgraph of T induced by those vertices t of T for which W_t contains v. Then Θ is called a **tree-decomposition** of G if the following two conditions hold:

(1) For every edge $e = xy$ of G, there is a vertex t of T such that $\{x, y\} \subseteq W_t$;

(2) For every vertex v of G, the subgraph F_v of T is a tree.

The **width** of Θ is the maximum, over all vertices t of T, of $|W_t| - 1$; and the **tree-width** of G is the least width of any tree-decomposition of G. We refer to [8, 15] for properties and other results on tree-decompositions.

We may assume that the tree T in any tree-decomposition is binary. If this is not the case, one can modify T, introducing new vertices in such a way that the modified tree-decomposition preserves the width of the original one.

For a mixed graph M, the concept of tree-decomposition refers to the tree-decomposition of its underlying undirected graph G. Let $\Theta := \left(T, (W_t)_{t \in V(T)}\right)$ be a tree-decomposition of G. For each vertex t of T, choose a set $E(W_t)$ of edges of M and a set $A(W_t)$ of arcs of M such that all edges and arcs in $E(W_t)$ and $A(W_t)$ have their two ends in W_t and every edge or arc in M appears in exactly one of these sets. These sets form a partition of $E(M) \cup A(M)$, which we call an **associated partition** of Θ. (Note that it is not unique.)

Given a mixed graph $M = (V, E, A)$ and a subset X of V, we denote by $\delta(X)$ the set of edges of M with exactly one end in X. We denote by $\delta^+(X)$ the set of arcs of M starting at a vertex in X and ending at a vertex not in X. Also, let $\delta^-(X) := \delta^+(V \setminus X)$. A **bridge** in M is an edge e of M such that $\delta(X) \cup \delta^+(X) \cup \delta^-(X) = \{e\}$, for some subset X of V. We say M is **Eulerian** if $|\delta(X)| - ||\delta^+(X)| - |\delta^-(X)||$ is an even non-negative number, for any subset X of V. The following theorem [4, 11] generalizes the results known for graphs and digraphs.

Theorem 1.3 *Any Eulerian mixed graph can be partitioned into a set of edge and arc disjoint cycles.* ■

2 The Chinese postman problem

In this section, we prove Theorem 1.1, that is, we present a polynomial-time algorithm for CPP on mixed graphs with bounded tree-width. To simplify the description of the algorithm, we represent a solution of CPP for a mixed graph $M = (V, E, A)$ in an unusual form. Let P be a postman walk in M. Consider the supergraph $H = (V_H, E_H, A_H)$ of M for which $V_H := V$ and, for each edge (arc) e of M, if e appears t times in P, then there are t copies of e in E_H (A_H). Clearly H is an Eulerian mixed graph. Therefore, by Theorem 1.3, H can be written as a union of edge and arc disjoint cycles, say, $C_1, \ldots, C_q$. Note that some of these cycles might not correspond to cycles in M. This happens exactly when a cycle C_i consists of two copies of the same edge of M. In this case, we say C_i is a **pseudocycle** in M. On the other hand, if $C_1, \ldots, C_q$ is a collection of cycles and pseudocycles of M whose union contains all edges and arcs of M, then the mixed graph given by $\bigcup_{i=1}^{q} C_i$ is Eulerian and corresponds to a postman walk in M.

Define the **weight** of a collection $C_1, \ldots, C_q$ of cycles and pseudocycles as $\sum_{i=1}^{q} |C_i|$. We can formulate CPP as the problem of finding a minimum-weight collection of cycles and pseudocycles containing all edges and arcs of M. From now on, we consider this new formulation of CPP.

Let $m := |E \cup A|$. Observe that, if $C_1, \ldots, C_q$ is an optimal solution for CPP then $q \leq m$. Indeed, suppose $C_1, \ldots, C_q$ is an optimal solution for CPP. In each C_i, there exists an edge or arc that appears in no other (pseudo)cycle C_j (otherwise, by removing C_i we would get a better solution). As a consequence, no edge or arc of M appears in more than m (pseudo)cycles.

Before describing the algorithm, we need some additional notation. Let S be a subset of V. An S-**configuration** is a function $d : S \to \{-m, \ldots, m\}$ such that $\sum_{v \in S} d(v) = 0$. Let M' be a subgraph of M and d be an S-configuration, for some set S of vertices of M'. If $\mathcal{P}$ is a collection of paths in M' with ends in S and $\mathcal{C}$ is a collection of (pseudo)cycles of M', then the pair $(\mathcal{P}, \mathcal{C})$ is a d-**realization** in M' if

- for each v in S, the number of paths in $\mathcal{P}$ that start at v minus the number of paths in $\mathcal{P}$ that end at v is exactly $d(v)$;
- each edge or arc of M' belongs to some path in $\mathcal{P}$ or to some (pseudo)cycle in $\mathcal{C}$.

We define the **weight** of a d-realization $(\mathcal{P}, \mathcal{C})$ as the quantity given by $\sum_{P \in \mathcal{P}} |P| + \sum_{C \in \mathcal{C}} |C|$. We say a d-realization in a graph is **optimal** if it has the smallest possible weight.

When M' is a digraph with n' vertices and m' arcs, an optimal d-realization can be found in $O((m' \log n')(m' + n' \log n'))$ time using a minimum cost flow algorithm [1, ?].

Let $(T, (W_t)_{t \in V(T)})$ be a tree-decomposition of (the underlying undirected graph of) M with tree-width k. Let $(E(W_t))_{t \in V(T)}$ and $(A(W_t))_{t \in V(T)}$ be an associated partition of Θ. Note that each set in this partition has size bounded by a function which depends only on k.

Root the tree T at an arbitrary vertex r in T. For each vertex t of T, denote by M_t the subgraph of M such that $V(M_t)$, $E(M_t)$ and $A(M_t)$ are the union, over all descendents v of t in T, of the sets W_v, $E(W_v)$ and $A(W_v)$ respectively. Note that the restriction of an optimal solution $\mathcal{C}^*$ of CPP to M_t corresponds to a collection $\mathcal{P}$ of paths and a collection $\mathcal{C}$ of (pseudo)cycles in M_t. It is not difficult to see that $(\mathcal{P}, \mathcal{C})$ corresponds to an optimal d-realization in M_t, for some W_t-configuration d. Moreover, if $(\mathcal{P}', \mathcal{C}')$ is an optimal d-realization in M_t, one can get an optimal solution of CPP in M by combining $(\mathcal{P}', \mathcal{C}')$ with the restriction of $\mathcal{C}^*$ to the graph $M - (E(M_t) \cup A(M_t))$.

The algorithm for CPP is based on the following strategy. For each vertex t of T and each W_t-configuration d, find an optimal d-realization in M_t. The algorithm solves this initially for the leaves of T and goes up in T until it reaches its root r. Note that in the particular case of the root, a d-realization for $d \equiv 0$ is a collection of (pseudo)cycles covering all arcs and edges of M, that is, it is a feasible solution for CPP. Hence, solving CPP is equivalent to finding an optimal d-realization in M for the W_r-configuration $d \equiv 0$.

2.1 Description of the algorithm

From now on, let us assume that M has tree-width at most k, where k is a fixed constant. Also, let $(T, (W_t)_{t \in V(T)})$ be a tree-decomposition of M of width at most k, with T binary. Clearly, as k is a constant, $|E(W_t)|$ and $|A(W_t)|$ are bounded by a constant, for all t.

For each vertex t of T, the algorithm builds a list of partial solutions consisting of triples $(d, \mathcal{P}(d), \mathcal{C}(d))$ where d is a W_t-configuration and $(\mathcal{P}(d), \mathcal{C}(d))$ is an optimal d-realization in M_t. Since $|W_t| \leq k+1$, there are at most $(2m+1)^{k+1}$ different W_t-configurations for each t.

The algorithm starts building these lists for the leaves of T. This can be done in polynomial time, because, as $|W_t| \leq k+1$, one can find an optimal d-realization by enumeration, for each W_t-configuration d. Going up the tree, the algorithm uses the lists of the children of a vertex to build the list for this vertex. This process is repeated and halts only when it reaches the root of the tree T.

Let us describe how the list for a vertex is obtained from the lists of its children. Let t be a vertex of the tree T and d be a W_t-configuration. Let t_1 and t_2 be the two children of t (the case where t has only one child is similar) and let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the lists of partial solutions for t_1 and t_2, respectively. Let $\pi := (d_i, \mathcal{P}(d_i), \mathcal{C}(d_i))$ be a partial solution in $\mathcal{L}_i$, for $i = 1, 2$. A ***d*-extension** for the pair (π_1, π_2) is a d-realization $(\mathcal{P}, \mathcal{C})$ in M_t whose restriction to the subgraph M_{t_i} corresponds to the partial solution $(\mathcal{P}(d_i), \mathcal{C}(d_i))$, for $i = 1, 2$. If, for each pair (π_1, π_2), we can compute an optimal d-extension for (π_1, π_2) (if one exists), then an optimal d-realization in M_t is simply an optimal d-extension for a pair that gives the smallest possible weight (considering all such pairs).

But how can we compute an optimal d-extension for a specific pair (π_1, π_2)? Each d_i is defined on W_{t_i}. A d-extension for the pair (π_1, π_2) exists only if the support of both d_1 and d_2 is contained in W_t. Extend d_1 and d_2 so that they are defined on W_t. Let $d_0 := d - (d_1 + d_2)$ and let $H := (W_t, E(W_t), A(W_t))$. Thus, to compute an optimal d-extension it suffices to find an optimal d_0-realization in H.

Let $(\mathcal{P}, \mathcal{C})$ be an optimal d_0-realization in H. For each $e = uv$ in $E(W_t)$, exactly one of the following holds:

(a) every element in $\mathcal{P} \cup \mathcal{C}$ containing e traverses e from u to v;

(b) every element in $\mathcal{P} \cup \mathcal{C}$ containing e traverses e from v to u;

(c) there are exactly two elements in $\mathcal{P} \cup \mathcal{C}$ containing e, one of them traverses e from u to v, and the other traverses e from v to u.

In order to compute an optimal d_0-realization in H, we consider all digraphs obtained from H by replacing each edge $e = uv$ of H by either (a) an arc (u, v), or (b) an arc (v, u), or (c) two arcs (u, v) and (v, u). For each one of these digraphs, we compute an optimal d_0-realization (using a minimum cost flow algorithm), and we select a d_0-realization with minimum weight. Note that the number of digraphs considered is $3^{|E(W_t)|} \leq 3^{k(k+1)/2}$. So the total time used to find an optimal d-extension of a pair (π_1, π_2) is $O(3^{k(k+1)/2}(|E(W_t) \cup A(W_t)| \log |W_t|)(|E(W_t) \cup A(W_t)| + |W_t| \log |W_t|)) = O(3^{k(k+1)/2} k^4 \log k)$.

We conclude that, given d and the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ of partial solutions for t_1 and t_2, one can find an optimal d-extension in M_t in time $O(|\mathcal{L}_1||\mathcal{L}_2| 3^{k(k+1)/2} k^4 \log k)$. The length of each list $\mathcal{L}_i$ is at most the number of W_{t_i}-configurations, which is at most $(2m + 1)^{k+1}$. Thus, one can find an optimal d-realization in M_t in polynomial time. Furthermore, one can build in polynomial time the list of all partial solutions for a vertex t from the lists for its children.

From the above, CPP in mixed graphs with tree-width bounded by k can be solved in polynomial time, completing the proof of Theorem 1.1. Unfortunately, due to its time complexity, the algorithm is only useful in practice for small values of k. One can make the algorithm a bit faster by including in the list of partial solutions for a vertex t only the partial solutions with support in $W_p \cap W_t$, where p is the parent of t in the tree T, but this does not reduce the complexity of the algorithm.

It is known that any graph problem that can be expressed as a formula in (extend) monadic second order logic can be solved in linear time on graphs with bounded tree-width [2, 7]. Many well-known graph problems can be formulated as such formulas: MAXIMUM CLIQUE, MAXIMUM INDEPENDENT SET, CHROMATIC NUMBER, and so on. However, it is still open whether there is a linear-time algorithm for CPP (and MCCP) on mixed graphs with bounded tree-width.

3 The minimum cycle cover problem

In this section, we prove Theorem 1.2. The algorithm for MCCP follows the same lines of the algorithm described above. The difference lies in how we define a partial solution for MCCP. Partial solutions for CPP include pseudocycles and these are not allowed in a solution for MCCP. Unfortunately, it is not enough simply to omit the pseudocycles from the partial solutions, because they appear naturally when we put together two partial solutions. The change has to be more radical. More information is needed so that we can put two partial solutions together without forming pseudocycles.

For the MCCP, an S-**configuration** is a function $d : \{-1, 0, 1\}^S \to \{0, \dots, m\}$. A **path-family** in M is a set of pairwise vertex-disjoint paths in M. Let $F := \{P_1, \dots, P_t\}$ be a path-family in M, and assume that, for each $i = 1, \dots, t$, P_i is a path from u_i to v_i. For z in $\{-1, 0, 1\}^S$, we say F **covers** z if $\{s \in S : z(s) = 1\} = \{u_1, \dots, u_t\}$, and $\{s \in S : z(s) = -1\} = \{v_1, \dots, v_t\}$.

Let d be an S-configuration, where S is a set of vertices of a subgraph M' of M. If $\mathcal{F}$ is a collection of path-families in M' (with their paths having ends in S) and $\mathcal{C}$ is a collection of cycles in M', we say $(\mathcal{F}, \mathcal{C})$ is a d-**realization** in M' if

- there exists a partition $\mathcal{F} := \bigcup_{z \in \{-1,0,1\}^S} \mathcal{F}_z$ such that $\mathcal{F}_z$ consists of exactly $d(z)$ path-families in M' that cover z;
- each edge or arc of M' belongs to a path in some path-family in $\mathcal{F}$ or to some cycle in $\mathcal{C}$.

The **weight** of a d-realization $(\mathcal{F}, \mathcal{C})$ is defined as the quantity given by $\sum_{F \in \mathcal{F}} \sum_{P \in F} |P| + \sum_{C \in \mathcal{C}} |C|$. A d-realization in a graph is **optimal** if it has the smallest possible weight.

Note that the restriction of a path-family to a subgraph M_t is also a path-family and the restriction of a cycle to M_t is either a cycle or a path-family. Moreover, if t is an arbitrary vertex of T, then the restriction of an optimal solution $\mathcal{C}^*$ for MCCP to the subgraph M_t corresponds to a collection $\mathcal{F}$ of path-families in M_t together with a collection $\mathcal{C}$ of cycles in M_t. It is not difficult to see that $(\mathcal{F}, \mathcal{C})$ corresponds to an optimal d-realization for some W_t-configuration d. Also, if $(\mathcal{F}', \mathcal{C}')$ is another optimal d-realization in M_t, then the restriction of $\mathcal{C}^*$ to the graph $M - (E(M_t) \cup A(M_t))$ combined with $(\mathcal{F}', \mathcal{C}')$ results in another optimal solution for MCCP in M. Note that, when $t = r$ and $d \equiv 0$, a d-realization in M_t $(= M)$ is a collection of cycles covering all arcs and edges of M, that is, it is a feasible solution of MCCP. Thus, solving MCCP is equivalent to finding an optimal d-realization in M for the W_r-configuration $d \equiv 0$.

The algorithm works as the previous one. Given a tree-decomposition $(T, (W_t)_{t \in V(T)})$ of M, with width at most k, where T is a binary tree, it builds, for each vertex t of T, a list of partial solutions for the MCCP in M_t. Each partial solution consists of a triple $(d, \mathcal{F}(d), \mathcal{C}(d))$, where d is a W_t-configuration and $(\mathcal{F}(d), \mathcal{C}(d))$ is an optimal d-realization. Since $|W_t| \leq k + 1$, there are at most $(m + 1)^{3^{k+1}}$ different W_t-configutations, for each t. For the leaves of T, the lists are built directly by brute force. For each internal vertex t, let t_1 and t_2 be its two children. (The case where t has only one child is similar.) Let us describe how the list for t is obtained from the lists for t_1 and t_2. More specifically, let us show how to extend a pair of partial solutions (π_1, π_2), with $\pi_i := (d_i, \mathcal{F}(d_i), \mathcal{C}(d_i))$ for $i = 1, 2$, to obtain a d-realization $(\mathcal{F}, \mathcal{C})$ in M_t. Such a d-realization is called a d-**extension** of (π_1, π_2) and is such that its restriction to M_{t_i} is exactly π_i for $i = 1, 2$. Specifically, $(\mathcal{F}, \mathcal{C})$ satisfies

- $\mathcal{C}(d_1) \cup \mathcal{C}(d_2) \subseteq \mathcal{C}$;
- the set of arcs and edges of each path-family in $\mathcal{F}$ and each cycle in $\mathcal{C} \setminus (\mathcal{C}(d_1) \cup \mathcal{C}(d_2))$ can be decomposed into L, F_1 and F_2, where L is a path-family in the graph $(W_t, E(W_t) \cup A(W_t))$ and, for $i = 1, 2$, either $F_i = \emptyset$ or $F_i \in \mathcal{F}(d_i)$;
- each path-family of $\mathcal{F}(d_i)$, $i = 1, 2$, appears in the decomposition of exactly either one path-family in $\mathcal{F}$ or one cycle in $\mathcal{C} \setminus (\mathcal{C}(d_1) \cup \mathcal{C}(d_2))$.

For $i = 1, 2$, let z_i in $\{-1, 0, 1\}^{W_t}$ be such that $d_i(z_i) > 0$, and let F_i be a path-family in $\mathcal{F}(d_i)$ that covers z_i. Let L be a path-family in $(W_t, E(W_t), A(W_t))$. Then L, F_1 and F_2 determine, besides possibly some cycles, a path-family in M_t covering some z in $\{-1, 0, 1\}^{W_t}$. Note that z depends only on the choice of L, z_1 and z_2. (Here we used the fact that the paths in the path-families are disjoint and F_1 and F_2 are contained in M_{t_1} and M_{t_2}, respectively.) So we say simply that L, z_1 and z_2 (instead of L, F_1 and F_2) determine a path-family in M_t and we call the triple (L, z_1, z_2) a **candidate path-family in** M_t. Similarly, for each z in $\{-1, 0, 1\}^{W_t}$ such that $d_i(z) > 0$, for $i = 1$ or 2, we can do the same with only L and z, and we also say (L, z) is an candidate path-family in M_t. A candidate path-family in M_t **covers** z if it determines a path-family in M_t that covers z. We can do the same with a cycle (instead of a path-family) in M_t. In this case, we say (L, z_1, z_2) (or (L, z)) is a **candidate cycle in** M_t. Each candidate path-family (cycle) represents several path-families (cycles) in M_t.

We reduce the problem of finding an optimal d-extension of (π_1, π_2) to the problem of enumerating all feasible solutions of an integer linear program of constant size. We use candidate path-families (cycles) to avoid enumerating all possible path-families and cycles in M_t. In this way, we need only to consider triples or pairs of subsets of a bounded-size set.

Let us describe the above mentioned integer linear program.

Let D_1 and D_2 be two disjoint copies of $\{-1, 0, 1\}^{W_t}$. Let $\mathbf{P}$ be a matrix with rows indexed by $R := E(W_t) \cup A(W_t) \cup D_1 \cup D_2$, whose columns are the incidence vectors of candidate path-families in M_t. Analogously, let $\mathbf{C}$ be a matrix with rows indexed by R, whose columns are the candidate cycles in M_t. For z in R, denote by $\mathbf{P}_z$ (respectively $\mathbf{C}_z$) the row of $\mathbf{P}$ (respectively $\mathbf{C}$) corresponding to z. Note that the size of R is $|E(W_t)| + |A(W_t)| + 2(3^{|W_t|}) \leq k(k+1)/2 + k(k+1) + 2(3^{k+1})$. Also, the number of columns of these two matrices is bounded by $2^{|E(W_t) \cup A(W_t)|}|D_1||D_2| \leq 2^{3k(k+1)/2} + 3^{2(k+1)}$. So, $\mathbf{P}$ and $\mathbf{C}$ have constant size.

For z in $\{-1, 0, 1\}^{W_t}$, denote by $\mathcal{K}(z)$ the set of candidate path-families in M_t that cover z. Consider pairs (x, y), where x is a non-negative integer vector indexed by candidate path-families in M_t and y is a non-negative integer vector indexed by candidate cycles in M_t, satisfying the following conditions:

$$\begin{array}{rcll} \sum_{F\in\mathcal{K}(z)} x_F & = & d(z) & \forall z \in \{-1,0,1\}^{W_t}; \\ \mathbf{P}_e x + \mathbf{C}_e y & \geq & 1 & \forall e \in E(W_t) \cup A(W_t); \\ \mathbf{P}_z x + \mathbf{C}_z y & = & d_1(z) & \forall z \in D_1; \\ \mathbf{P}_z x + \mathbf{C}_z y & = & d_2(z) & \forall z \in D_2. \end{array}$$

Let us show that there is a correspondence between integer solutions (x, y) and extensions of (π_1, π_2). Clearly, an extension of (π_1, π_2) corresponds to an integer solution (x, y). Now, suppose that (x, y) is an integer vector satisfying the integer linear program above. Let $\mathcal{F}$ be the collection of candidate path-families in M_t, with each candidate path-family F appearing exactly x_F times, and let $\mathcal{C}$ be the collection of candidate cycles in M_i, with each candidate cycle C appearing exactly y_C times. Recall that the pair $(\mathcal{F}(d_i), \mathcal{C}(d_i))$ is a d_i-realization in M_{t_i}, for $i = 1, 2$. The pair (x, y) forces the occurrence in $\mathcal{F} \cup \mathcal{C}$ of exactly $d_i(z)$ candidate path-families covering z in D_i. Replacing these candidate path-families in $\mathcal{F}, \mathcal{C}$ by path-families in $\mathcal{F}(d_1) \cup \mathcal{F}(d_2)$ for each z, we obtain a collection of path-families and cycles in M_t, which we call $\hat{\mathcal{F}}$ and $\hat{\mathcal{C}}$ respectively. It is easy to see that $(\hat{\mathcal{F}}, \hat{\mathcal{C}} \cup \mathcal{C}(d_1) \cup \mathcal{C}(d_2))$ is a d-realization in M_t.

Thus, to find an optimal extension, it is enough to find a pair (x, y) satisfying the system above and that minimizes the weight of the corresponding realization. Clearly, we may assume that the components of x and y lie in $\{0, \ldots, m\}$. So, an upper bound for the number of feasible solutions of the system is $(m+1)^{g(k)}$, where $g(k) = O(2^{3k(k+1)/2} + 3^{2(k+1)})$ is the sum of the number of columns of $\mathbf{P}$ and $\mathbf{C}$.

Therefore, given d and $\pi_i := (d_i, \mathcal{F}(d_i), \mathcal{C}(d_i))$, $i = 1, 2$, one can find an optimal d-realization (π_1, π_2) in time $O(g(k) + m^{g(k)})$. Given d and the lists $\mathcal{L}_1$,$\mathcal{L}_2$ of partial solutions for t_1,t_2, one can compute an optimal d-realization in M_t in time $|\mathcal{L}_1||\mathcal{L}_2|O(g(k) + m^{g(k)})$. The length of each list $\mathcal{L}_i$ is the number of W_{t_i}-configurations, which is at most $(m+1)^{3^{k+1}}$. Hence, one can find an optimal d-realization in M_t in polynomial time. Furthermore, one can build in polynomial time a list of partial solutions for a vertex t from the lists for its children repeating the process for all possible W_t-configurations. This concludes the proof of Theorem 1.2.

4 Concluding remarks

We hope the approach we described in this paper may help in the design of algorithms for other problems on graphs with bounded tree-width. We observe that the series-parallel graphs have tree-width at most 2. Although they constitute a very simple class of graphs it is not always immediate how to design polynomial-time algorithms for problems on these graphs. Viewing them as graphs with bounded tree-width may be a good strategy to deal algorithmically with them.

5 Acknowledgments

We would like to thank Robin Thomas for helpful discussions, and an anonymous referee for several suggestions that helped to improve this paper.

References

[1] R.K. Ahuja, T.L. Magnanti e J.B. Orlin. *Network Flows.* Prentice-Hall, 1993.

[2] S. Arnborg, J. Lagergren and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, **12** (1991), 308–340.

[3] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete App. Math.*, **23** (1989), 11–24.

[4] V. Batagelj and T. Pisanski. On partially directed Eulerian multigraphs. *Publ. de l'Inst. Math. Soc.*, **25** (1979), 16–24.

[5] H.L. Bodlaender. Dynamic programming on graphs of bounded treewidth. In *Proc. 15th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science*, 317, Springer-Verlag, Berlin (1988), 631–643.

[6] J.A. Bondy and U.S.R. Murty (1976) *Graph Theory with Applications.* MacMillan Press.

[7] B. Courcelle. The monadic second order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, **85** (1990), 12–75.

[8] R. Diestel. *Graph Theory*, Second edition. Graduate Texts in Mathematics, 173. Springer-Verlag, New York, 2000.

[9] J. Edmonds. The Chinese postman problem. *Operations Research*, **13** (1965), B73–B77.

[10] J. Edmonds and E.L. Johnson. Matching, Euler tours and the Chinese postman problem. *Math. Programming*, **5** (1973), 88–124.

[11] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton U. Press, 1973.

[12] U. Jamshy and M. Tarsi. Shortest cycle covers and the cycle double cover conjecture. *J. Comb. Theory Ser. B*, **56** (1992), 197–204.

[13] O. Lee and Y. Wakabayashi, On the circuit cover problem for mixed graphs, *Combinatorics, Probability and Computing*, **11** (2002), 43–59.

[14] J.B. Orlin, A faster strongly polynomial minimum cost flow algorithm. *Oper. Res.* **41** (1993), no. 2, 338–350.

[15] C.H. Papadimitriou, On the complexity of edge traversing, *Journal of ACM*, **23** (1976), 544–554.

[16] B. Reed. Tree width and tangles: a new connectivity measure and some applications. In *Surveys in combinatorics, 1997* (London), 87–162, London Math. Soc. Lecture Note Ser., 241, Cambridge Univ. Press, Cambridge, 1997.

[17] C. Thomassen. On the complexity of finding a minimum cycle cover of a graph. *SIAM J. Comput.*, **26** (1997), 675–677.

The UPS Problem

Cristina G. Fernandes[1,*] and Till Nierhoff[2,**]

[1] Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo - Brazil
cris@ime.usp.br
[2] Institut für Informatik
Humboldt-Universität zu Berlin
nierhoff@informatik.hu-berlin.de

Abstract. The UPS Problem consists of the following: given a vertex set V, vertex probabilities $(p_v)_{v \in V}$, and distances $l : V^2 \to R^+$ that satisfy the triangle inequality, find a Hamilton cycle such that the expected length of the shortcut that skips each vertex v with probability $1 - p_v$ (independently of the others) is minimum. This problem appears in the following context. Drivers of delivery companies visit customers daily to deliver packages. For the company, the shorter the distance traversed, the better. For a driver, routes that change dramatically from one day to the other are inconvenient; it is better if one only has to shortcut a fixed route. The UPS problem, whose objective captures these two points of view, is at least as hard to approximate as the Metric TSP. Given that one of the vertices has probability one, we show that the performance ratio of a TSP tour for the UPS problem is $1/p_{\min}$, where $p_{\min} := \min_{v \in V} p_v$. We also show that this is tight. Consequently, Christofides' algorithm for the TSP has a performance ratio of $3/(2p_{\min})$ for the UPS problem and the approximation threshold for the UPS problem is at most $1/p_{\min}$ times the one for the TSP.

1 Introduction

1.1 Motivation

Package delivery companies, like the *United Parcel Service (UPS)*, have to deliver packages daily to several of their customers. The order of delivery is chosen so that to minimize the distance traversed by the drivers. Each delivery concerns only a subset of the customers. Therefore each delivery could be optimized individually. It is, however, easier for a driver

* Research partially done while at Humboldt-Universität zu Berlin, supported in part by CAPES/DAAD Proc. 089/99, CNPq Proc. 301174/97-0, FAPESP Proc. 96/04505-2 and ProNEx 107/97 - MCT/FINEP (Brazil).
** Research supported in part by Deutsche Forschungsgemeinschaft, Pr296/6-1.

to shortcut a fixed route than to travel each time a completely different route. In this paper we study a variation of the *Traveling Salesman Problem (TSP)* which captures the issue described above.

The delivery company has the information on how often each customer receives a package. From this information one can estimate the probability that a customer receives a package per day. Roughly speaking, the here called *UPS Problem* consists of the following: find an ordering of all customers that minimizes the expected length of the route that starts and ends at a company location and visits in this ordering a randomly chosen (according to the customer's estimated probabilities) subset of the customers. The problem, as well as the described application, was proposed by [8]. Also according to [8], even the special case where customers are divided into two clusters—the customers who receive packages often and the ones who receive packages not so often—is of interest. The setup is conceivable in other delivery systems as well. We therefore expect the study of this problem to have several applications.

1.2 Notation and Problem Statement

Let $G = (V, E)$ be the complete graph on n vertices. A *path* is a sequence $\langle v_0, v_1, \dots, v_k \rangle$ of distinct vertices of G. A *cycle* is a sequence $\langle v_0, v_1, \dots, v_k \rangle$ of vertices of G, where $v_0, v_1, \dots, v_{k-1}$ are distinct and $v_k = v_0$. For a path (cycle) $P = \langle v_0, v_1, \dots, v_k \rangle$, we denote by $V(P)$ the set $\{v_0, v_1, \dots, v_k\}$ and we say that P is a path (cycle) on $V(P)$. A *tour* (or a *Hamilton cycle*) is a cycle on V.

We denote the set $\{(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)\}$ by $E(P)$. The *length of P with respect to a function $l : V^2 \to R^+$* is denoted by $l(P)$, and is given by

$$l(P) := \sum_{e \in E(P)} l(e).$$

The *Traveling Salesman Problem (TSP)* is the following: given a complete graph $G = (V, E)$ and a function $l : V^2 \to R^+$, find a tour of minimum length. We refer to such a tour as a *TSP tour*.

Unless specified otherwise, we consider in the following only functions $l : V^2 \to R^+$ that satisfy the *triangle inequality*: for any x, y, z in V, $l(x, y) \leq l(x, z) + l(z, y)$. Under this condition, TSP is called *Metric TSP*.

Note that l may be given partially. Then the length of an edge is considered to be the infimum of the lengths of all paths between its end vertices. This *closure* satisfies the triangle inequality.

Given a path P and a subset S of $V(P)$, the *shortcut of P induced by S*, denoted by $sc_S(P)$, is the path on S given by the subsequence of P containing exactly the vertices of S. Similarly we can define the *shortcut of a cycle C induced by a subset S of $V(C)$* and denote it by $sc_S(C)$.

Assume each vertex v in V has an associated probability p_v and let $\mathbf{p} := (p_v)_{v \in V}$. These probabilities induce a probability distribution on the subsets of V: for each $S \subseteq V$,

$$Pr\,[S] := \prod_{v \in S} p_v \prod_{v \notin S} (1 - p_v).$$

Let $\mathbf{p}$ and $\mathbf{q}$ be two sets of vertex probabilities. We say that $\mathbf{p}$ dominates $\mathbf{q}$ if $p_v \geq q_v$ for all $v \in V$.

Given a cycle C, denote by $lsc_{\mathbf{p}}(C)$ the expected value of the length of the shortcut of C induced by a vertex subset which is randomly chosen according to $\mathbf{p}$:

$$lsc_{\mathbf{p}}(C) := \sum_{S \subseteq V} \Pr\,[S]\, l(sc_S(C)).$$

Now we are ready to state the UPS problem:

Definition 1. *Given the complete graph $G = (V, E)$, a function $l : V^2 \to R^+$ satisfying the triangle inequality, and probabilities $\mathbf{p} = (p_v)_{v \in V}$, the* UPS problem *asks for a tour C that minimizes $lsc_{\mathbf{p}}(C)$.*

The performance ratio of a tour C for the UPS problem is the ratio $lsc_{\mathbf{p}}(C)/\text{opt}$, where opt denotes the optimal value of the UPS problem, that is, opt $= lsc_{\mathbf{p}}(C^{\text{UPS}})$ for some optimal tour C^{UPS} of the UPS problem.

Throughout the paper, we consider the UPS problem under the additional assumption that $p_u = 1$ for at least one vertex u (representing, say, a UPS location).

1.3 Results

We start studying a restricted class of vertex probabilities for the UPS problem. Let G, l, and $\mathbf{p}^*$ be the input of the UPS problem, where we assume that, for some $0 \leq p \leq 1$, $p^*_v \in \{p, 1\}$ for all v. Our first result is a lower bound on the objective function for this particular case in terms of the TSP optimum.

Theorem 1. *Let C be a tour and C^{TSP} be a TSP tour. Then $lsc_{\mathbf{p}^*}(C) \geq p \cdot l(C^{TSP})$.*

The assumption that the vertex probabilities only attain the values p and 1 can be removed with the help of the following proposition.

Proposition 1. *Let* $\mathbf{p}$ *and* $\mathbf{q}$ *be two sets of vertex probabilities, where* $\mathbf{p}$ *dominates* $\mathbf{q}$*. Then* $lsc_{\mathbf{q}}(C) \leq lsc_{\mathbf{p}}(C)$ *for any tour* C*.*

The performance ratio of C^{TSP} as a solution for the general UPS problem follows from Theorem 1, using Proposition 1:

Corollary 1. *Let* C^{TSP} *be a TSP tour, let* C^{UPS} *be an optimal solution of the UPS problem, and let* $p_{\min} := \min_{v \in V} p_v$*. Denote by* opt *the optimal value* $lsc_{\mathbf{p}}(C^{UPS})$*. Then*

$$\frac{lsc_{\mathbf{p}}(C^{TSP})}{\mathrm{opt}} \leq \frac{1}{p_{\min}}.$$

Our second result is that this bound is tight.

Theorem 2. *For every* $\epsilon > 0$ *there is an instance* I_ϵ *to the UPS problem such that there are two TSP tours* C_1 *and* C_2 *with* $lsc_{\mathbf{p}}(C_2) \leq (p_{\min} + \epsilon) lsc_{\mathbf{p}}(C_1)$*.*

The tightness then follows from $\mathrm{opt} \leq lsc_{\mathbf{p}}(C_2)$.

When studying approximations for a computational problem, it is certainly necessary to explore the complexity theoretical limitations of that approach. We prove the following hardness of approximation result.

Theorem 3. *The approximation threshold of the UPS problem with any constantly bounded probability set is not less than the approximation threshold of the Metric TSP.*

The proof of Theorem 1 will be given in Section 2. In Section 3 we will sketch the proof of Proposition 1 and give more details on Corollary 1. Theorems 2 and 3 will be proved in Sections 4 and 5, respectively.

1.4 Conclusions and Open Problems

The UPS problem extends the TSP in that not only the length of the tour, but also the lengths of the subtours determine its objective value. As one might expect, the tradeoff depends on the vertex probabilities. We give matching upper and lower bounds on the rate in Theorems 1 and 2.

This result, being interesting in its own right, has several consequences for the approximation properties of the UPS problem. The currently best known approximation algorithm for the Metric TSP, Christofides' algorithm [3], has a performance ratio of $\frac{3}{2}$. As a consequence of Corollary 1, the same algorithm has a performance ratio of $\frac{3}{2p_{\min}}$ for the UPS problem.

Similarly, every other approximation algorithm for the Metric TSP can be applied to the UPS problem, while the performance ratio is multiplied by a factor of $\frac{1}{p_{\min}}$. Thus, the approximation threshold of the UPS problem is at most $\frac{\theta}{p_{\min}}$, where θ is the approximation threshold for the Metric TSP (for the definition of the approximation threshold and related notions see, e.g., Chapter 13 in [5]). This fact is complemented by Theorem 3, which states that it is at least θ.

One of the first questions that one might ask in this context concerns the influence of different probabilities. The factor of $\frac{1}{p_{\min}}$ might seem too pessimistic, if there were only few vertices with probability $p_{\min}$ and lots of vertices with much larger probabilities. However, the tight examples given in the proof of Theorem 2 can be modified so as to show that the bound given in Theorem 1 is very accurate.

The situation is less clear in the case of approximation algorithms. Here it is conceivable that an algorithm takes into account the distances given by l and combines them with the individual vertex probabilities in a clever way. The non-approximability result in Theorem 3 does not set any limit for that, however it is not less conceivable that the hardness result could be improved to show that the approximation threshold is actually $\frac{\theta}{p_{\min}}$.

The same consideration applies to the important special case of Euclidean instances. An instance of the TSP is called Euclidean if there is a point in the plane for every vertex such that the distance given by l is the Euclidean distance between the points. For this special case, there exist polynomial-time approximation schemes (PTAS) [2, 4] for TSP. (A PTAS consists of a polynomial-time algorithm for the problem with a performance ratio of at most $1+\epsilon$, for each $\epsilon > 0$.) Thus, for each $\epsilon > 0$, by Theorem 1 there exists a polynomial time algorithm for the UPS problem with Euclidean instances with a performance ratio of at most $\frac{1+\epsilon}{p_{\min}}$. On the other hand, it is conceivable both that there is a PTAS and that the approximation threshold is up to $\frac{1}{p_{\min}}$.

2 The UPS Problem with Probabilities 1 or p

The aim of this section is to prove Theorem 1. Recall that there is a $u \in V$ with $p^*_u = 1$ and that, for that theorem, the vertex probabilities are restricted to values of p and 1.

Let C^{TSP} be a TSP tour and C be a tour. To bound $lsc_{\mathbf{p}^*}(C)$ in terms of $l(C^{\text{TSP}})$ we first need another formulation for the corresponding expectation. To this end we introduce some more notation.

Let U be the set of vertices of probability 1, $\bar{U} := V \setminus U$, and let t be the number of vertices in $\bar{U}$. If $t = 0$ then $U = V$ and $lsc_{\mathbf{p}^*}(C) = l(C)$. Since $l(C) \geq l(C^{\text{TSP}})$, the theorem clearly holds in this case. So we may assume $t \geq 1$.

For every $u, v \in \bar{U}$, let P_{uv} be the subsequence of C beginning at u and ending at v (circularly). Let P^1_{uv} be the shortcut of P_{uv} induced by $\{u, v\} \cup U$. Note that P^1_{vv} denotes a cycle—the shortcut of C induced by $\{v\} \cup U$.

Denote by $v_0, v_1, \ldots, v_{t-1}$ the vertices of $\bar{U}$ in the order given by C. For $i = 0, \ldots, t-1$, set $\mathcal{C}_i := \{P^1_{v_j v_{j+i+1}} : 0 \leq j < t\}$, where indices are taken modulo t, and $l(\mathcal{C}_i) := \sum_{P \in \mathcal{C}_i} l(P)$. Each $\mathcal{C}_i$ is a collection of paths (cycles if $i = t-1$) in G whose concatenation results in an Eulerian subgraph of G. Because $U \neq \emptyset$ and each vertex in $\bar{U}$ appears in some path (cycle if $i = t-1$) in $\mathcal{C}_i$, each of these Eulerian subgraphs is connected and spanning. Therefore, for each i,

$$l(\mathcal{C}_i) \geq l(C^{\text{TSP}}). \tag{1}$$

Using the notation above we can give the following characterization of $lsc_{\mathbf{p}^*}(C)$:

Lemma 1. $lsc_{\mathbf{p}^*}(C) = (1-p)^t l(sc_U(C)) + p(1-p)^{t-1} l(\mathcal{C}_{t-1}) + \sum_{i=0}^{t-2} p^2 (1-p)^i l(\mathcal{C}_i)$.

Proof. By definition, $lsc_{\mathbf{p}^*}(C) = \sum_{U \subseteq S \subseteq V} \Pr[S]\, l(sc_S(C))$. Note that the summands where $|S \setminus U| \leq 1$ contribute with $(1-p)^t l(sc_U(C)) + p(1-p)^{t-1} l(\mathcal{C}_{t-1})$ to $lsc_{\mathbf{p}^*}(C)$.

Let $\mathcal{S} := \{S : U \subseteq S \subseteq V, |S \setminus U| \geq 2\}$ be the collection of the other vertex subsets. For any $S \in \mathcal{S}$, let $E_S := E(sc_{S \setminus U}(C))$. Then, adding indices modulo t,

$$\begin{aligned}
\sum_{S \in \mathcal{S}} \Pr[S]\, l(sc_S(C)) &= \sum_{S \in \mathcal{S}} \Pr[S] \sum_{(v_i, v_j) \in E_S} l(P^1_{v_i v_j}) \\
&= \sum_{i=0}^{t-1} \sum_{j=0, j \neq i}^{t-1} l(P^1_{v_i v_j}) \sum_{\substack{S \in \mathcal{S} \\ (v_i, v_j) \in E_S}} \Pr[S] \\
&= \sum_{i=0}^{t-2} \sum_{j=0}^{t-1} l(P^1_{v_j v_{j+i+1}}) \Pr[(v_j, v_{j+i+1}) \in E_S] \\
&= \sum_{i=0}^{t-2} l(\mathcal{C}_i) p^2 (1-p)^i
\end{aligned}$$

and the lemma holds. □

Putting (1) and Lemma 1 together, we have that

$$\begin{aligned} lsc_{\mathbf{p}^*}(C) &\geq p(1-p)^{t-1} l(C^{\mathrm{TSP}}) + \sum_{i=0}^{t-2} p^2(1-p)^i l(C^{\mathrm{TSP}}) \\ &= l(C^{\mathrm{TSP}}) \left(p(1-p)^{t-1} + \sum_{i=0}^{t-2} p^2(1-p)^i \right). \end{aligned}$$

Straightforward calculation shows that the right hand side is equal to $l(C^{\mathrm{TSP}})p$, concluding the proof of Theorem 1. □

3 Arbitrary Probabilities

Our result on the UPS problem with arbitrary probabilities, i.e. Corollary 1, is a consequence of Proposition 1. The proof of Proposition 1 is based on the FKG-Inequality [1, p. 75] and we only sketch it here.

Let $\mathbf{p}$ and $\mathbf{q}$ be two sets of vertex probabilities and assume that $\mathbf{p}$ dominates $\mathbf{q}$. Let C be any tour. For $S \subseteq V$, let $f(S) := l(sc_S(C))$ and $g(S) := \prod_{v \in S}(p_v/q_v) \cdot \prod_{v \notin S}(1-p_v)/(1-q_v)$. Observe that $S \mapsto \mathrm{Pr}_{\mathbf{q}}[S]$ is log-supermodular, that f is increasing because of the triangle inequality, and that g is increasing because $\mathbf{p}$ dominates $\mathbf{q}$. Thus, the requirements of the FKG inequality are met and we have

$$\sum_{S \subseteq V} f(S)\mathrm{Pr}_{\mathbf{q}}[S] \cdot \sum_{S \subseteq V} g(S)\mathrm{Pr}_{\mathbf{q}}[S] \leq \sum_{S \subseteq V} f(S)g(S)\mathrm{Pr}_{\mathbf{q}}[S] \cdot \sum_{S \subseteq V} \mathrm{Pr}_{\mathbf{q}}[S].$$

Since $\mathrm{Pr}_{\mathbf{p}}[S] = \mathrm{Pr}_{\mathbf{q}}[S] \cdot g(S)$, this implies Proposition 1. □

Corollary 1 follows by sandwiching $\mathbf{p}$ between two appropriate sets of vertex probabilities. More precisely, let

$$p_v^* := \begin{cases} p_{\min} & \text{if } p_v < 1, \\ 1 & \text{otherwise.} \end{cases}$$

Then $\mathbf{p}^*$ is dominated by $\mathbf{p}$, which in turn is dominated by the all-ones probability set. The corollary follows from

$$\mathrm{opt} = lsc_{\mathbf{p}}(C^{\mathrm{UPS}}) \geq lsc_{\mathbf{p}^*}(C^{\mathrm{UPS}}) \geq p_{\min}\, l(C^{\mathrm{TSP}}) \geq p_{\min}\, lsc_{\mathbf{p}}(C^{\mathrm{TSP}}),$$

where the first and the last inequality are implied by Proposition 1 and the second inequality by Theorem 1, applied to $C = C^{\mathrm{UPS}}$.

4 Tight Examples

Assume that $p < 1$ and let $\epsilon > 0$. In this section we give the construction of an instance I_ϵ of the UPS problem with probabilities p and 1. It has two TSP tours C_1 and C_2 and (3) states that their UPS values differ by a factor of at least $\frac{1}{p+\epsilon}$. This implies Theorem 2.

We assume w.l.o.g. that $\epsilon < 1 - p$. Let k be a positive integer, large enough so that $k + \log_{1-p}(8k^2) \geq \log_{1-p} \epsilon$. Let n be a prime such that $2k^2 < n < 4k^2$. Then

$$2n(1-p)^k \leq 8k^2(1-p)^k \leq \epsilon. \tag{2}$$

Let $V := \{0, \ldots, n-1\}$ and let $H := (V, E)$, where

$$E := \{ij : j - i \pmod{n} \leq k\}.$$

That is, $H = C_n^k$ is a cycle on n vertices plus all chords of length at most k. Let $l(e) = 1$ for all $e \in E$. Then two TSP tours for H and l are (indices are taken modulo n, as usual)

$$C_1 := \langle 0, k, \ldots, ik, \ldots, nk \rangle \text{ and}$$
$$C_2 := \langle 0, 1, \ldots, i, \ldots, n \rangle.$$

Note that C_1 is a tour because of the primality of n.

Let $p_0 := 1$ and $p_i := p$ for $i \geq 1$. Then I_ϵ consists of (the closure of) H, l, and $\mathbf{p}$. In the rest of this section we shall prove that

$$lsc_{\mathbf{p}}(C_2) \leq (p + \epsilon) lsc_{\mathbf{p}}(C_1). \tag{3}$$

Let S be a randomly chosen subset of V. Call S *dense for* C_1 if S intersects any set of k consecutive vertices of C_1. The probability of that event is at most $n(1-p)^k$, where $n(1-p)^k \leq \epsilon/2$ by (2). The event that S is *dense for* C_2 is defined analogously, and its probability is the same.

Assume that S is dense for C_1 and let ik and $jk > ik$ be two subsequent vertices of $sc_S(C_1)$. Then $j - i \leq k$ because S is dense for C_1. But then $jk - ik \leq k^2 < n/2$, and by the choice of H the distance between ik and jk is $j - i$. Assume that $sc_S(C_1) = \langle i_1 k, \ldots, i_{|S|} k \rangle$ and let $i_0 := i_{|S|}$. Then

$$n = \sum_{j=1}^{|S|} (i_j - i_{j-1} \pmod{n}) = \sum_{j=1}^{|S|} l(i_j k, i_{j-1} k) = l(sc_S(C_1)),$$

and therefore

$$lsc_{\mathbf{p}}(C_1) \geq n\mathrm{P}r\left[S \text{ is dense for } C_1\right] \geq (1-\epsilon/2)n. \tag{4}$$

If S is dense for C_2, then there is a chord between any two subsequent vertices of $sc_S(C_2)$ and thus $l(sc_S(C_2)) = |S|$. This implies that

$$lsc_{\mathbf{p}}(C_2) \leq n\mathrm{P}r\left[S \text{ is not dense for } C_2\right] + \sum_{S\subseteq V} |S|\mathrm{P}r\left[S\right] \leq (p+\epsilon/2)n. \tag{5}$$

As a consequence of (4) and (5),

$$\frac{lsc_{\mathbf{p}}(C_2)}{lsc_{\mathbf{p}}(C_1)} \leq \frac{p+\epsilon/2}{1-\epsilon/2},$$

which implies (3), using $\epsilon < 1-p$. □

5 Hardness of Approximation

The Metric TSP is APX-complete [7] and the currently best lower bound on the approximation threshold is $\frac{41}{40}$ in the asymmetric case and $\frac{129}{128}$ in the symmetric case [6]. It is trivial that the UPS problem has the same lower bounds, because the objective functions coincide for the all-ones probability set $p_v \equiv 1$. It might, however, be interesting to verify that the same holds for the probability set $p_v \equiv p$, where $0 < p < 1$. This, together with Proposition 1, proves Theorem 3.

Next we present a reduction from Metric TSP to the UPS problem in instances with probability set $p_v \equiv p$ for each $\epsilon > 0$. The reduction preserves the approximation ratio up to a factor of $1-\epsilon$.

Let V be a vertex set and let $l : V^2 \to R^+$ be an instance of the Metric TSP on V. The corresponding instance of the UPS problem consists of the following. Add $c_{p,\epsilon}(n)$ copies of every vertex to get V' and let l' be the extension of l to V' such that all copies of the same vertex have distance 0 to each other and copies of different vertices have the same distance as the original vertices. Here $c_{p,\epsilon}(n) = O(\log_{\frac{1}{1-p}} n)$ is chosen large enough that $(1-(1-p)^{c_{p,\epsilon}(n)})^n \geq 1-\epsilon$. Note that this can be done in polynomial time.

Any tour on V' whose performance ratio (for the UPS instance) is at most η can be converted into a tour on V whose performance ratio (for the original TSP instance) is at most $\eta/(1-\epsilon)$. Indeed, let C' be a tour on V' whose performance ratio is at most η for the UPS problem. We may assume w.l.o.g. that all copies of each original vertex occur subsequently in C'. Then $l(sc_S(C')) = l(C')$ as long as S contains at least one copy of each original vertex. Since the probability for that event is at least $1-\epsilon$, we have that $lsc_{\mathbf{p}}(C') \geq (1-\epsilon)l'(C') = (1-\epsilon)l(C)$, where C is the ordering of V induced by C'. Now let C^{TSP} be a TSP tour on V. If we extend it to a tour on V' by visiting all copies of every vertex subsequently, we know that $lsc_{\mathbf{p}}(C^{\text{TSP}}) \leq l'(C^{\text{TSP}}) = l(C^{\text{TSP}})$. Therefore the optimal UPS solution has length at most $l(C^{\text{TSP}})$. Thus $(1-\epsilon)l(C) \leq lsc_{\mathbf{p}}(C) \leq \eta lsc_{\mathbf{p}}(C^{\text{UPS}}) \leq \eta l(C^{\text{TSP}})$. This completes the analysis of the reduction, and, together with Proposition 1, the proof of Theorem 3. □

6 Acknowledgment

We thank Martin Savelsbergh for suggesting the problem and Deryk Osthus and Hanno Lefmann for fruitful discussions. Carlos Eduardo Rodrigues Alves gave a simple proof of a weaker version of Theorem 1 (with p^2 instead of p), which inspired our proof. An anonymous referee gave several useful hints on the presentation of the paper.

References

1. N. Alon, J. Spencer, P. Erdős, *The Probabilistic Method*, Wiley, New York, 1992.
2. S. Arora, *Polynomial Time Approximation Schemes for Euclidean TSP and other Geometric Problems*, Proceedings of the 37th Symposium on the Foundations of Computer Science, 2–11 (1996).
3. N. Christofides, *Worst-case Analysis of a new Heuristic for the Travelling Salesman Problem*, Technical Report (Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA), 1976.
4. J. Mitchell, *Guillotine Subdivisions Approximate Polygonal Subdivisions: A Simple Polynomial-Time Approximation Scheme for Geometric TSP, k-MST, and Related Problems*, SIAM Journal on Computing 28, 1298–1309 (1999).
5. C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.
6. C.H. Papadimitriou and S. Vempala, *On the Approximability of the Traveling Salesman Problem*, Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 126–133 (2000).
7. C.H. Papadimitriou and M. Yannakakis, *The Traveling Salesman Problem with Distances One and Two*, Mathematics of Operations Research 18, 1–11 (1993).
8. M. Savelsbergh, personal communication (1999).

Edge-Coloring Series-Parallel Multigraphs

Cristina G. Fernandes [*]
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo - Brazil
E-mail: cris@ime.usp.br

and

Robin Thomas [†]
School of Mathematics
Georgia Institute of Technology
Atlanta, GA 30332-0160, USA
E-mail: thomas@math.gatech.edu

August 18, 2000

Abstract

We give a simpler proof of Seymour's Theorem on edge-coloring series-parallel multigraphs and derive a linear-time algorithm to check whether a given series-parallel multigraph can be colored with a given number of colors.

1 Introduction

All *graphs* in this paper are finite, may have parallel edges, but no loops. Let $k \geq 0$ be an integer. A graph G is *k-edge-colorable* if there exists a map $\kappa : E(G) \to \{1, \ldots, k\}$, called a *$k$-edge-coloring*, such that $\kappa(e) \neq \kappa(f)$ for any two distinct edges e, f of G that share at least one end. The *chromatic index* $\chi'(G)$ is the minimum $k \geq 0$ such that G is k-edge-colorable. Clearly $\chi'(G) \geq \Delta(G)$, where $\Delta(G)$ is the maximum degree of G, but there is another lower bound. Let

$$\Gamma(G) = \max\left\{\frac{2|E(G[U])|}{|U|-1} : U \subseteq V(G), |U| \geq 3 \text{ and } |U| \text{ is odd}\right\}.$$

If U is as above, then every matching in $G[U]$, the subgraph induced by U, has size at most $\lfloor \frac{1}{2}|U| \rfloor$. Consequently, $\chi'(G) \geq \Gamma(G)$. If G is the Petersen graph, or the Petersen graph with one vertex deleted, then $\chi'(G) > \max\{\Delta(G), \lceil \Gamma(G) \rceil\}$. However, Seymour conjectures that equality holds for planar graphs:

Conjecture 1.1 *If G is a planar graph, then $\chi'(G) = \max\{\Delta(G), \lceil \Gamma(G) \rceil\}$.*

Conjecture 1.1 most likely does not have an easy proof, because it implies the Four-Color Theorem. However, Seymour [5] proved that his conjecture holds for series-parallel graphs (a graph is *series-parallel* if it has no subgraph isomorphic to a subdivision of K_4):

Theorem 1.2 *If G is a series-parallel graph, and k is an integer with $k \geq \max\{\Delta(G), \Gamma(G)\}$ then G is k-edge-colorable.*

It should be noted that Theorem 1.2 is fairly easy for simple graphs; the difficulty lies in the presence of parallel edges. Seymour's proof is elegant and interesting, but the induction step requires the verification of a large number of inequalities. We give a simpler proof, based on a structural

[*]Research supported in part by CNPq (Brazil) Proc. No. 301174/97-0, FAPESP (Brazil) Proc. No. 98/14329 and 96/04505-2 and ProNEx (Brazil) 107/97 – MCT/FINEP.

[†]Research supported in part by NSA under Grant No. MDA904-98-1-0517.

lemma about series-parallel graphs, which in turn is an easy consequence of the well-known fact that every simple series-parallel graph has a vertex of degree at most two. Our work was motivated by the list edge-coloring conjecture of [1] (see also [3, Problem 12.20]):

Conjecture 1.3 *Every graph is $\chi'(G)$-edge-choosable.*

At present there seems to be no credible approach for proving the conjecture in full generality. We were trying to gain some insight by studying it for series-parallel graphs. The conjecture has been verified for *simple* series-parallel graphs in [4], but it is open for series-parallel graphs with parallel edges. Our efforts only resulted in a simpler proof of Theorem 1.2 and in a linear-time algorithm for checking whether or not a series-parallel graph can be colored with a given number of colors.

2 Three lemmas

For our proof of Theorem 1.2 we need three lemmas. The first two are easy, and the third appeared in [4]. Let G be a graph, and let u, v be adjacent vertices of G. We use uv to denote the unique edge with ends u and v in the underlying simple graph of G. If G has m edges with ends u and v, then we say that uv has *multiplicity* m. If u and v are not adjacent, then we say that uv has multiplicity zero. Let G be a graph, let κ be a k-edge-coloring of a subgraph H of G, let $u \in V(G)$, and let $i \in \{1, 2, \dots, k\}$. We say that u *sees* i and that i *is seen by* u if $\kappa(f) = i$ for some edge f of H incident with u.

Lemma 2.1 *Let G be a graph, let $u_0 \in V(G)$, let u_1, u_2 be distinct neighbors of u_0, let H be the graph obtained from G by deleting all edges with one end u_0 and the other end u_1 or u_2, and let κ be a k-edge-coloring of H. For $i = 1, 2$ let m_i be the multiplicity of u_0u_i in G, and for $i = 0, 1, 2$ let S_i be the set of colors seen by u_i. If $m_1 + |S_0 \cup S_1| \leq k$, $m_2 + |S_0 \cup S_2| \leq k$ and $m_1 + m_2 + |S_0 \cup (S_1 \cap S_2)| \leq k$, then κ can be extended to a k-edge-coloring of G.*

Proof. Since $m_1 + |S_0 \cup S_1| \leq k$, the edges with ends u_0 and u_1 can be colored using colors not in $S_0 \cup S_1$. We do that, using as many colors in S_2 as possible. If the u_0u_1 edges can be colored using colors in S_2 only, then there are at least $k - |S_0 \cup S_2| \geq m_2$ colors left to color the edges with ends u_0 and u_2, and so κ can be extended to a k-edge-coloring of G, as desired. Otherwise, the u_0u_1 edges of G will be colored using $|S_2 - (S_0 \cup S_1)|$ colors from S_2, and $m_1 - |S_2 - (S_0 \cup S_1)|$ other colors. Thus the number of colors available to color the u_0u_2 edges of G is at least $k - |S_0 \cup S_2| - (m_1 - |S_2 - (S_0 \cup S_1)|) = k - m_1 - |S_0 \cup (S_1 \cap S_2)| \geq m_2$, and so the coloring can be completed to a k-edge-coloring of G, as desired. ■

Lemma 2.2 *Let k be an integer, and let G be a graph with $\Delta(G) \leq k$. Then $\Gamma(G) \leq k$ if and only if $2|E(G[U])| \leq k(|U| - 1)$ for every set $U \subseteq V(G)$ such that $|U|$ is odd and at least three, and the graph $G[U]$ has no vertices of degree at most one.*

Proof. The "only if" part is clear. To prove the "if" part we must show that $2|E(G[U])| \leq k(|U|-1)$ for every set $U \subseteq V(G)$ such that $|U|$ is odd and at least three. We proceed by induction on $|U|$. We may assume that $G[U]$ has a vertex u of degree at most one, for otherwise the conclusion follows from the hypothesis. If u has degree one in $G[U]$, then let v be its unique neighbor; otherwise let $v \in U \backslash \{u\}$ be arbitrary. Let $U' = U \backslash \{u, v\}$. Then $2|E(G[U])| \leq 2\Delta(G) + 2|E(G[U'])| \leq 2k + k(|U'| - 1) \leq k(|U| - 1)$ by the induction hypothesis if $|U| > 3$ and trivially otherwise, as desired. ■

The third lemma appeared in [4]. For the sake of completeness we include its short proof.

Lemma 2.3 *Every non-null simple series-parallel graph G has one of the following:*

(a) a vertex of degree at most one,

(b) two distinct vertices of degree two with the same neighbors,

(c) two distinct vertices u, v and two not necessarily distinct vertices $w, z \in V(G)\backslash\{u, v\}$ such that the neighbors of v are u and w, and every neighbor of u is equal to v, w, or z, or

(d) five distinct vertices v_1, v_2, u_1, u_2, w such that the neighbors of w are u_1, u_2, v_1, v_2, and for $i = 1, 2$ the neighbors of v_i are w and u_i.

Proof. We proceed by induction on the number of vertices. Let G be a non-null simple series-parallel graph, and assume that the result holds for all graphs on fewer vertices. We may assume that G does not satisfy (a), (b), or (c). Thus G has no two adjacent vertices of degree two. By suppressing all vertices of degree two (that is, contracting one of the incident edges) we obtain a series-parallel graph without vertices of degree two or less. Therefore, by a well-known property of series-parallel graphs [2], this graph is not simple. Since G does not satisfy (b), this implies that G has a vertex of degree two that belongs to a cycle of length three. Let G' be obtained from G by deleting all vertices of degree two that belong to a cycle of length three. First notice that if G' has a vertex of degree less than two, then the result holds for G (cases (a), (b), or case (c) with $w = z$). Similarly, if G' has a vertex of degree two that does not have degree two in G, then the result holds (one of the cases (b)–(d) occurs). Thus we may assume that G' has minimum degree at least two, and every vertex of degree two in G' has degree two in G. By induction, (b), (c), or (d) holds for G', but it is easy to see that then one of (b), (c), or (d) holds for G. ■

3 Proof of Theorem 1.2

We proceed by induction on $|E(G)|$, and, subject to that, by induction on $|V(G)|$. The theorem clearly holds for graphs with no edges, so we assume that G has at least one edge, and that the theorem holds for graphs with fewer edges or the same number of edges but fewer vertices. Let S be the underlying simple graph of G. We apply Lemma 2.3 to S, and distinguish the corresponding cases.

If case (a) holds, let G' be the graph obtained from G by removing a vertex of degree at most one in S. The rest is straightforward: $k \geq \max\{\Delta(G'), \Gamma(G')\}$ and so, by induction, there is a k-edge-coloring of G'. From this k-edge-coloring, it is easy to obtain a k-edge-coloring for G.

If case (b) holds, let u and v be two distinct vertices of degree two in S with the same neighbors. Let the common neighbors be x and y. Let a, b, c, d be the multiplicities of ux, uy, vx, vy, respectively. See Figure 1(a). From the symmetry we may assume that $a \geq d$. Let G' be obtained

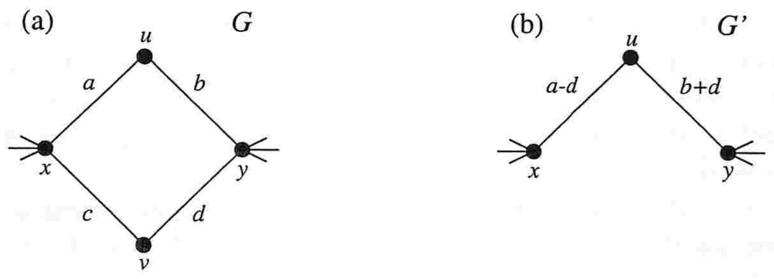


Figure 1: Configurations referring to Case (b)

from $G\backslash v$ by deleting d edges with ends u and x, and adding d edges with ends u and y. See Figure 1(b). Then clearly $\Delta(G') \le k$, and it follows from Lemma 2.2 that $\Gamma(G') \le k$. By the induction hypothesis the graph G' has a k-edge-coloring κ'. Let A be a set of colors of size d used by a subset of the edges of G' with ends u and y, chosen so that as few as possible of these colors are seen by x. By deleting those edges we obtain a coloring of $G\backslash v$, where d edges with ends u and x are uncolored. Next we color those d uncolored edges, first using colors in A not seen by x, and then using arbitrary colors not seen by x or u. This can be done: if at least one color in A is seen by x, then once we exhaust colors of A not seen by x, the choice of A implies that every color seen by u is seen by x, and so the coloring can be completed, because x has degree at most k. This results in a k-edge-coloring of $G\backslash v$ with the property that at least d of the colors seen by x (namely the colors in A) are not seen by y. Thus the number of colors seen by both x and y is at most $k-c-d$ (v sees no colors), and clearly the number of colors seen by x is at most $k-c$ and the number of colors seen by y is at most $k-d$. By Lemma 2.1 this coloring can be extended to a k-edge-coloring of G, as desired.

We now assume a special case of (c) of Lemma 2.3. Let u, v, w, z be as in that lemma, with $w = z$. Then clearly $\Delta(G\backslash v) \le k$ and $\Gamma(G\backslash v) \le k$, and so $G\backslash v$ has a k-edge-coloring. This k-edge-coloring can be extended to a k-edge-coloring of G by first coloring the edges with ends w and v (this can be done because the degree of w is at most k), and then coloring the edges with ends u and v (there are enough colors for this because $|E(G[U])| \le k$ for $U = \{u, v, w\}$).

Finally we assume that case (d) of Lemma 2.3 holds and we will show that our analysis includes the remainder of case (c) as a special case. Let v_1, v_2, u_1, u_2 and w be as in the statement of Lemma 2.3, and let a, b, c, d, e and f be the multiplicities of $u_1v_1, u_1w, v_1w, v_2w, u_2w$ and u_2v_2, respectively, as in Figure 2(a). In order to include case (c) we will not be assuming that a, b, c, d, e and f are nonzero; we only assume that $c+d > 0$. (This is why the primary induction is on $|E(G)|$.) If $a+b+c+d+e+f \le k$, then a k-edge-coloring of $G\backslash w$ can be extended to a k-edge-coloring of G, and so we may assume that $k < a+b+c+d+e+f$. Since w has degree at most k we have $b+c+d+e \le k$, and by considering the sets $U = \{u_1, v_1, w\}$ and $U = \{u_2, v_2, w\}$ we deduce that $a+b+c \le k$ and $d+e+f \le k$. Let $z_1 = \max\{0, a+b+c+e-k\}$, $z_2 = \max\{0, b+d+e+f-k\}$ and $s = k-(b+c+d+e)$. Thus $z_1 \le e$, $z_2 \le b$, $s \ge 0$ and

$$(*) \qquad a+f-z_1-z_2-s = \begin{cases} k-(b+e) & \text{if } z_1 > 0 \text{ and } z_2 > 0 \\ a+c & \text{if } z_1 = 0 \text{ and } z_2 > 0 \\ d+f & \text{if } z_1 > 0 \text{ and } z_2 = 0 \\ a+f-s & \text{if } z_1 = z_2 = 0. \end{cases}$$

We claim that there exist nonnegative integers s_1 and s_2 such that $s = s_1 + s_2$, $s_1 \le a - z_1$ and $s_2 \le f - z_2$. To prove this claim it suffices to check that $a - z_1 \ge 0$, $f - z_2 \ge 0$ and $a - z_1 + f - z_2 \ge s$. We have $a - z_1 \ge \min\{a, k-(b+c+e)\} \ge \min\{a, d\} \ge 0$, and by symmetry $f - z_2 \ge 0$. The third inequality follows from $(*)$. This proves the existence of s_1 and s_2.

Let G' be obtained from G by removing the vertices v_1, v_2, w, adding two new vertices, x and y, and adding $a - z_1 - s_1$ edges with ends x and u_1, $f - z_2 - s_2$ edges with ends x and u_2, $b - z_2$ edges with ends y and u_1, $e - z_1$ edges with ends y and u_2, and $z_1 + z_2$ edges with ends u_1 and u_2. See Figure 2(b). Thus $|E(G')| < |E(G)|$.

It follows from $(*)$ that x has degree at most k. Since all other vertices of G' clearly have degree at most k, we see that $k \ge \Delta(G')$. We claim that $k \ge \Gamma(G')$. By Lemma 2.2 we must show that $2|E(G'[X'])| \le k(|X'|-1)$ for every set $X' \subseteq V(G')$ such that $|X'|$ is odd, $|X'| \ge 3$ and $G'[X']$ has no vertices of degree at most one. If $|X' \cap \{u_1, u_2\}| \le 1$, then $G[X'] = G'[X']$, and the result follows. Thus we may assume that $u_1, u_2 \in X'$. We need to distinguish several cases. If

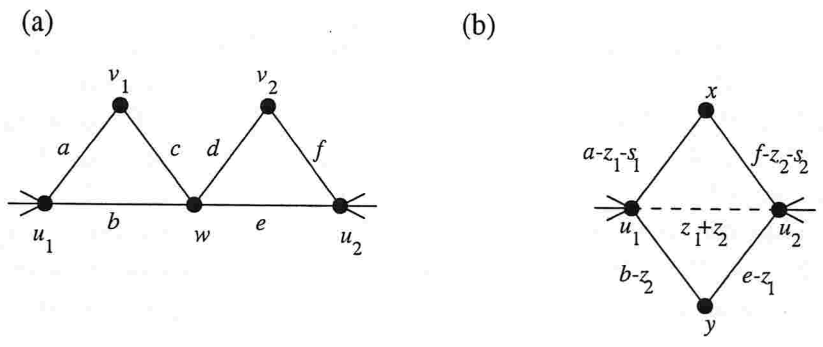


Figure 2: Configurations referring to Case (d)

$x, y \in X'$, then let $X = X' \setminus \{x, y\}$. We have $2|E(G'[X'])| = 2|E(G[X])| + 2(a - z_1 - s_1 + f - z_2 - s_2 + z_1 + z_2 + b - z_2 + e - z_1) \leq k(|X'| - 1)$, using the induction hypothesis and the relations $s_1+s_2 = k-(b+c+d+e)$, $z_1 \geq a+b+c+e-k$ and $z_2 \geq b+d+e+f-k$. If $x \in X'$ and $y \notin X'$ we put $X = X' \setminus \{x\} \cup \{w, v_1, v_2\}$, and if $x \notin X'$ and $y \in X'$ we put $X = X' \setminus \{y\} \cup \{w\}$. In either of these two cases the counting is straightforward. Finally, we assume that $x, y \notin X'$. If $z_1 = z_2 = 0$, then $G[X'] = G'[X']$, and so the conclusion holds. If $z_1 > 0$ and $z_2 > 0$, then let $X = X' \setminus \{u_1, u_2\}$. We have $2|E(G'[X'])| \leq 2|E(G[X])| + 2(k-(a+b)+k-(e+f)+z_1+z_2) \leq k(|X|-1)+2(b+c+d+e) \leq k(|X'| - 1)$, where the second inequality follows from the induction hypothesis (or is trivial if $|X| = 1$) and the definition of z_1 and z_2. Finally, from the symmetry between z_1 and z_2 it suffices to consider the case $z_1 = 0$ and $z_2 > 0$. In that case we put $X = X' \cup \{w, v_2\}$. Then $2|E(G'[X'])| = 2|E(G[X])| + 2(z_1 + z_2 - (b + d + e + f)) \leq k(|X'| - 1)$, using the induction hypothesis and the definition of z_1 and z_2. This completes the proof that $k \geq \Gamma(G')$.

By induction there exists a k-edge-coloring κ' of G'. Let $Z_1 \cup Z_2$ be the colors used on the $z_1 + z_2$ edges of $E(G') - E(G)$ with ends u_1 and u_2, so that $|Z_1| = z_1$ and $|Z_2| = z_2$. Let G'' be the graph obtained from G by deleting all edges with one end w and the other end v_1 or v_2. We first construct a suitable k-edge-coloring κ'' of G''. To do so we start with the restriction of κ' to $E(G'') \cap E(G')$, and then use Z_1 and the colors of the xu_1 edges of G' to color a subset of the u_1v_1 edges of G, we use Z_2 and the colors of the yu_1 edges of G' to color all of the wu_1 edges of G, and symmetrically we use Z_1 and the colors of the u_2y edges of G' to color all the wu_2 edges of G, and we use Z_2 and all the colors of the xu_2 edges of G' to color a subset of the v_2u_2 edges of G. We color the s_1 uncolored u_1v_1 edges and the s_2 uncolored u_2v_2 edges arbitrarily. That can be done, because u_i is the only neighbor of v_i in G''. This completes the definition of κ''. Now the number of colors seen by v_1 or w is at most $a - z_1 - s_1 + z_1 + z_2 + b - z_2 + e - z_1 + s_1 = a + b + e - z_1 \leq k - c$, and similarly the number of colors seen by v_2 or w is at most $k - d$. The number of colors seen by w, or by both v_1 and v_2 is at most $b - z_2 + e - z_1 + z_1 + z_2 + s \leq k - (c + d)$. By Lemma 2.1 the k-edge-coloring κ'' can be extended to a k-edge-coloring of G, as desired.

4 A linear-time algorithm

In this section we present a linear-time algorithm to decide whether $\chi'(G) \leq k$, where the series-parallel graph G and the integer k are part of the input instance. The idea of the algorithm is very simple – we repeatedly find vertices of the underlying simple graph satisfying one of (a)–(d) of Lemma 2.3, construct the graph G' as in the proof of Theorem 1.2, apply the algorithm recursively to G' to check whether $\chi'(G') \leq k$, and from that knowledge we deduce whether $\chi'(G) \leq k$. The construction of G' is straightforward, and the decision whether $\chi'(G) \leq k$ is easy: suppose, for instance, that we find vertices v_1, v_2, u_1, u_2, w as in Lemma 2.3(d), and let a, b, c, d, e, f be as in the proof of Theorem 1.2. If $a + b + c + d + e + f \geq k$, then construct G' as in the proof; we have $\chi'(G) \leq k$ if and only if $\chi'(G') \leq k$ and $a+b+c \leq k$ and $d+e+f \leq k$. If $a+b+c+d+e+f \leq k$, then $\chi'(G) \leq k$ if and only if $\chi'(G \backslash w) \leq k$. Thus it remains to describe how to find the vertices as in Lemma 2.3. That can be done by a slight modification of a linear-time recognition algorithm for series-parallel graphs. We need a few definitions in order to describe the algorithm.

Let H be a graph, and let λ be a function assigning to each edge $e \in E(H)$ a set $\lambda(e)$ disjoint from $V(H)$ in such a way that $\lambda(e) \cap \lambda(e') = \emptyset$ for distinct edges $e, e' \in E(H)$. Let H_λ be the graph obtained from H by adding, for each edge $e \in E(H)$ and each $x \in \lambda(e)$, a vertex x of degree two, adjacent to the two ends of e. Then H_λ is unique up to isomorphism, and so we can speak of the graph H_λ. Now let $\mu : E(H_\lambda) \to \mathbb{Z}_0^+$ be a function, and let H_λ^μ be the graph obtained from H_λ by replacing each edge $e \in E(H_\lambda)$ by $\mu(e)$ parallel edges with the same ends. In those circumstances we say that (H, λ, μ) is an *encoding*, and that it is an *encoding* of H_λ^μ.

For a graph H and $v \in V(H)$ we let $\deg_H(v)$ denote the number of edges incident to v in H and $\mathrm{val}_H(v)$ denote the number of distinct neighbors of v in H. Thus $\mathrm{val}_H(v) \leq \deg_H(v)$ with equality if and only if v is incident with no parallel edges. We say that a function $C : V(H) \to \mathbb{Z}_0^+$ is a *counter* for a graph H if $\deg_H(v) - \mathrm{val}_H(v) \leq C(v)$ for every vertex $v \in V(H)$. We say that a vertex $v \in V(H)$ is *active* if either $\deg_H(v) \leq 2$ or $\deg_H(v) \leq 3C(v)$.

The following lemma guarantees that if there are no active vertices, then the graph is null.

Lemma 4.1 *Let H be a non-null series-parallel graph, and let C be a counter for H. Then there exists an active vertex.*

Proof. As noted in the proof of Lemma 2.3, the underlying simple graph of H has a vertex of degree at most two. Thus H has a vertex v with $\mathrm{val}_H(v) \leq 2$. If $\deg_H(v) > 3C(v)$, then

$$\deg(v) - 2 \leq \deg_H(v) - \mathrm{val}_H(v) \leq C(v) < \deg_H(v)/3,$$

which implies $\deg_H(v) \leq 2$. Thus v is active, as desired. ■

4.1 The algorithm

The input for the algorithm is a series-parallel graph G and a non-negative integer k, where the graph G is presented by means of its underlying undirected graph and a function $E(G) \to \mathbb{Z}^+$ that describes the multiplicity of each edge.

The algorithm starts by checking whether $\deg_G(v) \leq k$ for all $v \in V(G)$. If not, it outputs "no, $\chi'(G) \not\leq k$" and terminates. Otherwise let H be the underlying undirected graph of G, let $\lambda(e) := \emptyset$ for every edge $e \in E(H)$, let $\mu(e)$ be the multiplicity of e in G, and let $C(v) := 0$ for every $v \in V(H)$. Then (H, λ, μ) is an encoding of G and C is a counter for H. The algorithm computes the list of all active vertices of H. It does not matter how L is implemented as long as elements can be deleted and added in constant time.

After this, the algorithm is iterative. Each iteration starts with an encoding (H, λ, μ) of the current series-parallel graph G, a counter C for H and a list L which includes all active vertices of H.

Each iteration consists of the following. If $L = \emptyset$, then we output "yes, $\chi'(G) \leq k$" and terminate, else we let v be a vertex in L. If $v \notin V(H)$ or v is not active, then we remove v from L and move to the next iteration. If $v \in V(H)$ and v is active, then there are three possible cases.

If $\deg_H(v) > 2$, then $\deg_H(v) \leq 3C(v)$, because v is active. We rearrange the adjacency list of v, removing all but one edge from each class of parallel edges incident with v, adjusting λ and μ so that (H, λ, μ) is still an encoding of G. We set $C(v) := 0$, include in L all vertices whose degree decreased and move to the next iteration.

If $\deg_H(v) = \mathrm{val}_H(v) = 2$ and $\lambda(vx) = \lambda(vy) = 0$, where x and y are the two distinct neighbors of v, then we remove v from H and add a new edge $f = xy$ to H. We set $\mu(f) := 0, \lambda(f) := \{v\}$, increase both $C(x)$ and $C(y)$ by one, add x and y to L and move to the next iteration.

If $\deg_H(v) \leq 2$ but the previous case does not apply, then we have located vertices of G satisfying one of (a) to (d) of Lemma 2.3. We check if the local conditions are satisfied or not (for example, in case (d), if $a+b+c+d+e+f \geq k$, we check whether $a+b+c \leq k$ and $d+e+f \leq k$); if they are not, we output "no, $\chi'(G) \not\leq k$" and terminate. Otherwise, we modify the encoding (H, λ, μ) to get an encoding of the graph G' described in the proof of Theorem 1.2. This involves deleting vertices from H and adding edges to H. Every time an edge of H incident with a vertex $z \in V(H)$ is deleted or added we increase $C(z)$ by one and add z to L. We move to the next iteration.

The correctness of the algorithm follows from Lemma 4.1 and from the proof of Theorem 1.2.

To analyze the running-time, let n denote the number of vertices of the input graph G. The initial steps of the algorithm can be done in $O(n)$ time. Each iteration takes time proportional to the decrease in the quantity

$$2K \cdot |V(H)| + K \cdot \sum_{e \in E(H)} \lambda(e) + |L| + 4 \cdot \sum_{v \in V(H)} C(v),$$

where K is a sufficiently large constant. Thus the running-time of the algorithm is $O(n)$.

References

[1] B. Bollobás, A. J. Harris, *List colorings of graphs*, Graphs and Combinatorics 1 (1985), 115–127.

[2] J. Duffin, *Topology of series-parallel networks*, Journal of Mathematical Analysis and Applications 10 (1965) 303–318.

[3] T. R. Jensen, B. Toft, *Graph Coloring Problems*, Wiley, New York, 1995.

[4] M. Juvan, B. Mohar and R. Thomas, *List Edge-Colorings of Series-Parallel Graphs*, Electronic Journal of Combinatorics 6 (1999), no. 1, Research Paper 42.

[5] P.D. Seymour, *Colouring Series-Parallel Graphs*, Combinatorica **10** (4) (1990) 379–392.

Multilength Single Pair Shortest Disjoint Paths [*]

Cristina G. Fernandes[1] [†] Hein van der Holst[2] [‡] José Coelho de Pina[1]

[1] Instituto de Matemática e Estatística
Universidade de São Paulo - Brazil
Rua do Matão 1010, 05508-090 São Paulo/SP, Brazil
E-mail: `{cris,coelho}@ime.usp.br`

[2] Fachbereich Mathematik und Informatik,
Freie Universität Berlin,
Arnimallee 2–6, D-14195 Berlin, Germany
E-mail: `hvdholst@math.fu-berlin.de`

Topics: algorithms and computational complexity.

Abstract

The k-SHORTEST PATHS problem consists of: given a digraph D, a pair (s,t) of vertices of D and k non-negative functions $l_1,\ldots,l_k$ on the arcs of D, find k internally vertex-disjoint paths $P_1,\ldots,P_k$ from s to t such that $l_1(P_1)+\cdots+l_k(P_k)$ is as small as possible. We describe, for each fixed k, a polynomial-time algorithm for the k-SHORTEST PATHS restricted to acyclic digraphs. We prove two complexity results: unless P = NP, for each constant c, there is no polynomial-time n^c-approximation algorithm (1) for the 2-SHORTEST PATHS, where n is the number of vertices of D, and (2) for the k-SHORTEST PATHS restricted to acyclic digraphs. We also show a polynomial-time algorithm for a multicommodity variation of the problem in planar graphs.

[*]Research supported in part by PRONEX/CNPq 664107/1997-4 (Brazil).
[†]Research supported in part by CNPq Proc. 301174/97-0 (Brazil).
[‡]Research supported in part by CAPES-PROAP (Brazil).

1 Introduction

The well-known single pair shortest path problem consists of: given a digraph D, a non-negative function l on the arcs of D and two vertices s and t, find a path P from s to t that minimizes $l(P)$, where $l(P)$ denotes the sum of $l(e)$ over all arcs e in P. This problem is solvable in polynomial time. We address the following generalization of the single pair shortest path problem, which we call k-SHORTEST PATHS:

given : – a digraph $D = (V, A)$;
– a pair (s, t) of vertices of D;
– non-negative functions $l_1, \ldots, l_k$ on the arcs of D;
find : – k internally vertex-disjoint paths $P_1, \ldots, P_k$ from s to t such that

$$l_1(P_1) + \cdots + l_k(P_k)$$

is as small as possible.

For $l_1 = \ldots = l_k$ the k-SHORTEST PATHS reduces to the min-cost flow problem and, therefore, can be solved in polynomial time.

We consider first the problem on acyclic digraphs. Algorithms for finding arc-disjoint paths in acyclic digraphs have applications on scheduling problems [1] and aircraft assignment problems [7]. We reformulate the k-SHORTEST PATHS in acyclic digraphs in terms of finding a shortest path in a (large) acyclic digraph. This is a known reformulation due to Perl and Shiloach [6] for finding two vertex-disjoint paths in an acyclic digraph. Later this was extended by Fortune, Hopcroft and Wyllie [3] in order to derive a polynomial-time algorithm for the k vertex-disjoint paths problem in acyclic digraphs (see also Schrijver [7]). From this reformulation, we derive the theorem below.

Theorem 1.1 *For each fixed k, there exists a polynomial-time algorithm for the k-*SHORTEST PATHS *restricted to acyclic digraphs.*

We also prove the following inapproximability result, which shows that the problem becomes much harder on general digraphs, even if $k = 2$.

Theorem 1.2 *For each constant c, there is no polynomial-time n^c-approximation algorithm for the* 2-SHORTEST PATHS *unless* P = NP, *where n is the number of vertices of the given digraph.*

With respect to the intractability and inapproximabity of the problem in acyclic digraphs, we show the following theorem.

Theorem 1.3 *For each constant c, there is no polynomial-time n^c-approximation algorithm for the k-*SHORTEST PATHS *restricted to acyclic digraphs unless* P = NP, *where n is the number of vertices of the given digraph.*

Theorems 1.2 and 1.3 show that the result in Theorem 1.1 is tight in the sense that it does not hold, unless P = NP, if we drop either the restriction on k being fixed or on D being acyclic. Surprisingly, the problem becomes much harder if we drop any of these restrictions.

We consider also a variant of the problem in undirected graphs, with multiple pairs of terminals. For this variant, we present a polynomial-time algorithm for the case where all length functions are the same, the given graph is planar and the terminals lie on the boundary of the same face in an adequate order.

2 Disjoint paths in acyclic digraphs

In order to prove Theorem 1.1, we consider the following disjoint paths problem:

given : – a directed graph $D = (V, A)$;
– pairs $(s_1, t_1), \ldots, (s_k, t_k)$ of vertices of D;
– subsets $A_1, \ldots, A_k$ of A;
– a set H of pairs $\{i, j\}$ from $\{1, \ldots, k\}$; (1)
find : – paths $P_1, \ldots, P_k$ in D such that:
(*i*) P_i is an s_i-t_i-path in $D[A_i]$ $(i = 1, \ldots, k)$;
(*ii*) P_i and P_j are vertex-disjoint for $\{i, j\}$ in H.

Fortune, Hopcroft and Wyllie [3] showed that this disjoint paths problem is NP-hard even for $k = 2$, $A_1 = A_2 = A$ and $H = \{\{1, 2\}\}$. According to Even, Itai and Shamir [2], problem (1) is also NP-hard for acyclic digraphs. In fact, problem (1) is NP-hard even for a fixed acyclic digraph, as noted by Alexander Schrijver. At the end of this section, we include the proof of this unpublished and surprising result.

We prove in the next theorem that problem (1) is polynomially solvable for instances satisfying the following condition:

There exists no directed cycle $C = P_{j_0} \cdot P_{j_1} \cdot \cdots \cdot P_{j_t}$ *in* D *such that:*
(*i*) P_{j_i} *is a path from* u_i *to* u_{i+1} *in* $D[A_{j_i}]$, $u_i \neq t_{j_i}$ $(i = 0, \ldots, t)$, *where* $u_{t+1} = u_0$; (2)
(*ii*) $\{j_0, j_1\}, \ldots, \{j_{t-1}, j_t\}, \{j_t, j_0\}$ *belong to* H.

If P and Q are paths then $P \cdot Q$ denotes the path obtained by the concatenation of P and Q. Note that any acyclic digraph satisfies the condition above. This theorem is a slight generalization of a result by Fortune, Hopcroft and Wyllie [3]. They showed that, for each fixed k, the problem of finding k vertex-disjoint paths in an acyclic digraph is polynomially solvable.

Theorem 2.1 *For each fixed k, there exists a polynomial-time algorithm for the disjoint paths problem (1) for instances satisfying (2).*

Proof. The proof is a minor modification of Schrijver's proof [7] of Fortune, Hopcroft and Wyllie's k vertex-disjoint paths theorem [3, 8]. We include it here for the sake of completeness.

Consider an instance of problem (1), that is, a digraph D, pairs $(s_1, t_1), \ldots, (s_k, t_k)$ of vertices of D, subsets $A_1, \ldots, A_k$ of arcs of D and a set H of pairs $\{i, j\}$ from $\{1, \ldots, k\}$. Make an auxiliary digraph $D' = (V', A')$ as follows. The vertex set V' consists of all k-tuples $(v_1, \ldots, v_k)$ of vertices of D such that $v_i \neq v_j$ for all $\{i, j\}$ in H. There is an arc in D' from $(v_1, \ldots, v_k)$ to $(w_1, \ldots, w_k)$ if and only if there exists an i in $\{1, \ldots, k\}$ such that:

(*i*) $v_j = w_j$ for all $j \neq i$;
(*ii*) (v_i, w_i) is an arc of A_i; (3)
(*iii*) if $j \neq i$, $\{i, j\} \in H$ and $v_j \neq t_j$, there is no path in $D[A_j]$ from v_j to v_i.

Note that, as k is fixed, the size of D' is polynomially bounded on the size of D. Moreover, the following holds:

D contains paths $P_1, \ldots, P_k$ such that P_i is an s_i-t_i-path in $D[A_i]$ $(i = 1, \ldots, k)$
and P_i and P_j are vertex-disjoint for $\{i, j\}$ in H
if and only if (4)
D' contains a path P from $(s_1, \ldots, s_k)$ to $(t_1, \ldots, t_k)$.

Suppose that $P_1, \ldots, P_k$ exist. For any i, let P_i follow the vertices $v_{i,0}, v_{i,1}, \ldots, v_{i,t_i}$. So $v_{i,0} = s_i$ and $v_{i,t_i} = t_i$ for each i. Choose $j_1, \ldots, j_k$ such that $0 \leq j_i \leq t_i$ for each i and such that:

(*i*) D' contains a path from $(s_1, \ldots, s_k)$ to $(v_{1,j_1}, \ldots, v_{k,j_k})$, and

(*ii*) $j_1 + \cdots + j_k$ is as large as possible.

Let $I := \{i \mid j_i < t_i\}$. Let us prove by contradiction that $I = \emptyset$. Suppose $I \neq \emptyset$. By the definition of D' and the maximality of $j_1 + \cdots + j_k$, for each i in I, there exists an $i' \neq i$ such that there is a path in $D[A_{i'}]$ from $v_{i',j_{i'}}$ to v_{i,j_i}, with $v_{i',j_{i'}} \neq t_{i'}$ and $\{i', i\}$ in H. So, for i in I, each vertex v_{i,j_i} is an endpoint of a path in $D[A_{i'}]$ starting at another vertex $v_{i',j_{i'}} \neq t_{j_{i'}}$, with i' in I and $\{i, i'\}$ in H. This contradicts (2), so $I = \emptyset$, that is, $j_i = t_i$ for all i, in which case we are done.

Conversely, let P be a path from $(s_1, \ldots, s_k)$ to $(t_1, \ldots, t_k)$ in D'. Let P follow the vertices $(v_{1,j}, \ldots, v_{k,j})$ for $j = 0, \ldots, t$. So $v_{i,0} = s_i$ for $i = 1, \ldots, k$. For each $i = 1, \ldots, k$, let P_i be the path in D following $v_{i,j}$ for $j = 0, \ldots, t$, taking repeated vertices only once. So P_i is an s_i-t_i-path in $D[A_i]$. Moreover, P_i and P_j are vertex-disjoint for each $\{i, j\}$ in H. Indeed, suppose P_1 and P_2 (say) have a vertex in common, where $\{1, 2\}$ belongs to H, that is, $v_{1,j} = v_{2,j'}$ for some $j \neq j'$. Without loss of generality, $j < j'$ and $v_{1,j} \neq v_{1,j+1}$. By the definition of D', there is no path in $D[A_2]$ from $v_{2,j}$ to $v_{1,j}$. This however contradicts the fact that $v_{1,j} = v_{2,j'}$ and that there exists a path in $D[A_2]$ from $v_{2,j}$ to $v_{2,j'}$.

Therefore, to solve problem (1), it is enough to find a path in D' from $(s_1, \ldots, s_k)$ to $(t_1, \ldots, t_k)$, which can be done in polynomial time. ■

The arc-disjoint version of the disjoint paths problem (1) consists of replacing (*ii*) in (1) by:

$$P_i \text{ and } P_j \text{ are arc-disjoint for } \{i, j\} \text{ in } H. \tag{5}$$

This arc-disjoint paths problem can be reformulated in terms of the disjoint paths problem (1). Indeed, let an instance of the arc-disjoint paths problem be given, that is, a digraph $D = (V, A)$, pairs of vertices $(s_1, t_1), \ldots, (s_k, t_k)$, arc sets $A_1, \ldots, A_k$ and a set H of pairs $\{i, j\}$ from $\{1, \ldots, k\}$. We may assume that each s_i is the tail of a unique arc a_i of D and that t_i is the head of a unique arc b_i of D ($i = 1, \ldots, k$). We make a digraph $D' = (V', A')$ as follows. The vertex set of D' is the arc set A of D (i.e. $V' := A$). There is an arc in D' from a to b if the head of a and the tail of b coincide. For $i = 1, \ldots, k$, we define $A'_i := \{(a, b) \in A' \mid a, b \in A_i\}$. Finally we take $H' := H$.

Finding paths $P_1, \ldots, P_k$ in D satisfying (5) such that P_i is an s_i-t_i-path in $D[A_i]$ ($i = 1, \ldots, k$) is equivalent to the problem of finding paths $P'_1, \ldots, P'_k$ in D' satisfying (*ii*) of (1) such that P'_i is an a_i-b_i-path in $D'[A'_i]$ ($i = 1, \ldots, k$). Hence, the arc-disjoint version of problem (1) is polynomially solvable for instances satisfying a condition similar to condition (2).

Now, suppose that, for an instance of problem (1), one is given also non-negative functions $l_1, \ldots, l_k$ on the arcs of D. Then it is possible to find in polynomial time a solution $P_1, \ldots, P_k$ of problem (1) such that $\sum_{i=1}^{k} l_i(P_i)$ is as small as possible. Just define a length function on the arcs of D' (the digraph from the proof of Theorem 2.1) as follows. The length of an arc of D' from $(v_1, \ldots, v_k)$ to $(w_1, \ldots, w_k)$ satisfying (3) is $l_i(v_i, w_i)$. Now, a shortest path from $(s_1, \ldots, s_k)$ to $(t_1, \ldots, t_k)$ in D' with this length function on its arcs gives the desired paths. As the shortest path problem in an acyclic digraph with arbitrary length on its arcs can be solved in linear time, we have Theorem 1.1.

Theorem 1.1 *For each fixed k, there exists a polynomial-time algorithm for the k-*SHORTEST PATHS *restricted to acyclic digraphs.* ■

We conclude this section with the proof of Schrijver's result on the complexity of problem (1).

Theorem 2.2 (A. Schrijver) *The disjoint paths problem (1) restricted to instances having the digraph in Figure 1 as input is* NP*-hard.* ■

Proof. Consider the following transformation of PLANAR 3-COLORABILITY to problem (1) restricted to instances having the acyclic digraph displayed in Figure 1 as input. A *k-coloring* of a graph $G = (V, E)$ is a function f from V to $\{1, \ldots, k\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\}$ belongs to E. Graph G is *k-colorable* if G has a k-coloring. The PLANAR 3-COLORABILITY problem consists of: *given:* a planar graph $G = (V, E)$; *question:* is G 3-colorable? PLANAR 3-COLORABILITY was shown to be NP-complete by Stockmeyer [9].

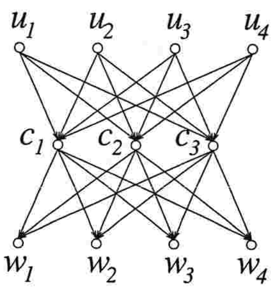


Figure 1: Problem (1) is NP-hard for instances having this acyclic digraph as input.

Let us be given a planar graph $G = (V, E)$ with $V = \{v_1, \ldots, v_k\}$ and let f' be a 4-coloring of G. This function f' can be computed in polynomial time (see, for instance, Nishizeki and Chiba [5]). We construct an instance of problem (1) depending on G and f' as follows. Let $H := \{\{i, j\} \mid \{v_i, v_j\} \in E\}$ and, for $i = 1, \ldots, k$, let $s_i := u_{f'(v_i)}$, $t_i := w_{f'(v_i)}$ and A_i be the set of all arcs of the digraph in Figure 1. We claim that G is 3-colorable if and only if the constructed instance of problem (1) is feasible. Suppose G is 3-colorable and let f be a 3-coloring of G. For $i = 1, \ldots, k$, let P_i be the path from s_i to t_i that traverses $c_{f(v_i)}$. One can check that $P_1, \ldots, P_k$ is a solution to the problem (1). Conversely, let $P_1, \ldots, P_k$ be a solution to the disjoint paths problem (1) and define f from V to $\{1, 2, 3\}$ such that, for $i = 1, \ldots, k$, the path P_i from s_i to t_i traverses the vertex $c_{f(v_i)}$. One can verify that f is a 3-coloring of G. ■

3 Inapproximability for the 2-SHORTEST PATHS

In this section we analyze the complexity of the 2-SHORTEST PATHS problem. Specifically, we prove Theorem 1.2.

Theorem 1.2 *For each constant c, there is no polynomial-time n^c-approximation algorithm for the 2-SHORTEST PATHS unless P = NP, where n is the number of vertices of the given digraph.*

Proof. We may assume $c \geq 1$. Suppose that there is a polynomial-time n^c-approximation algorithm A for the 2-SHORTEST PATHS, where n is the number of vertices of the given digraph. Let us show that, if this is the case, we can solve 3-SAT in polynomial time, which implies that P = NP. For this, consider the following polynomial-time reduction from 3-SAT to 2-SHORTEST PATHS.

Let Φ be an instance of 3-SAT, that is, a set $\{C_1, \ldots, C_m\}$ of 3-clauses on variables $x_1, \ldots, x_h$. Let us describe a digraph D, two length functions l_1 and l_2 on the arcs of D and two vertices s and t.

For each variable x_i, denote by d_i the largest between the number of times x_i appears in Φ and the number of times that $\overline{x}_i$ appears in Φ. There is a gadget as in Figure 2(a) for each x_i. The number of undirected four-cycles in the gadget is $d_i + 1$. The source vertex in the gadget is called v_i and the sink vertex, w_i. The vertices of in-degree one in the gadget are partitioned into two sets: L_i and R_i, as in Figure 2(a).

For each clause C_j, there is a gadget as in Figure 2(b). The sink and source vertices are called u_j and l_j respectively. Each of the other vertices has as label one of the literals in clause C_j.

The digraph of the instance of 2-SHORTEST PATHS is obtained as follows. First, we connect the gadgets of all variables and clauses in series, identifying w_i and v_{i+1} ($i = 1, \ldots, n-1$) and u_j and l_{j+1} ($j = 1, \ldots, m-1$). Then, we add an arc from s to v_1, one from w_h to l_1 and one from u_m to t. The arcs we have up to now

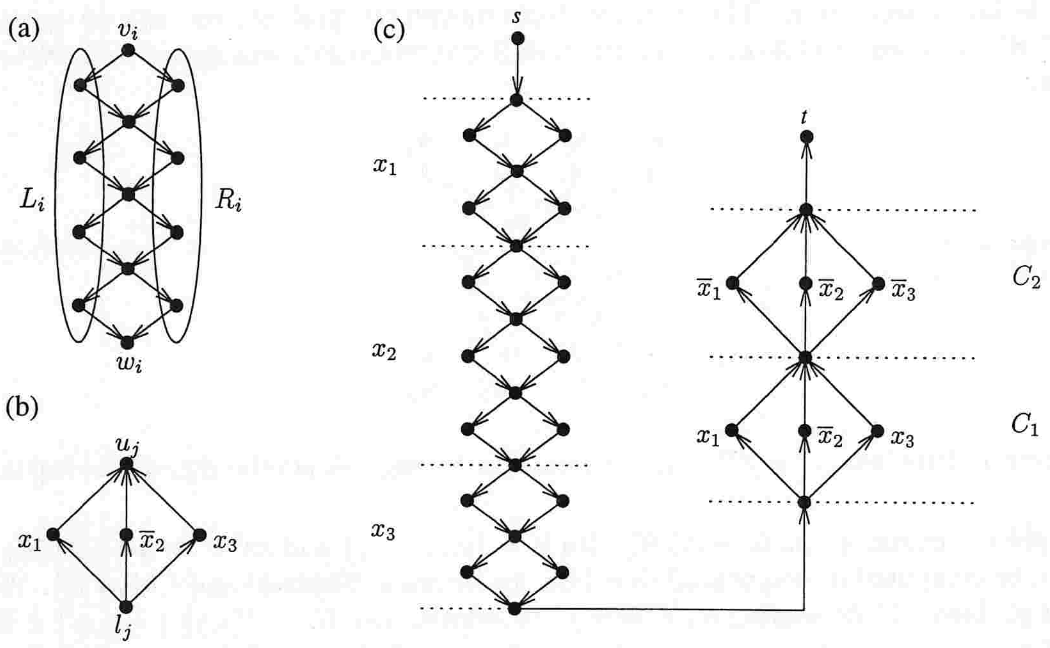


Figure 2: (a) The gadget for variable x_i. One, between x_i or $\overline{x}_i$, appears three times in Φ, while the other appears at most three times. (b) The gadget for clause $C_j = \{x_1, \overline{x}_2, x_3\}$. (c) Arcs of type 1 of the digraph built from $\Phi = (x_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$.

are said to be of type 1. See Figure 2(c). Second, we add three arcs from s: one to t, one to the first vertex in L_1 and another to the first vertex in R_1. Similarly, we add two arcs to t: one from the last vertex in L_h and one from the last vertex in R_h. For each two consecutive vertices in L_i, we add a path from the upper one to the lower one, of length one or two. When the path has length two, the middle vertex is one of the vertices labeled x_i in the clause gadgets. The same holds for R_i with $\overline{x}_i$ in the place of x_i. This is done in such a way that any labeled vertex is in exactly one of these two-length paths. Finally, there are also arcs from the last vertex in L_i and from the last vertex in R_i to both, the first vertex in L_{i+1} and the first vertex in R_{i+1} ($i = 1, \ldots, n-1$). The arcs added in this second phase are said to be of type 2. This finishes the description of the digraph D and vertices s and t. See Figure 3(a) for a complete example. Note that the number of vertices in this digraph is at most $4 + 3d + 3h + 4m$, where $d := \sum_{i=1}^{h} d_i \leq 3m$. Also, there are two internally disjoint paths from s to t in D.

To complete the description of the instance of 2-SHORTEST PATHS, it is missing only to describe the two length functions l_1 and l_2 on the arcs of D. In l_1, arcs of type 1 have length one, while arcs of type 2 have length $M := (4 + 3d + 3h + 4m)^{c+1} + 1$. In l_2, all arcs have length one, but arc st, whose length is M.

Note that the construction of D, s, t, l_1 and l_2 takes polynomial time on the size of Φ.

Claim 3.1 *Φ is satisfiable if and only if there are two internally disjoint paths P_1 and P_2 from s to t in D such that $l_1(P_1) + l_2(P_2) \leq 4 + 3d + 3h + 4m$.*

Proof. Assume Φ is satisfiable and consider an assignment which satisfies Φ. Let us describe two internally disjoint paths P_1 and P_2 in D from s to t such that $l_1(P_1) + l_2(P_2) \leq 4 + 3d + 3h + 4m$.

Path P_1 starts with arc sv_1, goes from v_1 to w_h using only arcs in the variable gadgets, then uses arc $w_h l_1$ and goes from l_1 to u_m using only arcs in the clause gadgets. It ends with arc $u_m t$. Inside the variable gadget for x_i, path P_1 goes through all vertices in L_i if x_i is TRUE in the assignment or all vertices in R_i if x_i is FALSE. In the clause gadgets, P_1 goes always through a vertex whose label is a TRUE literal in the assignment. Note that P_1 uses only type 1 arcs. Thus $l_1(P_1) = 3 + 2d + 2h + 2m$.

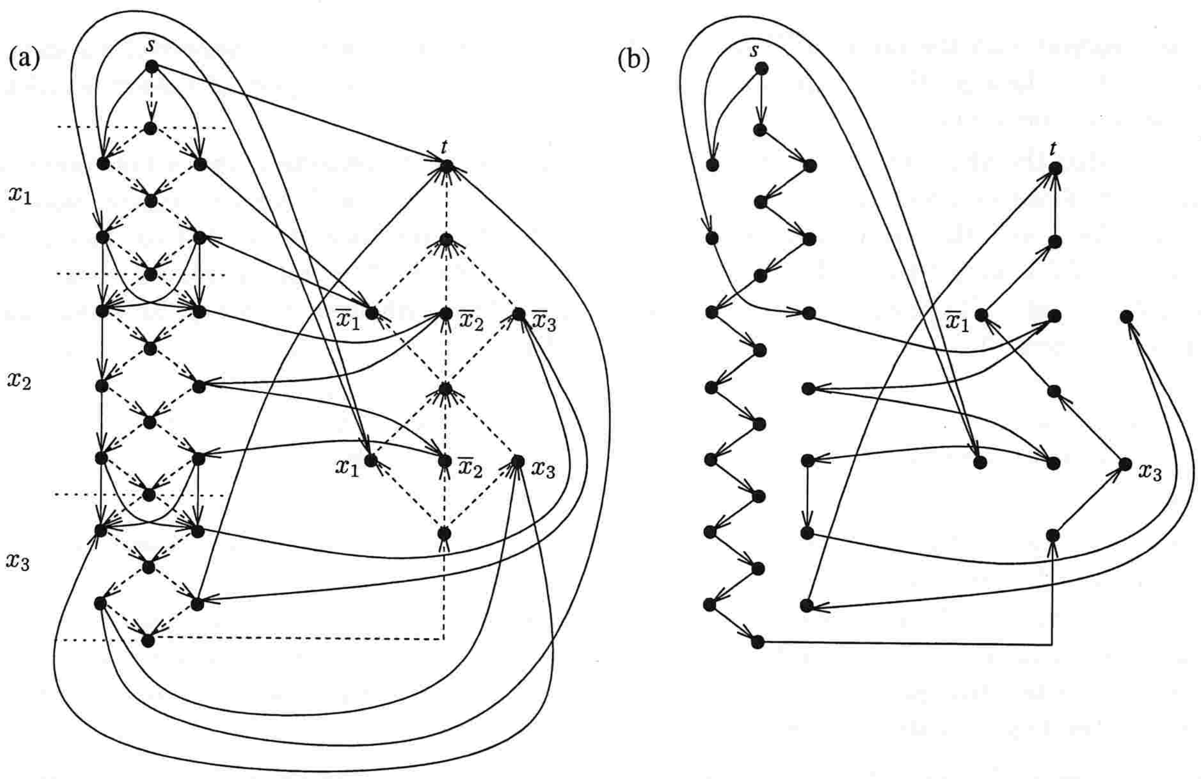


Figure 3: (a) Digraph built from $\Phi = (x_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$. The dashed arcs have l_1 equals one, while the others have l_1 equals M. (b) Paths P_1 and P_2 corresponding to the assignment $x_1 = F$, $x_2 = T$ and $x_3 = T$.

Path P_2 uses only type 2 arcs. If x_1 is TRUE (FALSE), it goes from s to the first vertex in L_1 (R_1). From there, it traverses all vertices in L_1 (R_1) and jumps to L_2 if x_2 is TRUE or to R_2 if x_2 is FALSE and proceeds in the same way until it gets to the last vertex in R_h or L_h. In this traversal, it goes back and forth to the clause gadgets through some length-two paths, always using a vertex whose label is a literal set to FALSE. From the last vertex in L_h or R_h, it goes directly to t. Note that P_2 is indeed internally disjoint from P_1, as it uses only type 2 arcs. Moreover, $l_2(P_2) = 1 + d + h + 2m$.

Therefore $l_1(P_1) + l_2(P_2) = 4 + 3d + 3h + 4m$, as desired. See in Figure 3(b) how P_1 and P_2 look like for the example given in Figure 3(a).

Now assume there are two internally disjoint paths P_1 and P_2 in D from s to t such that $l_1(P_1)+l_2(P_2) \leq 4 + 3d + 3h + 4m$. Note that P_1 can only use type 1 arcs, otherwise $l_1(P_1) \geq M > 4 + 3d + 3h + 4m$ (the last inequality holds as $c \geq 1$). Also, P_2 does not use arc st, as $l_2(st) = M > 4 + 3d + 3h + 4m$. As P_1 uses only type 1 arcs, P_1 uses sv_1, then it goes from v_1 to w_h using only arcs in the variable gadgets, then it uses $w_h l_1$ and goes from l_1 to u_m inside the clause gadgets, finishing with $u_m t$. Path P_1 cannot use vertices both in L_i and R_i, otherwise the only path from s to t in D internally disjoint from P_1 consists of st. But $l_2(st) = M > 4 + 3d + 3h + 4m$. Indeed, P_1 must pass by all vertices of the variable gadget x_i not in $L_i \cup R_i$ and by all unlabeled vertices in the clause gadgets. But then, if P_2 uses the first vertex in L_i, it has no other way except using all other vertices in L_i. The same holds for R_i.

Now we are ready to describe the assignment. Set x_i to TRUE if and only if P_2 uses vertices of L_i. Note that P_2 visits all labeled vertices in the clause gadgets whose labels were set to FALSE. But path P_1 necessarily uses a labeled vertex in each clause gadget. The label $\tilde{x}_i$ of this labeled vertex must then be TRUE, which means there is a TRUE literal in each clause. That is, Φ is satisfiable. ■

Now we proceed with the proof of Theorem 1.2. Run algorithm A on the constructed instance of 2-SHORTEST PATHS. The algorithm returns two paths, P_1 and P_2. If $l_1(P_1)+l_2(P_2) < M$ then Φ is satisfiable, otherwise Φ is not satisfiable.

First, note that the above algorithm runs in polynomial-time, as the reduction and A take polynomial-time. Moreover, it solves 3-SAT. Indeed, assume $l_1(P_1) + l_2(P_2) \geq M$. As A is an n^c-approximation and $n = 4 + 3d + 3h + 4m$, the value of an optimal solution for this instance of the 2-SHORTEST PATHS is at least $M/(4 + 3d + 3h + 4m)^c > 4 + 3d + 3h + 4m$. By the claim, Φ is not satisfiable. Now, assume $l_1(P_1) + l_2(P_2) < M$. This means P_1 uses no type 2 arc. Any path from s to t in D which uses no type 2 arc uses exactly $3 + 2d + 2h + 2m$ arcs of type 1, that is, $l_1(P_1) = 3 + 2d + 2h + 2m$. But then, as $n = 4 + 3d + 3h + 4m$, path P_2 uses at most $2 + d + h + 2m$ vertices, that is, $l_2(P_2) \leq 2 + d + h + 2m$. Therefore, by the claim, Φ is satisfiable. As 3-SAT is solvable in polynomial time only if P = NP, there is no n^c-approximation algorithm for 2-SHORTEST PATHS unless P = NP. ■

In fact, it is easy to modify this theorem to show that, for any polynomial-time computable function f, there is no polynomial-time $f(|I|)$-approximation algorithm for 2-SHORTEST PATHS, where I denotes an arbitrary instance of 2-SHORTEST PATHS.

Consider the undirected edge/vertex-disjoint versions of the k-SHORTEST PATHS problem. There are well-known reductions from the undirected edge-disjoint version to the undirected vertex-disjoint and to the directed arc/vertex-disjoint versions of the problem. One can modify the proof of the previous theorem in order to get the following stronger theorem.

Theorem 3.2 *For each constant c, there is no polynomial-time n^c-approximation algorithm for the undirected edge-disjoint* 2-SHORTEST PATHS *unless* P = NP, *where n is the number of vertices of the given graph.*

4 Inapproximability for acyclic digraphs

In this section we show that if k is non-fixed then the k-SHORTEST PATHS is hard to approximate, even restricted to acyclic digraphs. The proof of the theorem below is a modification of the proof of Theorem 1.2.

Theorem 1.3 *For each constant c, there is no polynomial-time n^c-approximation algorithm for the k-*SHORTEST PATHS *restricted to acyclic digraphs unless* P = NP, *where n is the number of vertices of the given digraph.*

Proof. We may assume $c \geq 1$. Suppose that there is a polynomial-time n^c-approximation algorithm A for the k-SHORTEST PATHS on acyclic digraph, where n is the number of vertices of the given digraph. Consider the following polynomial-time reduction from 3-SAT to k-SHORTEST PATHS.

Let Φ be an instance of 3-SAT, that is, a set $\{C_1, \ldots, C_m\}$ of 3-clauses on variables $x_1, \ldots, x_h$. Let us describe an acyclic digraph D, two vertices s and t and length functions $l_0, \ldots, l_h$ on the arcs of D.

Digraph D consists of vertices $s, v_0, \ldots, v_h, t$, arcs st, sv_0, v_ht and, for each variable x_i, two internally disjoint paths, Q_i and $\bar{Q}_i$, from v_{i-1} to v_i. Path Q_i ($\bar{Q}_i$) has as many internal vertices as appearances of x_i ($\overline{x}_i$). Additionally, for each clause C_j, there are three length-two paths from s to t, each one having as middle vertex a vertex labeled by one of the literals in C_j. This is done in such a way that no two of these paths share the middle vertex. See Figure 4 for an example.

Let $M := (3 + 3m + h)^c(2 + 3m + 3h) + 1$. For each C_j, set $l_j(e) := 1$ if e is one of the arcs in the three length-two paths added for C_j and set $l_j(e) := M$ otherwise. Set $l_0(e) := 1$ if $e = sv_0$ or $e = v_ht$ or e is in path Q_i or $\bar{Q}_i$ for some i. Otherwise set $l_0(e) := M$. This completes the description of the instance of k-SHORTEST PATHS. Observe that D is acyclic and that there are k internally disjoint paths from s to t in D. Also, observe that D, s, t and $l_0, \ldots, l_h$ can be constructed in polynomial time.

Claim 4.1 *Φ is satisfiable if and only if there are internally disjoint paths $P_0, \ldots, P_h$ from s to t in D*

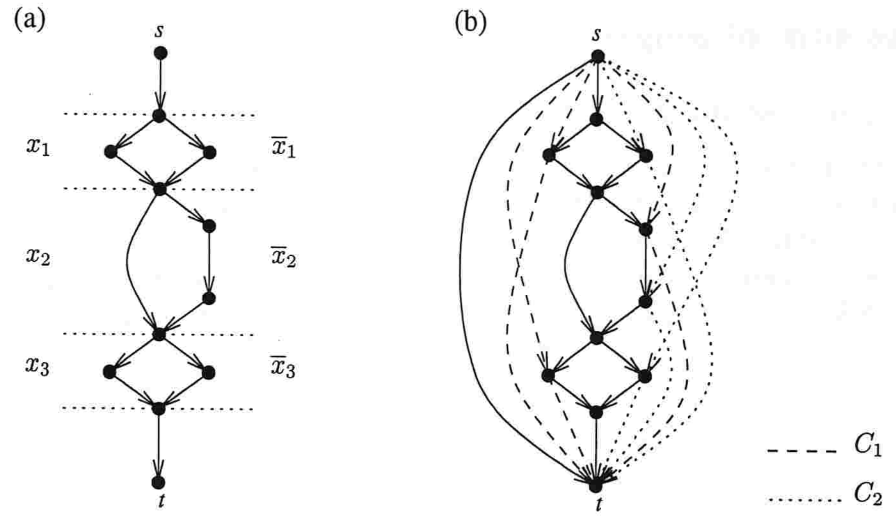


Figure 4: (a) Arcs for the variables. (b) Digraph for $\Phi = (x_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$.

such that $l_0(P_0) + \cdots + l_h(P_h) \leq 2 + 3m + 3h$.

Proof. Assume Φ is satisfiable and consider an assignment which satisfies Φ. Let P_0 be the path starting with arc sv_0, ending with arc $v_h t$, and using Q_i, if x_i is FALSE, or $\bar{Q}_i$, if x_i is TRUE, for each i. Note that each labeled vertex in P_0 has as label a literal that is FALSE in the assignment. Moreover, $l_0(P_0) \leq 2 + 3m + h$. For each clause C_j, let P_j be one among the three paths for C_j that use a vertex whose label is a literal set to TRUE in the assignment. There is one such path because the assignment satisfies Φ. Paths $P_0, \ldots, P_h$ are such that $l_0(P_0) + \cdots + l_h(P_h) \leq 2 + 3m + 3h$.

Suppose now that there are internally disjoint paths $P_0, \ldots, P_h$ from s to t in D such that $l_0(P_0) + \cdots + l_h(P_h) \leq 2 + 3m + 3h$. Note that $M > 2 + 3m + 3h$, because $c \geq 1$. Therefore, $P_0, \ldots, P_h$ use only arcs whose length is one in their respective length functions. In particular, P_0 uses necessarily arcs sv_0 and $v_h t$ and passes by vertices $v_1, \ldots, v_{h-1}$. To go from v_{i-1} to v_i, path P_0 uses either Q_i or $\bar{Q}_i$. Set variable x_i to TRUE if P_0 uses $\bar{Q}_i$ and set x_i to FALSE if P_0 uses path Q_i. For each C_j, path P_j has to be one of the length-two paths for C_j. Let $\tilde{x}_i$ be the label of the middle in P_j. If P_0 uses Q_i, then $\tilde{x}_i = \overline{x}_i$ (or P_j and P_0 would not be internally disjoint). If P_0 uses $\bar{Q}_i$, then $\tilde{x}_i = x_i$. In both cases, $\tilde{x}_i$ is TRUE in the assignment and therefore this assignment satisfies C_j, for all j. ■

To complete the proof of Theorem 1.3, it is enough to describe how to use algorithm A to get a polynomial-time algorithm for 3-SAT. Just run algorithm A on the constructed instance of k-SHORTEST PATHS. It returns paths $P_0, \ldots, P_h$. If $l_0(P_0) + \cdots + l_h(P_h) < M$, then Φ is satisfiable, otherwise Φ is not satisfiable. The resulting algorithm is clearly polynomial and solves 3-SAT. Indeed, assume that $l_0(P_0) + \cdots + l_h(P_h) \geq M$. Algorithm A is an n^c-approximation, where n is the number of vertices of D, that is, $n = 3 + 3m + h$. Therefore, the value of an optimal solution for this instance of the k-SHORTEST PATHS is at least $M/(3 + 3m + h)^c > 2 + 3m + 3h$. By the claim, Φ is not satisfiable. Now, assume $l_0(P_0) + \cdots + l_h(P_h) < M$. This means P_0 uses neither st nor arcs in the paths of the clauses. Therefore P_0 uses arcs sv_0 and $v_h t$ and, for each i, uses either Q_i or $\bar{Q}_i$. Thus $l_0(P_0) \leq 2 + 3m + h$. Also, each path P_j has to be one of the paths for clause C_j, otherwise $l_j(P_j) \geq M$. Hence $l_j(P_j) = 2$, for all j, and $l_0(P_0) + \cdots + l_h(P_h) \leq 2 + 3m + 3h$. By the claim, Φ is satisfiable. ■

This theorem also holds for any polynomial-time computable function f: there is no polynomial-time $f(|I|)$-approximation algorithm for k-SHORTEST PATHS in acyclic digraphs. Here, I denotes an arbitrary instance of k-SHORTEST PATHS in acyclic digraphs.

5 Minimizing sum of lengths

Consider the following shortest disjoint paths problem:

given : – an undirected planar graph $G = (V, E)$, embedded in $\mathbb{R}^2$;
– pairs $\{s_1, t_1\}, \dots, \{s_k, t_k\}$ of vertices on the boundary of G;
– a non-negative function l on the edges of G;

find : – pairwise vertex-disjoint paths $P_1, \dots, P_k$ in G where P_i is an s_i-t_i-path, for each $i = 1, \dots, k$, and $l(P_1) + \cdots + l(P_k)$ is as small as possible.

We denote this problem by $\textsc{sdp}(G, \{s_1, t_1\}, \dots, \{s_k, t_k\})$, or simply by SDP, when the instance is clear from the context.

If the vertices $s_1, \dots, s_k, t_k, \dots, t_1$ occur in this order when following the boundary of G, then SDP can be seen as a particular case of the min-cost flow problem. Indeed, from G, we construct a digraph D by splitting each vertex v of G into two vertices v^+ and v^-, joined by an arc (v^-, v^+) of cost zero. An edge vw of G becomes arcs (v^+, w^-) and (w^+, v^-) of D, both of cost $l(vw)$. In addition, D has vertices s, t and arcs (s, s_i^-), (t_i^+, t), for $i = 1, \dots, k$, of cost zero. Solving SDP is equivalent to finding a maximum s-t-flow of minimum cost in D, with each arc having capacity one. Hence, in this particular case, SDP can be solved in polynomial time.

A graph $G = (V \cup \{c\}, E)$ is called a *wheel* if $G - c$ is a circuit and $\{v, c\} \in E$ for each v in V. Grötschel, Martin and Weismantel [4] showed that if $G = (V \cup \{c\}, E)$ is a wheel and $s_1, t_1, \dots, s_k, t_k$ occur in this order when following the circuit $G - c$ then the edge-disjoint version of SDP can be solved in polynomial time. Moreover, Grötschel, Martin and Weismantel gave a complete description of the *path packing polytope* (the convex hull of incident vectors of sets $E' \subseteq E$, such that $G[E']$ is a packing of edge-disjoint s_i-t_i-paths in G).

Using dynamic programming, we show that, also in the following case, SDP can be solved in polynomial time:

$$\text{k is fixed and the vertices $s_1, t_1, \dots, s_k, t_k$ occur in this order when following the boundary of G.} \tag{6}$$

We assume that SDP is feasible (this can be tested in linear time) and that G is 2-connected.

We shall use the following notation. If P is a path and u, v are vertices in P then $P(u, v)$ is the subpath of P connecting u to v. We denote by Q_i a shortest s_i-t_i-path. Let R_i be the subgraph of G induced by the vertices in the closed region bounded by the path Q_i and the path on the boundary of G from s_i to t_i containing no other s_j or t_j ($i = 1, \dots, k$). Also, let $Q_{i,j} := R_i \cap R_j$. By shortcut arguments, we may assume that $Q_{i,j}$ is either a path or empty, for $i \neq j$. Figure 5 illustrates the notation. There, P_1, P_2, P_3 are paths of an optimal solution.

Let G, $\{s_1, t_1\}, \dots, \{s_k, t_k\}$ and l be an instance of SDP with the property that $s_1, t_1, \dots, s_k, t_k$ occur in this order when following the boundary of G. Observe that SDP has an optimal solution $P_1, \dots, P_k$ so that P_i is entirely contained in the region R_i ($i = 1, \dots, k$).

Consider some i and j with $i \neq j$. Let $Q_{i,j} := (v_0, e_1, v_1, \dots, e_d, v_d)$. We call (f, h) a *possible choice* (for (i, j)) if $f = h = \mathsf{nil}$ or $f = v_p$ and $h = v_q$ for some p, q satisfying $0 \leq p \leq q \leq d$. We say that (f, h, f', h') is a *feasible choice* (for (i, j)) if (f, h) and (f', h') are possible choices and if $f \neq \mathsf{nil} \neq f'$ implies $\{f, h\} \cap \{f', h'\} = \emptyset$. For each feasible choice (f, h, f', h'), let

$$\begin{aligned} G_{i,j,f,h,f',h'} \;&:=\; G[V(R_i) \setminus (V(Q_{i,j}) \setminus V(Q_{i,j}(f, h)))] \\ &\qquad \cup\, G[V(R_j) \setminus (V(Q_{i,j}) \setminus V(Q_{i,j}(f', h')))], \end{aligned}$$

where $Q_{i,j}(\mathsf{nil},\mathsf{nil}) := \emptyset$. Finally, we say that a sequence

$$((f_{i,j}, h_{i,j}, f'_{i,j}, h'_{i,j}) : 1 \leq i < j \leq k)$$

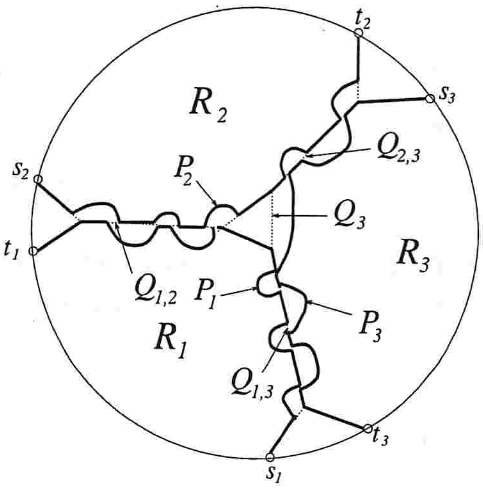


Figure 5: Illustration of the definitions.

is an *acceptable choice* provided that $(f_{i,j}, h_{i,j}, f'_{i,j}, h'_{i,j})$ is a feasible choice for each (i, j), for all $1 \leq i < j \leq k$.

Our dynamic programming approach is based on the following optimality criterion:

Let $P_1, \ldots, P_k$ be an optimal solution to SDP and let f_i, h_i, f_j, h_j be the first and last vertices of $Q_{i,j}$ in P_i and in P_j, respectively, for some i, j with $1 \leq i < j \leq k$. Then $P_i(f_i, h_i), P_j(f_j, h_j)$ is an optimal solution to $\text{SDP}(G_{i,j,f_i,h_i,f_j,h_j}, \{f_i, h_i\}, \{f_j, h_j\})$ (see Figure 6).

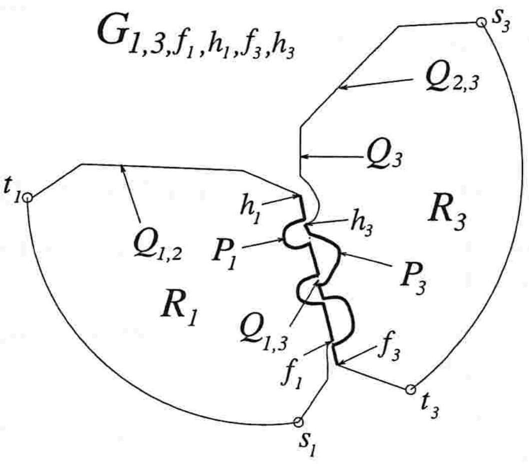


Figure 6: Illustration of the optimality condition for SDP.

The algorithm consists of first computing, for each (i, j), with $1 \leq i < j \leq k$, and for each feasible choice (f, h, f', h') for (i, j), a solution $P_{i,j,f,h,f',h'}$, $P_{j,i,f,h,f',h'}$ to $\text{SDP}(G_{i,j,f,h,f',h'}, \{f, h\}, \{f', h'\})$, where, $P_{i,j,\mathsf{nil},\mathsf{nil},f',h'} := \emptyset$ and $P_{j,i,f,h,\mathsf{nil},\mathsf{nil}} := \emptyset$). Now, we enumerate all acceptable choices

$$A \quad := \quad ((f_{i,j}, h_{i,j}, f'_{i,j}, h'_{i,j}) : i, j = 1, \ldots, k, i < j)$$

(so, $(f_{i,j}, h_{i,j}, f'_{i,j}, h'_{i,j})$ is a feasible choice for all (i, j) with $1 \leq i < j \leq k$) and we compute $P_1^A, \ldots, P_k^A$, where P_i^A is a shortest s_i-t_i-path in

$$G[(V(R_i) \setminus V(Q_{i,j})) \ \cup \ (\cup_{i \neq j} V(P_{i,j,f_{i,j},h_{i,j},f'_{i,j},h'_{i,j}}))] \ \ (i = 1, \ldots, k),$$

if there exists any.

The algorithm returns, for some acceptable choice A^*, vertex-disjoint paths $P_1^{A^*}, \dots, P_k^{A^*}$ so that,

$$\sum_{i=1}^k l(P_i^{A^*}) = \min\{\sum_{i=1}^k l(P_i^A) \mid A \text{ is an acceptable choice}\}.$$

Theorem 5.1 *If $s_1, t_1, \dots, s_k, t_k$ occur in this order when following the boundary of G then, for each fixed k,* SDP *can be solved in polynomial time.*

Proof. Any solution $P_1, \dots, P_k$ to SDP such that P_i is a subgraph of R_i induces an acceptable choice to SDP. (For each (i,j) with $i < j$, we define $(f_{i,j}, h_{i,j}, f'_{i,j}, h'_{i,j})$ as the first and last vertices of $Q_{i,j}$ in P_i and P_j, respectively. If the path $Q_{i,j}$ does not meet, say, P_i then $f_{i,j} := h_{i,j} := \mathsf{nil}$.) Since by shortcut arguments there exists at least one such a solution, it follows that the algorithm generates and returns a solution to SDP.

One sees that SDP$(G_{i,j,f,h,f',h'}, \{f,h\}, \{f',h'\})$ can be solved in polynomial time, as (if $f \neq \mathsf{nil} \neq f'$) f, h, f', h' are vertices on the boundary of $G_{i,j,f,h,f',h'}$. Thus, in order to show polynomiality, it remains to verify that the number of acceptable choices is polynomially bounded. Indeed,

$$|\{Q_{i,j} \mid i < j \text{ and } Q_{i,j} \neq \emptyset\}| = O(k^2)$$

and there exist $O(n^4)$ feasible choices for each (i,j). Hence, there are $O(n^{4k^2})$ acceptable choices. ■

Remark. Using similar techniques one can prove that if $s_1, t_1, \dots, s_k, t_k$ occur in this order when following the boundary of G then the edge-disjoint version of SDP is polynomially solvable for each fixed k. It can also be proved that if, in addition, $|i - j| > 1$ implies $Q_{i,j} = \emptyset$, then SDP can be solved in polynomial time (here k does not need to be fixed).

Acknowledgments

We thank Bruce Shepherd for bringing the k-SHORTEST PATHS problem to our attention. We also thank Alexander Schrijver and Bruce Shepherd for many valuable comments and suggestions.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network flows*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] S. Even, A. Itai, and A. Shamir, *On the complexity of timetable and multicommodity flow problems*, SIAM Journal on Computing **5** (1976), 691–703.

[3] S. Fortune, J. Hopcroft, and J. Wyllie, *The direct subgraph homeomorphism problem*, Theoretical Computer Science **10** (1980), 111–121.

[4] M. Grötschel, A. Martin, and R. Weismantel, *Optimum path packing on wheels: the consecutive case*, Comput. Math. Appl. **31** (1996), no. 11, 23–35.

[5] T. Nishizeki and N. Chiba, *Planar graphs: Theory and algorithms*, North-Holland, Amsterdam, 1988.

[6] Y. Perl and Y. Shiloach, *Finding two disjoint paths between two pairs of vertices in a graph*, Journal of the Association for Computing Machinery **25** (1978), 1–9.

[7] A. Schrijver, *A group-theoretical approach to disjoint paths in directed graphs*, CWI Quarterly **6** (1993), 257–266.

[8] ———, *Combinatorial optimization—polyhedra and efficiency*, Springer-Verlag, forthcoming book, Part VII. Multiflows and Disjoint Paths.

[9] L.J. Stockmeyer, *Planar 3-colorability is* NP*-complete*, SIGACT News **5** (1973), 19–25.