Algoritmos para junções em digrafos acíclicos e uma aplicação na Antropologia

Álvaro Junio Pereira Franco

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Programa: Ciência da Computação
Orientador: Prof. Dr. Carlos Eduardo Ferreira

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo, 18 de dezembro de 2013

Algoritmos para junções em digrafos acíclicos e uma aplicação na Antropologia

Esta versão da tese contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 18/12/2013. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. Carlos Eduardo Ferreira (orientador) - IME – USP
- Prof. Dr. José Coelho de Pina Junior - IME – USP
- Prof. Dr. Orlando Lee - UNICAMP
- Profª. Drª. Márcia Rosana Cerioli - UFRJ
- Prof. Dr. Marcelo Henriques de Carvalho - UFMS

Agradecimentos

É com muita alegria que escrevo os meus agradecimentos por mais uma fase concluída. A minha tese de doutorado não é composta somente por este livreto. O texto deste trabalho é o resultado de uma série de eventos que ocorreram comigo durante estes sete anos e meio que eu passei no IME. Vou adicionar nos meus agradecimentos alguns desses eventos que foram fundamentais para a conclusão desta tese. São eventos simples que me deram energia para continuar os meus estudos em São Paulo.

Eu gostaria de começar agradecendo a Deus pela incrível oportunidade que eu tenho de estar experimentando essas coisas maravilhosas da vida.

Aos meus pais, Álvaro e Suely, eu gostaria de agradecer pela oportunidade que me deram de nascer. Eles sempre apoiaram as minhas decisões. Quando eu decidi viver em São Paulo sabíamos que não seria fácil. Eu chegaria no IME sem garantia de bolsa, e mesmo com todas as dificuldades que tinhamos na época, eu recebi o apoio dos meus pais. A nossa família é uma família abençoada. Agradeço aos nossos ancestrais (principalmente aqueles mais próximos como os meus avós) por terem nos educado dessa maneira. Nossa família é unida, e sentimos amor um pelo outro.

Lembro do meu primeiro dia em São Paulo, em Agosto de 2006. Meu pai estava comigo. Ele não queria que eu viesse sozinho por vários motivos. Um deles é que eu ainda não tinha uma casa para morar. Chegamos no IME por volta das 13 horas no primeiro dia de aula da pós-graduação. Eu tinha uma aula com o Prof. Carlinhos às 14 horas. Fui para a aula e o meu pai foi procurar um lugar para eu morar. Ele encontrou uma república (a casa do Mário), passamos a noite nesta casa, e meu pai voltou para Goiânia no dia seguinte. Eu continuei na casa do Mário. Nunca esquecerei do que meu pai fez por mim nos meus primeiros dias em São Paulo. Muito obrigado pai.

A minha mãe estava em Goiânia no dia que eu cheguei em São Paulo. Mas tenho certeza que ela estava comigo em espírito. Lembro que quando acabou a minha bolsa de mestrado, eu fui morar no CRUSP e minha mãe ficou chateada com isso. Acho que era preocupação de mãe, algo que vem da natureza das mães. Como eu queria continuar "só estudando", isto é, não queria começar a trabalhar em uma empresa, eu não tinha outra opção a não ser ir morar no CRUSP. Nesta época, a minha mãe dizia para eu rezar a prece de Cáritas várias vezes. Isso iria me ajudar. E eu rezava e certamente me ajudou. Este é outro momento que eu não esquecerei. Muito obrigado mãe.

Eu morei no CRUSP por um ano. Foram seis meses no alojamento e seis em um apartamento. O CRUSP é uma ótima opção para os alunos com baixa renda familiar e funciona razoavelmente bem. As assistentes sociais me atenderam com carinho e estou muito agradecido pela oportunidade que me deram quando eu precisei.

Durante minha vida, quase sempre os estudos estavam como primeiro plano. Eu desejava que este fascínio pelos estudos estivessem também nos meus irmãos, Rakel e Ricardo. Fiquei muito feliz nos dias que minha irmã se formou em Administração e o meu irmão em Computação. Por outro lado, fique triste por não ter conseguido ir à formatura dos dois. Eu estava em São Paulo e os dois estavam em Goiânia. Espero que eles me perdoem por esta falta. Os meus irmãos foram muito importantes durante meu processo de adaptação em uma nova cidade. Eles me incentivavam e torciam por mim. Isso faz diferença quando estamos distante de quem gostamos.

Aprendi muito com eles, e agradeço pelas energias positivas.

Nos anos que passei no IME conheci várias pessoas: o Francisco (Chicão), o Anderson (Pagode), o Carlos (Carlitos), o Alexandre (Chegado), o Wanderley (Wander), o Facci, o Lobato, o Rafael (Rafa Marques), o Márcio, o Ander, o Albano, e muitos outros. Peço que me perdoem os amigos cujo nomes não aparecem aqui. Obrigado meus queridos amigos pelos momentos de estudos e diversões que passamos juntos. Esta tese tem muito das nossas conversas durante um café, um almoço, ..., vocês sabem disso, certo? Também não me esqueço das nossas conversas sobre os mais variados assuntos. Obrigado a todos e um abraço muito especial aos amigos que estavam mais presentes.

Quando eu cheguei no IME fui recebido pela Profª. Nami Kobayashi. Por algumas vezes tive o prazer de trabalhar (como monitor) em disciplinas que ela ministrava. Quando eu ia na sua sala, conversávamos sobre coisas do trabalho e, além disso, ela sempre me perguntava se eu estava bem, como eu estava indo com as disciplinas e com o meu mestrado/doutorado. Agradeço muito a Profª. Nami por todos os conselho e por ter me ensinado a dinâmica de uma disciplina.

A Catalina, o John, o Simon, a Aleja, o Javier, e o Nils foram as pessoas que me deram a oportunidade das amizades estrangeiras. Obrigado amigos pelos momentos felizes que tivemos e pela oportunidade de aprender um pouquinho o espanhol.

Os Profs. Carlinhos, Coelho, Leliane, Yoshiko, Nami, Alair, Cris, Mandel, Reverbel, José Augusto, Luis Carlos, e Rudi são os professores que eu vi trabalhando inclusive dentro da sala de aula. Agradeço a eles(as) pela experiência que compartilharam comigo. Passarei adiante os ensinamentos que vocês me deram. Eu tentarei ser didático, organizado e justo dentro da sala de aula assim como vocês são.

Um evento importante que aconteceu durante a minha pós no IME foi o futebol semanal. Esta atividade física fazia bem para o meu corpo e para minha mente embora às vezes os jogos mexiam muito com a minha mente porque costumavam ser "quentes" e sérios. Conheci grandes pessoas lá: Profs. Saulo, Clodoaldo, Jesse, entre outros.

O meu orientador no mestrado e no doutorado foi o Prof. Carlos Eduardo Ferreira. Ele é incrível. Uma pessoa simples, humilde e paciente. Ele estava sempre disponível quando eu tinha uma dúvida. Sinceramente, eu não sei o que seria de mim no IME sem ele como orientador. Agradeço a ele por tudo que me ofereceu principalmente pela oportunidade de estudar juntos. Foi através dele que percebi que seria possível sim terminar meus estudos no IME. Foi através dele que ampliei meus conhecimentos com cursos na Alemanha, viagens à conferências, entre outras coisas. Se hoje eu sou uma pessoa mais esclarecida, isso é devido ao esforço e a paciência do meu querido orientador. Muito obrigado Carlinhos.

Eu conheci algumas pessoas no Centro Espírita Aurora dos Aprendizes uma delas (o Sr. Accaccio) é um senhor muito simpático que gosta de ajudar outras pessoas. Aprendi com o Sr. Accaccio a importância do trabalho no bem e da prática do bem. Sinto-me tranquilo em dizer que o lema dele é "Trabalhar com dedicação e amor no bem para que tenhamos um mundo (interior e exterior) melhor". É claro que tenho muitos outros amigos do centro como a Dona Teresa, Dona Isabel, a Dona Verônica, a Ariane, o Fernando, a Luciana, o Macos, a Eliane, o Fábio, a Ivone (vale destacar que ela faz bolos maravilhosos), o Luiz, e muitos outros. Sinto muito não conseguir colocar os nomes de todos aqui mas queria enfatizar que todos são importantes. Aos meus queridos amigos da escola dos Aprendizes do Evangelho queria dizer muito obrigado por compartilharem as suas experiências e o seu tempo comigo.

Outra coisa que eu acho que vale a pena destacar é a beleza da cidade de São Paulo. Parabéns aos paulistanos pela cidade que tem, fruto de quem vive e de quem viveu nela. A cidade de São Paulo, com toda a sua estrutura, foi importante na minha formação como pessoa. Sem contar que a cidade é simplesmente demais com todos os seus shoppings, parques, museus, cinemas, feiras, restaurantes, shows, e espetáculos dos mais váriados. Adorei as experiências que passei por aí.

Para o final eu deixei o principal evento que me aconteceu em São Paulo. Eu conheci a Priscila, minha esposa, no início do meu segundo ano no IME nas aulas de Geometria Computacional da Cris. Em um dos momentos em que eu estava com o Carlitos, eu disse a ele: "Carlos, essa menina é a pessoa certa". E é mesmo! Desde Agosto de 2007 estamos juntos. Logo começamos a namorar e nos casamos em julho de 2010. Nós nos divertimos muito em São Paulo. Foi através dela que eu conheci a Catalina, o Prof. Alexandre Roma, e vários outros amigos. Eu adorei conhecer a sua família, seus pais, irmãos, e sobrinhos. Um dos sobrinhos dela, o Mateus, passou uma temporada na nossa casa para fazer um curso de difusão do IME no programa de verão.

Juntos, eu e a Priscila, tivemos muitas alegrias e tristezas. Passamos por coisas da vida que acredito vir para nos ensinar algo. Sem minha querida companheira e amiga eu não teria conseguido terminar o meu doutorado. Ela me apoiou e me acompanhou em todas as minhas decisões. Me ajudou a compreender o respeito que devemos ter aos animais. Me incentivou e me animou nos momentos de dificuldade. Me apresentou o lado positivo da vida e me fez ser uma pessoa mais otimista. Priscila, obrigado por tudo que tem feito por mim, obrigado pelas ótimas comidas, obrigado pelos gatinhos que temos, obrigado por aceitar a viver comigo.

Resumo

Franco, Á. J. P. **Algoritmos para junções em digrafos acíclicos e uma aplicação na Antropologia**. 2013. 97 f. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2013.

Neste trabalho consideramos um problema da Antropologia. A modelagem de sociedades e casamentos de indivíduos é feita com grafos mistos e encontrar caminhos disjuntos é uma questão central no problema. O problema é $\mathcal{NP}$-completo e, quando visto como um problema parametrizado, ele é $\mathcal{W}[1]$-difícil. Alguns subproblemas que surgem durante o processo de obter uma solução para o problema, envolvem caminhos disjuntos e podem ser resolvidos em tempo polinomial. Implementamos algoritmos polinomiais que são usados em uma ferramenta desenvolvida para solucionar o problema na Antropologia considerado. Nossa solução funcionou bem para as sociedades fornecidas pelos nossos parceiros.

Palavras-chave: algoritmos para junções, digrafos acíclicos, caminhos disjuntos, complexidade computacional, antropologia estrutural.

Abstract

Franco, Á. J. P. **Algorithms for junctions in acyclic digraphs and an application in the Anthropology**. 2013. 97 p. PhD Thesis - Institute of Mathematical and Statistics, University of São Paulo, São Paulo, 2013.

In this work we consider a problem from the Anthropology. The model of the societies and the marriages of individuals is done with mixed graphs and to find disjoint paths is a central question in the problem. The problem is $\mathcal{NP}$-complete and $\mathcal{W}[1]$-hard when it is considered a parameterized problem. Some subproblems that arise during the process to obtain a solution for the problem, involve disjoint paths and can be solved in polynomial time. We implemented some polynomial algorithms that are used in a tool developed to solve the problem in the Anthropology considered. Our solution worked well for the societies provided by our partners.

Keywords: algorithms for junctions, acyclic digraph, disjoint paths, computational complexity, structural anthropology.

Sumário

Lista de Figuras **xi**

Lista de Tabelas **xv**

1 Introdução **1**

2 Preliminares **5**
2.1 Definições Fundamentais 5
2.2 Matrizes e o Fecho Transitivo 9
2.3 Junções e Ancestrais 9
2.4 Anéis em Grafos Mistos 10
2.5 Algoritmos com Fases de Pré-processamento e Consultas 11
2.6 Aspectos de Complexidade Computacional 12
2.6.1 Problemas Parametrizados 13
2.6.2 Redução entre Problema Parametrizados 13
2.6.3 A Classe $\mathcal{FPT}$ 13
2.6.4 As Classes $\mathcal{W}[P], \mathcal{W}[1], \mathcal{W}[2], \ldots$ 14

3 Algoritmos para Junções em Digrafos - Revisão da Literatura **17**
3.1 Ancestral Comum mais Próximo em Árvores Enraizadas 17
3.1.1 Algoritmo de Aho, Hopcroft e Ullman 18
3.1.2 Algoritmo de Harel e Tarjan 19
3.1.3 Algoritmo de Berkman e Vishkin 20
3.2 Um Ancestral Comum mais Próximo em Digrafos Acíclicos 22
3.2.1 Algoritmo de Baumgart, Eckhardt, Griebsch, Kosub e Nowak 22
3.2.2 Algoritmo de Bender, Farach-Colton, Pemmasani, Skiena e Sumazin . . . 23
3.2.3 Algoritmo de Czumaj, Kowaluk e Lingas 24
3.2.4 Contribuição 26
3.3 Todos os Ancestrais Comuns mais Próximos em Digrafos Acíclicos 26
3.3.1 Algoritmos de Baumgart, Eckhardt, Griebsch, Kosub e Nowak 26
3.3.2 Algoritmos de Eckhardt, Mühling e Nowak 27
3.3.3 O Problema *todos-acps-todos-pares* em Digrafos Gerais 28
3.3.4 Contribuição 28
3.4 Uma Junção em Digrafos Acíclicos 28
3.4.1 Algoritmo de Yuster 29
3.4.2 Contribuição 31
3.5 Todas Junções em Digrafos Acíclicos 31
3.5.1 Algoritmos Polinomiais para o Problema *s-junção-todos-pares* 32
3.5.2 Contribuição 33
3.6 Árvores de Dominadores 34
3.6.1 Algoritmo de Ramalingam e Reps 35
3.6.2 Contribuição 35

4 Algoritmos para Junções em Digrafos Acíclicos - Contribuições 37
4.1 Definição e Complexidade do Problema *s-junção-k-pares* 37
4.2 Um Algoritmo Linear para o Problema *s-junção-k-pares* 37
4.2.1 Algoritmos de Programação Dinâmica e Recursivo 41
4.2.2 Obtendo um Par de Caminhos Vértice-Disjuntos 47
4.3 Usando uma Partição em Estrelas nos Problemas de Junções 48
4.4 Um Algoritmo Linear para o Problema *s-arco-junção-k-pares* 50
4.5 Multiplicação de Matrizes Booleanas Retangulares 51
4.6 Uma Relação min-max em Grafos Fluxos 52
4.6.1 Construindo uma Árvore de Dominadores para Grafos Fluxos Redutíveis 57

5 Uma Aplicação na Antropologia 59
5.1 Uma Aplicação na Antropologia - Definição e Complexidade 59
5.1.1 Limitando o Digrafo de Entrada 62
5.1.2 Complexidade Parametrizada 64
5.2 Relação entre os Problemas *um-k-anel* e *subdigrafo-homeomorfo* 65
5.3 O Problema *um-k-anel* com k Fixo 66
5.3.1 Melhoramentos 68
5.4 Experimentos 70
5.4.1 Experimentos - Primeira Parte 71
5.4.2 Experimentos - Segunda Parte 77

6 Conclusão 81

Referências Bibliográficas 83

Anexos 89

A Pseudocódigos 91
A.1 Pseudocódigo para o Problema *s-junção-k-pares* 91
A.2 Pseudocódigo para o Problema *s-arco-junção-k-pares* 92
A.3 Pseudocódigo para o Problema *árvore-de-dominadores* 92

B Desenho de algumas sociedades 93

Índice Remissivo 95

Lista de Figuras

1.1 (a) - Um casamento entre primos. (b) - Um casamento entre irmãos. 1

2.1 (a) - Um grafo misto G com duas arestas. (b) - O digrafo induzido pelos arcos de G. (c) - O grafo induzido pelas arestas de G. 6
2.2 (a) - Um digrafo D. (b) - O digrafo linha de D. 7
2.3 (a) - Um digrafo D. (b) - Uma partição dos arcos de D. 7
2.4 (a) - Um grafo fluxo G com raiz em 1. (b) - A árvore de dominadores de G. . . . 8
2.5 (a) - Um grafo fluxo redutível D com raiz em 0. (b), (c) - as duas agbps de D com raiz em s. (d) - o subgrafo acíclico maximal de D. 8
2.6 Os vértices marcados são acps e junções de alguns pares de vértices em árvores enraizadas, digrafos acíclicos e digrafos gerais. 10
2.7 a - Um 1-anel. b - Um 2-anel. 11
2.8 (a) - Um k-anel. (b) - Um 2-anel degenerado. 11
2.9 Diferentes consultas em diferentes digrafos. 12
2.10 Um circuito booleano (ou lógico). Os vértices x, y e z são os vértices de entrada. O vértice s é o vértice de saída. O vértice OU é irrestrito. 14
2.11 Relações entre as classes $\mathcal{FPT}, \mathcal{W}[1], \mathcal{W}[2], \ldots, \mathcal{W}[P]$ [FG06]. 14

3.1 Uma árvore enraizada e uma lista de consultas 18
3.2 Uma árvore enraizada. 21
3.3 Matrizes retangulares A^i e B^i [CKL07]. 25
3.4 Ordenação topológica que maximiza o valor de z. 27
3.5 Digrafos D, D' e D''. 29
3.6 Digrafo acíclico com rótulo nos vértices. 30
3.7 Digrafos $D'_{6,7}$ e $D'_{8,9}$ correspondentes ao digrafo da Figura 3.6. Os arcos $1' \to 1''$ têm capacidade igual a 2. Os demais arcos têm capacidade igual a 1. 32
3.8 Cada vértice na primeira linha é uma junção de todo par de vértices na segunda. Assim, listar todas as junções de todos os $\Omega(n^2)$ pares poderia consumir tempo $\Omega(n^3)$. 33

4.1 Uma agbp e um ciclo composto pelos caminhos de y para v', de v' para x, de x para u' e de u' para y. Isso não pode ocorrer em um digrafo acíclico. 39
4.2 Uma agbp e novos caminhos internamente vértice-disjuntos – Lema 4.3. Caso 1. Um dos caminhos possui o vértice $z = \mathcal{ACP}^{T^s}_{u,v}$. 40
4.3 Uma agbp e novos caminhos internamente vértice-disjuntos – Lema 4.3. Caso 2. Um dos caminhos possui o vértice $z = \mathcal{ACP}^{T^s}_{u,v}$. O vértice $y \in Q$ é o mais próximo de z em $(P \cup Q) \cap R$. 40
4.4 Uma agbp e novos caminhos internamente vértice-disjuntos – Corolário 4.1. Casos 1 e 2. 41

4.5 Existem $\Omega(n)$ vértices nos caminhos de x_1 para y_1 e em T^{y_1} (resp. x_2, y_2 e T^{y_2}). Existe um arco ligando todo vértice destes caminhos para todos os descendentes próprios de y_1 (resp. y_2) representado pelos arcos mais largos e tracejados ligando os vértices maiores. Para um número quadrático de descendentes próprios de y_1 (resp. y_2), o algoritmo de programação dinâmica descrito anteriormente processa $\Omega(n)$ arcos. Neste caso, o tempo total gasto é $\Omega(n^3)$. 42
4.6 Um digrafo acíclico, uma agbp, e a primeira e a última configuração da partição dos vértices. O valor pós-ordem de cada vértice está entre parênteses. 43
4.7 Novos caminhos internamente vértice-disjuntos – Lema 4.6. 45
4.8 Algoritmo que encontra dois caminhos internamente vértice-disjuntos. Os vértices em negrito são representantes e o número em cada vértice é o valor pós-ordem do vértice em T^s. 48
4.9 Os filhos do vértice w serão representantes no contexto arco-junção pois qualquer par de vértices u e v com u em T^{w_i} e v em T^{w_j} tem s como arco-junção. 51
4.10 A construção de um digrafo acíclico D a partir das matrizes A e B dadas. Os vértices com grau de entrada iguais a zero foram omitidos. 52
4.11 Podemos ter $\mathcal{A}^s = \{s, x\}$, ou $\mathcal{A}^s = \{s, x, v\}$, ou $\mathcal{A}^s = \{s, x, v, u\}$. 54
4.12 O vértice q_3 é o último vértice em R comum aos caminhos P e Q. Em destaque aparecem caminhos vértice-disjuntos. 56
4.13 Um grafo fluxo D com raiz em 0 e sua árvore de dominadores T. A raiz de T mais os filhos da raiz são os elementos do conjunto mínimo de dominadores não triviais para todo vértices de D, isto é, $\mathcal{A}^0 = \{0, 1, 2, 3, 4\}$. O conjunto $\mathcal{B}^0_0 = \{0\}$ e $\mathcal{B}^0_u = \{i\}$ para $i = 1, 2, 3, 4$ e para todo vértice u descendente de i em T. 57

5.1 Existem quatro clãs na figura: $\mathcal{C}_1 = \{1\}$, $\mathcal{C}_2 = \{2, 3, 4\}$, $\mathcal{C}_3 = \{5, 9\}$ e $\mathcal{C}_4 = \{6, 7, 8\}$. O casamento da mulher 1 com um homem do clã $\mathcal{C}_3$ redobra o casamento do seu tio, o indivíduo 4, com uma mulher do clã $\mathcal{C}_3$ [dPFdS13]. 60
5.2 Os conjuntos $\mathcal{C}_1 = \{1, 2, 3, 4\}$, $\mathcal{C}_2 = \{5, 6, 7, 8, 9\}$ e $\mathcal{C}_3 = \{10, 11, 12, 13, 14\}$ são três clãs. O casamento da mulher 1 do clã $\mathcal{C}_1$ com o homem 14 do clã $\mathcal{C}_3$ reencadeia as alianças 4 e 5, e 9 e 10 [dPFdS13]. 60
5.3 Tempo exponencial para listar todos os 1-anéis. 61
5.4 Um k-anel em D'. 62
5.5 Estamos olhando somente para os digrafos acíclicos. À esquerda temos dois exemplos de digrafos acíclicos de entrada e à direita temos os dois digrafos correspondentes com as restrições nos graus de entrada e saída em cada vértice. . . 63
5.6 Um caminho em H e um caminho em H'. 64
5.7 Um k-anel $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ em função dos mapeamentos m_V e m_A do digrafo padrão H sobre o digrafo de entrada D. 66
5.8 Na primeira figura temos uma entrada do problema *um-2-anel*: o grafo misto acíclico H e $\mathcal{E} = (u_1 - v_2, u_2 - v_1)$. Na segunda, fixados dois vértices s_1 e s_2 de H, temos uma entrada para o problema 4-*caminhos-vértice-disjuntos*: os pares ordenados $(s_1, u_1), (s'_1, v_1), (s_2, u_2), (s'_2, v_2)$, e o digrafo acíclico D induzido pelos arcos de H. 68
5.9 O vértice s_1 está em $\mathcal{J}_{u_1,v_1}$ e o vértice s_2 está em $\mathcal{J}_{u_2,v_2}$. Os descendentes de y_1 estão no triângulo ilustrado. Como não existe caminho de y'_1 para v_1, o segundo melhoramento não permite que a busca continue pelo vértice (x_1, y'_1, x_2, y_2) de D_{s_1,s_2}. 69
5.10 O vértice s_1 está em $\mathcal{J}_{u_1,v_1}$ e o vértice s_2 está em $\mathcal{J}_{u_2,v_2}$. Como $\mathcal{S}_{y_1 \rightsquigarrow v_1} \cap \mathcal{S}_{x_2 \rightsquigarrow u_2} \neq \emptyset$ (o vértice z aparece em ambos os conjuntos), o terceiro melhoramento não permite que a busca continue pelo vértice (x_1, y_1, x'_2, y_2) de D_{s_1,s_2}. 70

5.11 Os casamentos são $u - x$ e $v - y$. Os colegas em casamento estão representados nesta figura pelos indivíduos ligados por pontilhados duplos. 71
5.12 1º estatística - Para inteiros não-negativos $i = 0, 1, \ldots, n$, quantos casamentos têm exatamente i junções? 73
5.13 2º estatística - Para cada vértice u, quantos casamentos têm u como junção? . . 74
5.14 3º estatística - Para inteiros não-negativos $i = 0, 1, \ldots, n$, quantos colegas em casamentos têm exatamente i junções? 75
5.15 4º estatística - Para cada vértice u, quantos colegas em casamentos tem u como junção? 76

6.1 Um k-anel que reencadeia casamentos. 82

B.1 Sociedade Arara. Note uma junção 1005 (segundo vértice no segundo nível) para o casamento envolvendo os indivíduos 1046 (segundo vértice no terceiro nível) e 1058 (no meio do quinto nível). 93
B.2 Sociedade Xavante. 94

Lista de Tabelas

1.1 Problemas sobre junções em digrafos acíclicos 2

5.1 Eppstein com DFS e M1 - Problema *um*-2-*anel* 77
5.2 Eppstein com BFS, M1, M2 e M3 - Problema *um*-2-*anel* 78
5.3 Eppstein com DFS, M1, M2 e M3 - Problema *um*-2-*anel* 78
5.4 Eppstein com BFS, M1, M2 e M3 - Problema *um*-3-*anel* 78
5.5 Eppstein com DFS, M1, M2 e M3 - Problema *um*-3-*anel* 79
5.6 Arcos, M2 e M3 no digrafo de Eppstein - DFS *um*-3-*anel* 79
5.7 Todos os caminhos - Problema *um*-2-*anel* 80
5.8 Todos os caminhos - Problema *um*-3-*anel* 80

6.1 Algoritmos rápidos para problemas sobre junções - todos os pares 81
6.2 Algoritmos rápidos para problemas sobre junções - k pares 82

1

Introdução

A Ciência da Computação tem encontrado aplicações de interesse nas mais diversas áreas. Problemas em Genética, Ecologia, Economia ou Ciências Humanas têm produzido dados em quantidades enormes, e a análise destes dados tem exigido métodos cada vez mais eficientes. Este fenômeno deu origem a toda uma área de Computação, chamada *e-science*.

Este trabalho tem sua origem na necessidade de pesquisadores da área de Antropologia de usar recursos computacionais sofisticados na análise de redes de parentesco. O professor Dr. Marcio Ferreira da Silva do Departamento de Antropologia da Faculdade de Filosofia, Letras e Ciências Humanas da Universidade de São Paulo e o professor Dr. João dal Poz Neto do Departamento de Ciências Sociais do Instituto de Ciências Humanas da Universidade Federal de Juiz de Fora apresentaram um problema envolvendo caminhos vértice-disjuntos em grafos mistos que modelam sociedades e casamentos dos indivíduos. Os casamentos dão origem a ligações entre vértices do grafo que formam, juntamente com os arcos de filiação, estruturas que são de interesse dos pesquisadores de Ciências Humanas. Por exemplo, os participantes de um casamento podem ser primos (ter o mesmo avô) como na Figura 1.1 (a). Ou, ao considerar dois casamentos podemos observar que eles ligam duas famílias, onde dois irmãos casam com duas irmãs (veja Figura 1.1 (b)). Tais estruturas surgem nas diversas sociedades de acordo com os tabus que inviabilizam alguns casamentos e incentivam outros.

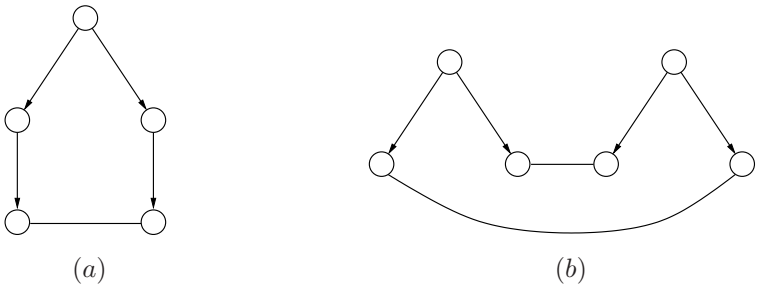


Figura 1.1: (a) - Um casamento entre primos. (b) - Um casamento entre irmãos.

O interesse nessas estruturas começou com Lévi-Strauss [LS69] com o estudo da *teoria das alianças matrimoniais*. Nesta teoria as regras e os tabus de cada sociedade são pontos importantes. Segundo Lévi-Strauss, em qualquer sociedade existem regras que possibilitam ou impossibilitam alianças matrimoniais. Determinar uma regra a partir de casamentos entre indivíduos garantiria determinar a vida social da sociedade através da circulação dos indivíduos casados. Com o passar dos anos, a teoria das alianças matrimoniais evoluiu e as estruturas desejadas tornaram-se mais complexas.

Caminhos disjuntos em digrafos acíclicos estão no centro deste trabalho. Um de nossos principais interesses é encontrar algoritmos rápidos para junções em digrafos. Dado um digrafo D, uma *junção* dos vértices u e v em D é um vértice s em D com caminhos internamente vértice-disjuntos de s para u e de s para v. Uma junção s dos vértices u e v é um *ancestral*

comum mais próximo (**acp**) de u e v, se não existir caminho em D de s para s', onde s' é uma outra junção dos vértices u e v. Existem diversos trabalhos na literatura que tratam do problema de determinar junções. Alguns deles apresentam algoritmos polinomiais para resolvê-los. Nossa principal preocupação, neste e nos outros problemas considerados nesta tese, foi mostrar algoritmos eficientes, elegantes e de implementação simples. Dessa forma, evitamos desenvolver algoritmos que fazem uso de outros que somente na teoria possuem uma boa complexidade de tempo. Veremos que muitos algoritmos rápidos da literatura resolvem problemas sobre junções usando o algoritmo rápido para multiplicar matrizes. Isso torna mais difícil a utilização destes algoritmos na prática, na solução do problema. Dado que existe um algoritmo rápido teórico, existe também um algoritmo rápido que funciona bem na prática? Tentamos responder a esta pergunta positivamente.

Estudamos nesta tese diversos problemas ligados a caminhos disjuntos. Em alguns destes problemas queremos encontrar todas as junções de um par de vértices. Outras vezes, nosso interesse é encontrar uma junção para todos os pares de vértices, ou todas as junções para todos os pares de vértices, ou mesmo, fixado um vértice s, encontrar todos os pares para os quais s é uma junção. Na Tabela 1.1 mostramos uma lista destes problemas e o que desejamos em cada um deles. Para o primeiro problema é dada uma árvore enraizada. Para os demais problemas é dado um digrafo acíclico (não necessariamente uma árvore enraizada).

Tabela 1.1: Problemas sobre junções em digrafos acíclicos

Problema	Determinar
acp-todos-pares	O acp para quaisquer vértices u e v
representante-acp-todos-pares	Um acp para quaisquer vértices u e v
representante-junção-todos-pares	Uma junção para quaisquer vértices u e v
todos-acps-todos-pares	Todos os acps para quaisquer vértices u e v
todas-junções-todos-pares	Todas as junções para quaisquer vértices u e v
s-acp-todos-pares	Todos os pares que têm s como um acp
s-junção-todos-pares	Todos os pares que têm s como uma junção

Em muitos dos algoritmos estudados para estes problemas, a ideia é pré-processar o digrafo da entrada, construindo uma estrutura de dados capaz de encontrar rapidamente uma junção para qualquer par de vértices do digrafo. Um caso interessante ocorre quando o digrafo da entrada é uma árvore enraizada. Dada uma árvore enraizada T com n vértices, é possível pré-processar T em tempo $O(n)$, construindo uma estrutura de dados capaz de responder em tempo constante qual é a junção para qualquer um dos $O(n^2)$ pares de vértices de T. Podemos citar alguns trabalhos que tratam o problema com uma árvore enraizada na entrada: Aho, Hopcroft e Ullman [AHU76], Harel e Tarjan [HT84], Berkman e Vishkin [BV94], Cole e Hariharan [CH05], Nykänen e Ukkonen [NU94] e Wen [Wen94]. Em 2001, Bender et al. [BFCP$^+$05] definiram o problema de encontrar certas junções em digrafos acíclicos. Eles adicionaram ao tempo do algoritmo, o expoente ω que aparece no algoritmo que multiplica matrizes. Com isso, o algoritmo rápido descrito tem mais valor teórico do que prático. A partir daí, outros problemas foram definidos e o uso do algoritmo de multiplicar matrizes continuou, por exemplo, nos trabalhos de Baumgart et al. [BEG$^+$07], Czumaj et al. [CKL07], Eckhardt et al. [EMN07] e Yuster [Yus08]. A grosso modo, os três primeiros problemas da Tabela 1.1 recebem um digrafo e devolvem uma estrutura de dados capaz de encontrar rapidamente uma determinada junção para qualquer par de vértices do digrafo. Os dois próximos problemas, recebem um digrafo e devolvem uma estrutura de dados capaz de encontrar todas as junções para qualquer par de vértices do digrafo. Os dois últimos problemas recebem um digrafo e um vértice do digrafo e devolvem uma estrutura de dados capaz de encontrar todos os pares de vértices que possuem o

vértice dado como junção. Com exceção do primeiro e do último, a complexidade de tempo do algoritmo mais rápido de cada problema depende da multiplicação de matrizes.

Para alguns problemas citados anteriormente, suponha que os pares de vértices para os quais desejamos saber uma junção são dados na entrada. Os problemas *representante-acp-k-pares*, *representante-junção-k-pares*, *todos-acps-k-pares* e *todas-junções-k-pares* surgem naturalmente dos correspondentes na Tabela 1.1. Agora são dados um digrafo acíclico e k pares de vértices. Queremos construir uma estrutura de dados capaz de encontrar rapidamente junções para cada par de vértices dado. É claro que os problemas correspondentes anteriores, que encontram junções para todos os pares, podem ser usados para encontrar junções de k pares dados. Se este for o caso, e adicionando algumas restrições no tamanho do digrafo e no número de pares de vértices dados, obtemos algoritmos mais rápidos. É importante citar que os algoritmos que desenvolvemos para estes casos não usam multiplicação de matrizes.

Como dissemos no início dessa introdução, este trabalho tem sua origem em uma aplicação da área de Ciências Humanas, ou mais especificamente, no relacionamento entre Antropologia e Computação. O problema envolve caminhos vértice-disjuntos em redes de parentesco. Do ponto de vista computacional, o problema de interesse na Antropologia é $\mathcal{NP}$-completo, até mesmo se a rede é acíclica e cada vértice da rede ou tem grau de entrada no máximo 2 e de saída no máximo 1, ou tem grau de entrada no máximo 1 e de saída no máximo 2. Ou seja, até mesmo para redes mais restritivas que redes de sociedades reais, pois não é comum aparecer nas sociedades uma restrição tão forte no número de filhos de cada indivíduo. Ainda mostramos que este problema, quando visto como um problema parametrizado, é $\mathcal{W}[1]$-difícil. Porém, o problema surge nas aplicações e é de interesse dos antropólogos resolvê-lo. Mostramos que é possível encontrar tais estruturas para as redes de parentesco disponíveis aplicando, por exemplo, o método de Eppstein [Epp95] ao problema quando um parâmetro da entrada é fixo.

No Capítulo 2, definimos os conceitos e notações usadas neste trabalho. No Capítulo 3, estão a revisão da literatura dos algoritmos para junções em digrafos acíclicos, e as definições de alguns problemas considerados. No Capítulo 4, apresentamos o principal algoritmo desenvolvido neste trabalho. O algoritmo é simples, elegante, eficiente e de fácil implementação. Ele resolve o problema *s-junção-todos-pares* sobre digrafos acíclicos. É importante notar que, ainda que existissem algoritmos na literatura para o problema com igual complexidade [GT04], sua implementação não é tão simples quanto a que propomos. Ainda propomos uma estrutura de dados de tamanho linear que codifica de forma eficiente a solução do problema. Esta é, em nossa opinião, outra contribuição importante desse trabalho. Ainda no Capítulo 4, mostramos como é possível obter caminhos internamente vértice-disjuntos de s para u e de s para v para vértices u e v que têm s como junção, mostramos como adaptar o algoritmo para resolver o problema *s-arco-junção-todos-pares*, uma versão arco-disjunto do problema *s-junção-todos-pares*. Mostramos como usar o algoritmo para obter a multiplicação de duas matrizes booleanas. Mostramos como construir uma *árvore de dominadores* de um grafo fluxo redutível, e finalmente mostramos uma relação min-max em grafos fluxos. No Capítulo 5, definimos o problema de interesse dos antropólogos, mostramos que este problema é $\mathcal{NP}$-completo e que, quando visto como um problema parametrizado, ele é $\mathcal{W}[1]$-difícil. Mostramos que a dificuldade do problema continua quando restringimos a entrada, descrevemos uma relação entre o problema de interesse dos antropólogos com um problema da área de digrafos, descrevemos como aplicar o método de Eppstein para resolver o problema em tempo polinomial quando um parâmetro da entrada é fixo. Por último, apresentamos alguns melhoramentos no método de Eppstein e concluímos com experimentos realizados sobre sociedades reais de diferentes regiões do Brasil fornecidas pelos nossos parceiros antropólogos.

2

Preliminares

Neste capítulo estabelecemos a notação e descrevemos as definições fundamentais usadas. Na Seção 2.1, descrevemos as definições fundamentais como grafos, digrafos, grafos mistos, digrafos linha, árvores geradoras de busca em profundidade, grafos fluxos, grafos fluxos redutíveis e árvores de dominadores. O consumo de tempo de muitos algoritmos descritos neste trabalho depende do consumo de tempo do algoritmo mais rápido para multiplicar matrizes. Por isso, na Seção 2.2, introduzimos o conceito de expoente da multiplicação de matrizes e apresentamos uma aplicação clássica em digrafos. Na Seção 2.3, definimos vértices ancestrais e descendentes de um determinado vértice, e junções e ancestrais comuns mais próximo de um determinado par de vértices. Na Seção 2.4, definimos uma estrutura de interesse em grafos mistos. Na Seção 2.5, apresentamos uma metodologia usada para resolver um problema de caminhos disjuntos em digrafos. Na Seção 2.6, descrevemos brevemente alguns aspectos da complexidade computacional clássica e da complexidade parametrizada.

2.1 Definições Fundamentais

Um *grafo* é formado por um conjunto finito e não vazio de *vértices* e um conjunto finito de *arestas*. Um elemento do conjunto de arestas conecta dois elementos distintos do conjunto de vértices. O vértice x é *adjacente* ao vértice y se existe uma aresta $x - y$. A aresta $x - y$ *incide* nos vértices x e y. Dados um grafo G e dois vértices v_1 e v_k de G, um *caminho* de v_1 para v_k em G é uma sequência de vértices sem repetição $P = v_1, v_2, \ldots, v_k$ tal que $v_i - v_{i+1}$ é uma aresta em G para $i = 1, \ldots, k-1$. Se v_1 é adjacente a v_k, então P mais a aresta $v_k - v$ é um *ciclo* em G. Dizemos que G é *conexo* se, para quaisquer vértices u e v, existe um caminho de u para v. Uma *árvore* é um grafo conexo sem ciclos.

Um *digrafo* é um grafo com orientação nas arestas. Neste caso, as conexões entre vértices são chamadas de *arcos*. Portanto, um digrafo é formado por um conjunto finito e não vazio de vértices e um conjunto finito de arcos. Neste texto, o número de vértices e o número de arcos são denotados por n e m, respectivamente. Dado um digrafo D, se não consideramos a orientação dos seus arcos, então temos o *grafo correspondente* de D. Um digrafo é conexo[1], se o seu grafo correspondente é conexo. Considere um digrafo D e o seu grafo correspondente G. Todos os vértices adjacentes em G são também adjacentes em D. Portanto, o vértice x é adjacente a y e y é adjacente a x para qualquer arco $x \to y$ em D. O arco $x \to y$ tem *pontas* x e y. A *ponta inicial* é x e a *ponta final* é y.

Neste texto, consideramos apenas digrafos conexos, sem *loops* e sem *arcos paralelos*. Um *caminho dirigido* em D é uma sequência de vértices de D sem repetição, $P = v_1, v_2, \ldots, v_k$, tal que $v_i \to v_{i+1}$ é um arco em D para $i = 1, \ldots, k-1$. Se existe um arco $v_k \to v_1$, então P mais o arco $v_k \to v_1$ é um *ciclo*. Se em um digrafo D não existem ciclos, então dizemos que D é *acíclico*. Um *subcaminho* de P do vértice v_i para o vértice v_j é denotado por $P[v_i, v_j]$, onde $i \leq j$. Se P é um caminho dirigido, então $P^{\mathcal{R}}$ denota o caminho P com os seus arcos *invertidos*. Em alguns casos, um único vértice é denotado como um caminho. A *concatenação* de dois caminhos

[1] Alguns autores preferem chamar um digrafo conexo de fracamente conexo.

$P = u_1, \ldots, u_i$ e $Q = u_i, \ldots, u_k$, denotado por PQ, é o caminho $PQ = u_1, \ldots, u_i, \ldots, u_k$, ou seja, um caminho de u_1 para u_k. Note que a concatenação PQ exige que o último vértice de P seja igual ao primeiro vértice de Q. Uma *ordenação topológica* de um digrafo acíclico D é uma ordenação dos vértices $x_1, \ldots, x_n$ de D tal que todo arco de D é da forma $x_i \to x_j$ com $i < j$. A ordem $x_1, \ldots, x_n$ é a *ordem topológica crescente* dos vértices, e a ordem $x_n, \ldots, x_1$ é a *ordem topológica decrescente*.

Dizemos que um grafo é *misto* se for formado por um conjunto de vértices, um conjunto de arcos e um conjunto de arestas. Um grafo misto H é acíclico se o digrafo *induzido* pelos arcos de H é acíclico, isto é, se o digrafo formado apenas pelos arcos de H e os vértices incidentes aos arcos é acíclico. Veja um grafo misto na Figura 2.1.

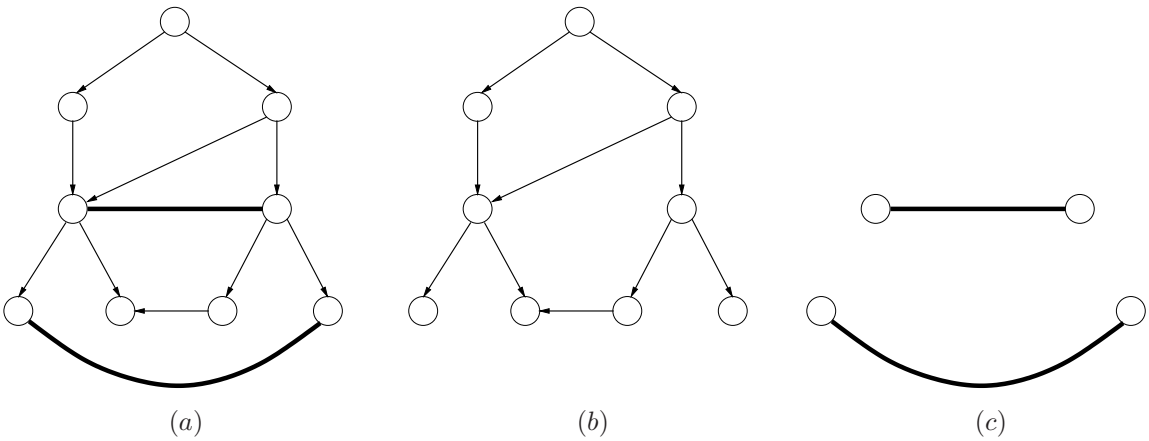


Figura 2.1: (a) - Um grafo misto G com duas arestas. (b) - O digrafo induzido pelos arcos de G. (c) - O grafo induzido pelas arestas de G.

Dados um digrafo acíclico D e dois vértices s e t em D, dizemos que s é um *pai* de t (ou que t é um *filho* de s) quando existe um arco $s \to t$ em D. O conjunto dos pais de um vértice u em D é denotado por $\delta_D^-(u)$ e o conjunto dos filhos por $\delta_D^+(u)$. O número de pais de um vértice u em D é denotado por $g_D^-(u)$ e o número de filhos por $g_D^+(u)$. Todo digrafo acíclico tem pelo menos um vértice com $g_D^-(u) = 0$ e outro com $g_D^+(u) = 0$. Um vértice u é um *vértice fonte* se $g_D^-(u) = 0$, enquanto que se $g_D^+(u) = 0$, então u é um *vértice sorvedouro*. Um digrafo D é uma *árvore enraizada* se existe um único vértice s com $g_D^-(s) = 0$ e o seu grafo correspondente é uma árvore. Chamamos s de *raiz* da árvore. A *altura* de um vértice v em uma árvore enraizada é o comprimento do maior caminho de v para uma *folha* da árvore.

Dado um digrafo D, o *digrafo linha* D_l de D é um digrafo onde cada arco $x \to y$ de D corresponde a um vértice xy de D_l e cada par de arcos $x \to y$ e $y \to z$ de D corresponde um arco $xy \to yz$ em D_l. Veja a Figura 2.2.

Dados um digrafo D e um subconjunto S do conjunto de vértices de D, o digrafo *induzido* por S é o digrafo formado pelos vértices de S mais os arcos que incidem somente em vértices de S. O digrafo induzido por S é denotado por $D[S]$. Por um instante, considere que S contém os vértices v tais que existe um caminho dirigido de s para v em D. Uma *árvore geradora de busca em profundidade* T^s sobre $D[S]$, abreviada por **agbp**, é uma árvore enraizada com raiz em s, o conjunto de vértices é igual ao conjunto de vértices de $D[S]$ e o conjunto de arcos é obtido por meio de uma *busca em profundidade* começando em s. Às vezes, trabalhamos sobre uma subárvore $T^{s'}$ de uma agbp T^s, onde s' é a raiz de $T^{s'}$ e s' é um vértice de S. Dada uma agbp T^s de $D[S]$, o conjunto de arcos de $D[S]$ pode ser particionado em:

1. *arcos da árvore* T^s;

2. *arcos de avanço*, ou seja, os arcos $u \to v$ de $D[S]$ que não estão em T^s porém existe um

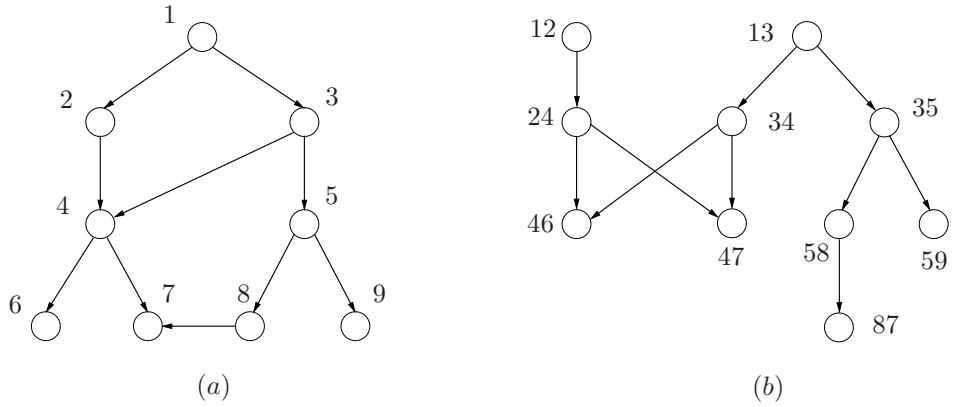


Figura 2.2: (a) - Um digrafo D. (b) - O digrafo linha de D.

caminho de u para v em T^s;

3. *arcos de retorno*, ou seja, os arcos $u \to v$ de $D[S]$ que não estão em T^s porém existe um caminho de v para u em T^s; e

4. *arcos cruzados*, ou seja, os arcos $u \to v$ de $D[S]$ que não estão em T^s e não existe em T^s um caminho de u para v e nem de v para u.

Tal partição aparece em vários trabalhos, como por exemplo, no livro de Aho, Lam, Sethi e Ullman [ALSU07]. Na Figura 2.3 temos um digrafo D, uma agbp de D com raiz em 1 e a correspondente partição dos arcos de D. Os arcos $3 \to 4$ e $8 \to 7$ são arcos cruzados; o arco $2 \to 6$ é um arco de avanço e o arco $9 \to 3$ é um arco de retorno.

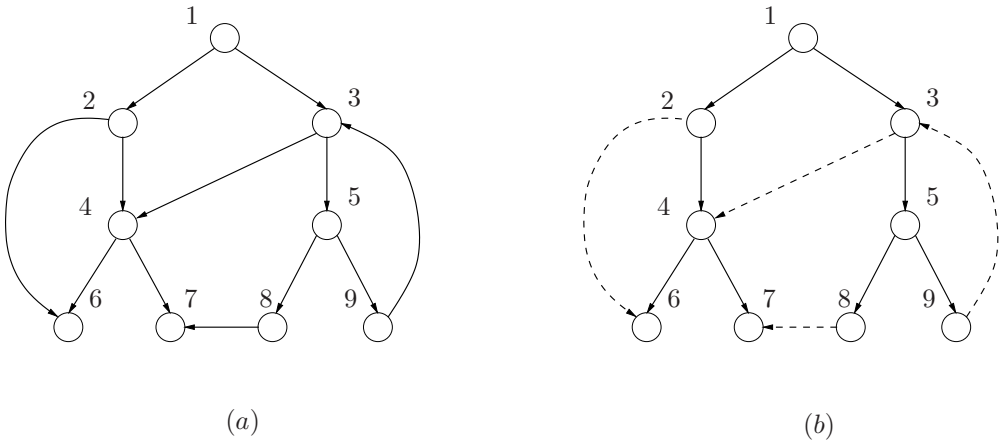


Figura 2.3: (a) - Um digrafo D. (b) - Uma partição dos arcos de D.

Um *grafo fluxo* D é um digrafo com um vértice s que alcança qualquer vértice v em D, isto é, existe em D um caminho dirigido de s para qualquer outro vértice de D. Chamamos s de raiz do grafo fluxo D. Um vértice w *domina* um outro vértice v se qualquer caminho dirigido de s para v passa por w. Um vértice w é o *dominador imediato* de um vértice v se w domina v, $w \neq v$, e qualquer dominador de v domina w. Para cada vértice $v \neq s$, existe um único vértice w que domina imediatamente v. Vamos denotar o dominador imediato de v por $di(v)$. Na Figura 2.4 (a), os vértices 1, 5, 7, 11, dominam o vértice 12, os vértices 1, 5 e 10 dominam o vértice 15. Os vértices 11 e 10 são, respectivamente, os dominadores imediatos dos vértices 12 ($di(12) = 11$) e 15 ($di(15) = 10$).

O digrafo T' cujo conjunto de vértices é igual ao conjunto de vértices de D e o conjunto de arcos é dado pelo seguinte conjunto $\{di(v) \to v \colon v$ está no conjunto de vértices de D e $v \neq s\}$ é uma árvore enraizada com raiz em s, chamada de *árvore de dominadores* de D. Veja a Figura 2.4 (b).

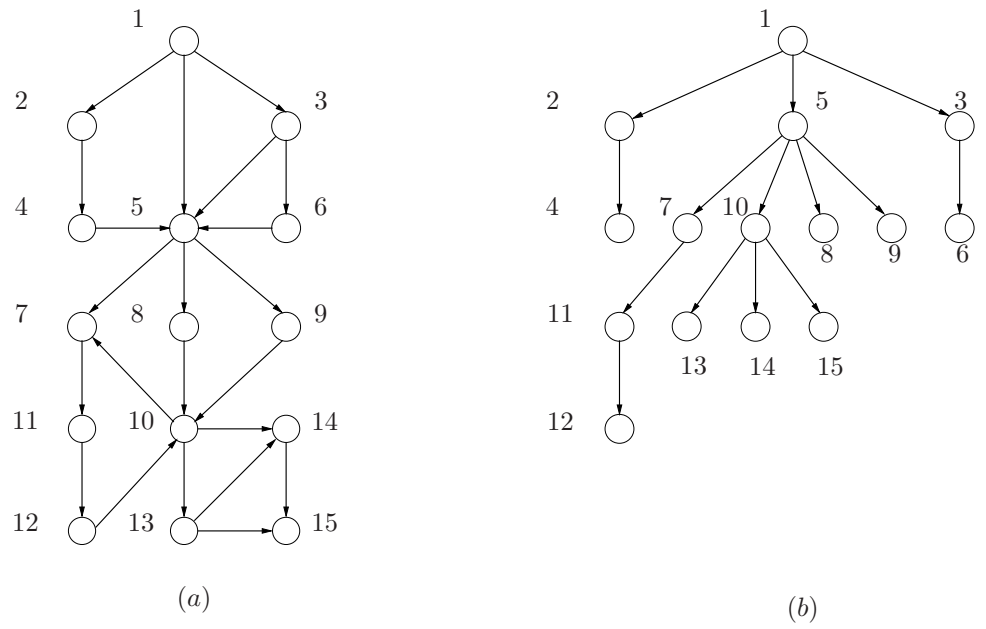


Figura 2.4: (a) - Um grafo fluxo G com raiz em 1. (b) - A árvore de dominadores de G.

Neste trabalho é apropriado definir um *grafo fluxo redutível*, abreviado por **gfr**, como um grafo fluxo que possui um único *subgrafo acíclico maximal* [HU74, Teorema 5]. Dado um grafo fluxo D, considere uma árvore geradora de busca em profundidade de D e a partição dos arcos de D determinada por essa árvore. Um subgrafo acíclico maximal de D é obtido removendo os arcos de retorno de tal partição. Se os arcos de retorno são iguais para qualquer agbp de D, então, ao remover os arcos de retorno, o resultado é um subgrafo acíclico maximal único. Veja a Figura 2.5.

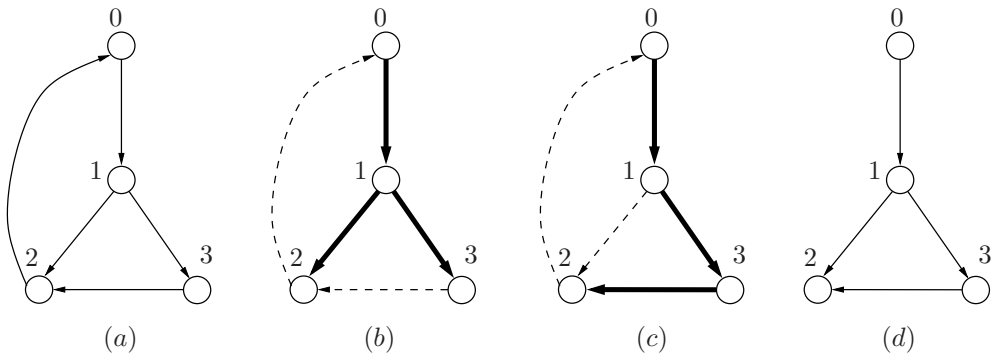


Figura 2.5: (a) - Um grafo fluxo redutível D com raiz em 0. (b), (c) - as duas agbps de D com raiz em s. (d) - o subgrafo acíclico maximal de D.

Um resultado importante devido a Hecht e Ullman em [HU74, Teorema 6], nos diz que a relação de dominância é preservada entre um grafo fluxo redutível e o seu subgrafo acíclico maximal. Portanto, para construir uma árvore de dominadores de um grafo fluxo redutível basta considerar o seu subgrafo acíclico maximal.

2.2 Matrizes e o Fecho Transitivo

Dadas duas *matrizes booleanas* A $(n \times n)$ e B $(n \times n)$, a multiplicação AB é uma nova matriz booleana C tal que $C(i,j) = 1$ se e somente se existe k em $\{1, \ldots, n\}$ tal que $A(i,k) = B(k,j) = 1$. Uma *matriz de testemunhas* W sobre $\{0, 1, \ldots, n\}$ para C é uma matriz que certifica as entradas de C, isto é, se $C(i,j) = 1$, então existe k em $\{1, \ldots, n\}$ tal que $A(i,k) = B(k,j) = 1$. Assim, a entrada $W(i,j) = k$. Se $C(i,j) = 0$, então $W(i,j) = 0$.

Dado um digrafo D, a *matriz de adjacência* M de D é uma matriz booleana $(n \times n)$ indexada pelos vértices de D, onde $M(v_i, v_j) = 1$ se e somente se $v_i \to v_j$ é um arco em D. O *fecho transitivo* D_{ft} de D é o digrafo cujo conjunto de vértices é igual ao conjunto de vértices de D e um arco $u \to v$ está em D_{ft} se e somente se existe um caminho de u para v em D. Os algoritmos de Furman [Fur70] e Munro [Mun71] obtém o fecho transitivo de D usando a matriz de adjacência M e um algoritmo que multiplica matrizes quadradas em tempo $\tilde{O}(n^\omega)$, onde $O(n^\omega)$ é o tempo do algoritmo mais rápido que multiplica duas matrizes quadradas $(n \times n)$.

Note a ocorrência da letra ω no consumo de tempo do algoritmo mais rápido que multiplica matrizes quadradas. De forma mais geral, o *expoente da multiplicação de matrizes retangulares* $\omega(1,1,k)$ é assim definido: para todo $k > 0$, $\omega(1,1,k) = \inf\{\tau \in \mathbb{R} \mid C(n, n, \lfloor n^k \rfloor) = O(n^\tau)\}$, onde $C(n, n, \lfloor n^k \rfloor)$ é essencialmente o tempo gasto para multiplicar duas matrizes $(n \times \lfloor n^k \rfloor)$ e $(\lfloor n^k \rfloor \times n)$ [LG12]. Para a multiplicação de matrizes quadradas temos $\omega = \omega(1,1,1)$. Atualmente ω é limitado superiormente por $2,373$, resultado devido a Williams [Wil12]. Um outro valor importante que aparece na literatura de multiplicação de matrizes é $\alpha = \sup\{k \mid \omega(1,1,k) = 2\}$. Como descrito em [LG12], o valor α representa o maior valor tal que o produto de matrizes retangulares $(n \times n^\alpha)$ e $(n^\alpha \times n)$ pode ser feito em tempo $O(n^{2+\varepsilon})$ para qualquer constante $\varepsilon > 0$. Note que o tempo para multiplicar duas matrizes quadradas é $\Omega(n^2)$. Assim, $\omega = 2$ se e somente se $\alpha = 1$. Uma maneira de mostrar que $\omega = 2$ é obter limitantes inferiores para α. Atualmente, $\alpha \geq 0,30298$. O leitor interessado pode consultar o trabalho de Le Gall [LG12].

2.3 Junções e Ancestrais

Considere um digrafo D e vértices s, u e v de D. Dizemos que o vértice s é *origem comum* aos vértices u e v se existem caminhos de s para u e de s para v. O vértice s é uma *junção* dos vértices u e v se s é origem comum aos vértices u e v e existem caminhos internamente *vértice-disjuntos* de s para u e de s para v. O vértice s é um *arco-junção* dos vértices u e v se s é origem comum aos vértices u e v e existem caminhos *arco-disjuntos* de s para u e de s para v.

Quando D é acíclico, dizemos que s é um *ancestral* de t (t é um *descendente* de s) se existe um caminho dirigido de s para t em D. Se s é um ancestral de t (t é um descendente de s) e $s \neq t$, então s é um *ancestral próprio* de t (t é um *descendente próprio* de s).

Um vértice s de um digrafo D é um *ancestral comum mais próximo* dos vértices u e v, abreviado por **acp**, se s é uma junção e não existe caminho em D de s para outro vértice s', onde s' é um vértice origem comum (ou simplesmente comum) aos vértices u e v.[2] Se D é uma árvore enraizada, então existe um único acp (ou uma única junção) para qualquer par de vértices u e v. Por outro lado, em um digrafo geral podem existir diferentes acps ou junções para um determinado par de vértices. Note que, em um digrafo que não seja uma árvore enraizada, um acp de u e v é uma junção de u e v porém o contrário não é sempre verdade.

Dados dois vértices u e v de D, o conjunto das junções de u e v é denotado por $\mathcal{J}_{u,v}$. Os conjuntos $\mathcal{J}_{u,v}$ e $\mathcal{J}_{v,u}$ são iguais. Escrevemos $s \in \mathcal{J}_{u,v}$ se s é uma junção dos vértices u e v. Consideramos que s é uma junção dos vértices s e u se existe um caminho dirigido de s para u. Tais junções são chamadas de *degeneradas*. Analogamente, denotamos por $\mathcal{AJ}_{u,v}$ o conjunto

[2] Pela nossa definição de um **ancestral** comum mais próximo, é possível que tal objeto ocorra em um digrafo geral, não necessariamente acíclico. Um outro nome para esse objeto é **vértice** origem comum mais próximo. Neste trabalho, vamos usar o nome ancestral comum mais próximo para qualquer caso.

dos arcos-junções de u e v e $\mathcal{ACP}_{u,v}$ o conjunto dos acps dos vértices u e v. Escrevemos $\mathcal{ACP}^D_{u,v}$ quando é importante destacar o digrafo D para o qual estamos considerando o conjunto dos acps de u e v. Quando D é uma árvore enraizada, abusamos da linguagem escrevendo $s = \mathcal{ACP}^D_{u,v}$. Veja alguns exemplos de acps e junções na Figura 2.6. Imagine que a direção em cada arco das árvores é sempre de cima para baixo.

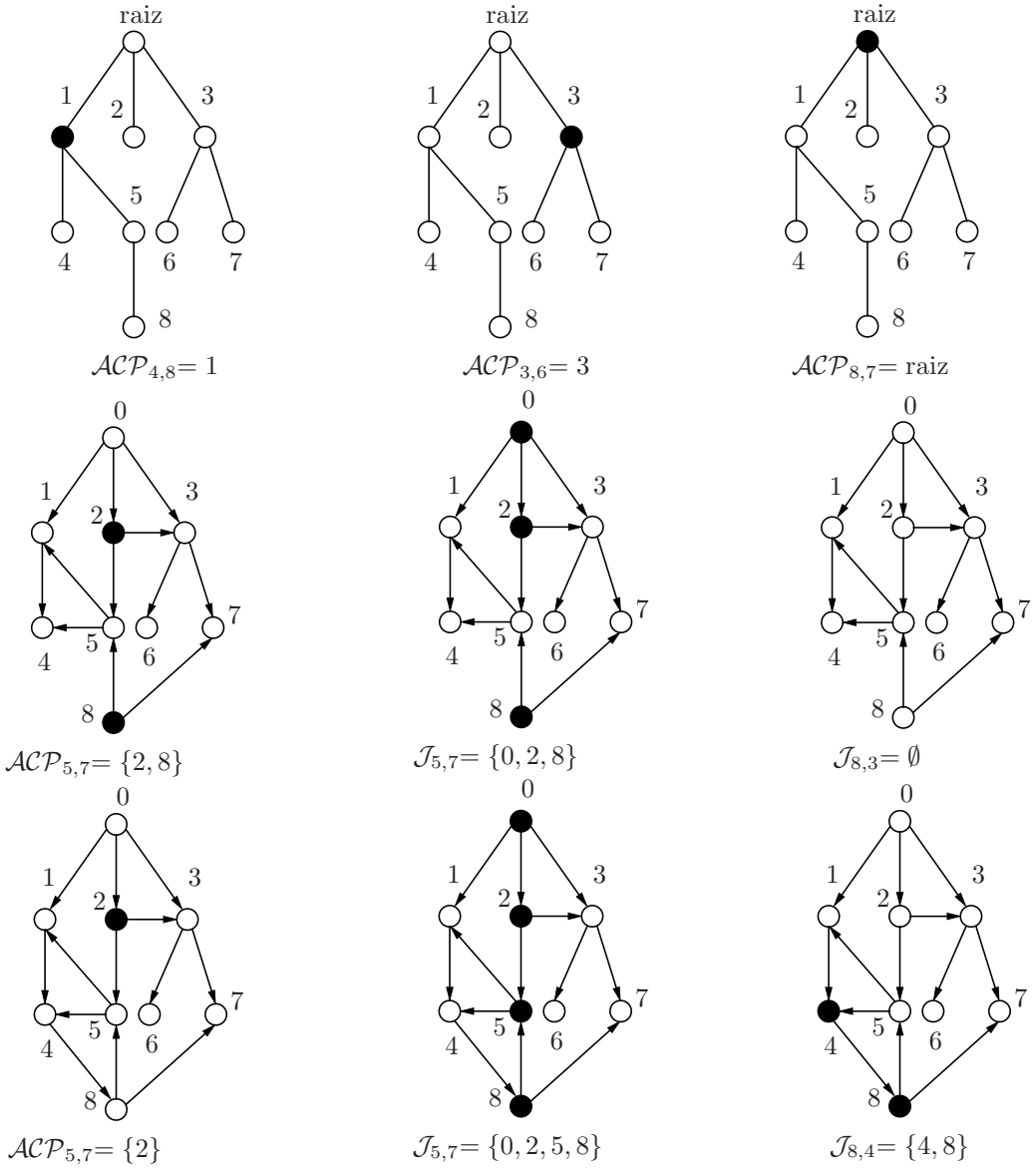


Figura 2.6: Os vértices marcados são acps e junções de alguns pares de vértices em árvores enraizadas, digrafos acíclicos e digrafos gerais.

2.4 Anéis em Grafos Mistos

Dado um grafo misto H, queremos encontrar em H uma estrutura que relaciona caminhos dirigidos vértice-disjuntos com arestas. Essa estrutura é chamada de *anel*. Um anel de tamanho k (ou k-*anel*) em H é uma estrutura com exatamente k arestas e $2k$ caminhos dirigidos e internamente vértice-disjuntos. Por exemplo, um 1-anel ocorre quando temos uma aresta $u - v$ de H e existe uma junção s dos vértices u e v (veja Figura 2.7 (a)). Em um 2-anel temos duas arestas $u_1 - v_2$ e $u_2 - v_1$ e existem uma junção s_1 dos vértices u_1 e v_1, uma junção s_2

dos vértices u_2 e v_2, $s_1 \neq s_2$ e todos os caminhos dirigidos desta estrutura são internamente vértice-disjuntos (veja Figura 2.7 (b)). Formalmente um k-anel é um par de conjuntos $(\mathcal{E}, \mathcal{P})$ onde $\mathcal{E} = (u_1 - v_k, u_2 - v_1, \ldots, u_k - v_{k-1})$ é um subconjunto ordenado do conjunto de arestas de H e $\mathcal{P} = (P_1^{\mathcal{R}} Q_1, \ldots, P_k^{\mathcal{R}} Q_k)$ é um conjunto ordenado de k caminhos dirigidos atendendo às seguintes restrições para $i = 1, \ldots, k$:

1. $P_i^{\mathcal{R}}$ é um caminho (invertido) de u_i para s_i,
2. Q_i é um caminho de s_i para v_i,
3. todos os caminhos são dois a dois vértice-disjuntos exceto para os caminhos com um mesmo índice i que são internamente vértice-disjuntos.

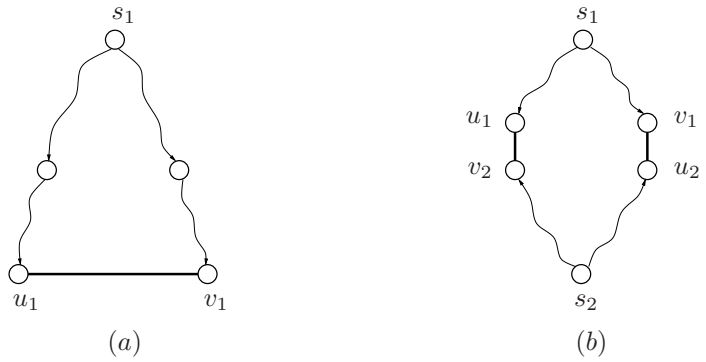


Figura 2.7: a - Um 1-anel. b - Um 2-anel.

A Figura 2.8 (a) ilustra um k-anel em um grafo misto H. Uma observação importante é que o vértice s_i é uma junção dos vértices u_i e v_i para todo $i = 1, \ldots, k$. Dado um k-anel A, se existe em A uma junção degenerada s_i dos vértices u_i e v_i para algum $i = 1, \ldots, k$, então dizemos que A é um *anel degenerado* (veja Figura 2.8 (b)).

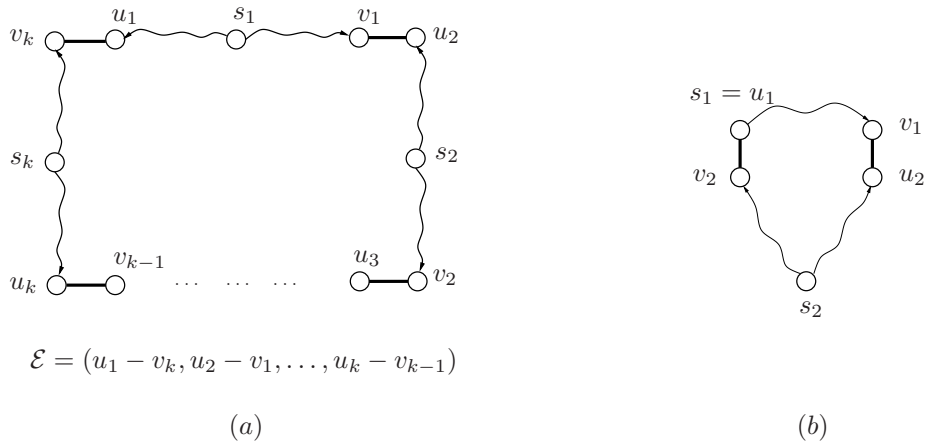


Figura 2.8: (a) - Um k-anel. (b) - Um 2-anel degenerado.

2.5 Algoritmos com Fases de Pré-processamento e Consultas

Vemos na literatura muitos problemas serem resolvidos em duas fases. Em uma primeira fase, chamada de *pré-processamento*, a entrada do problema é processada preparando uma *estrutura de dados* para a segunda fase. A segunda fase consiste de uma sequência de perguntas. Para

resolver cada pergunta podemos consultar a estrutura de dados criada na primeira fase. Como exemplo, citamos o seguinte problema.

> O problema *consulta-janela*: Dado um conjunto de pontos no plano, prepare uma estrutura de dados para responder rapidamente a uma sequência de perguntas. Em cada pergunta desejamos saber quais são os pontos dados inicialmente que estão em um retângulo (ou janela) definido na pergunta. Tal sequência de perguntas não é conhecida *a priori*.

Normalmente uma pergunta é "sinônimo" de *consulta* e é comum vermos na literatura a segunda fase como uma sequência de consultas.

Nos problemas tratados neste trabalho, é dado pelo menos um digrafo acíclico D. Uma sequência de consultas pode ou não ser dada. A natureza de uma consulta varia de acordo com o problema tratado. Por exemplo, uma consulta *representante-junção*(u, v) deseja saber uma junção em D para os vértices u e v. Uma consulta *todas-junções*(u, v) deseja saber todas as junções em D para os vértices u e v. Da mesma forma, uma consulta *representante-acp*(u, v) deseja saber um acp em D para os vértices u e v. Uma consulta *todos-acps*(u, v) deseja saber todos os acps em D para os vértices u e v. As consultas devolvem um ou mais vértices em D (acps ou junções). Quando o digrafo D é uma árvore enraizada, uma consulta *acp*(u, v) devolve o único acp de u e v em D. As consultas *s-acp*(u, v) e *s-junção*(u, v) são consultas especiais. Nestes casos são dados um digrafo acíclico D e um vértice s e as consultas desejam saber se s é respectivamente um acp ou uma junção dos vértices u e v. Temos também a consulta *s-arco-junção*(u, v) que deseja saber se s é um arco-junção dos vértices u e v. A Figura 2.9 ilustra algumas consultas citadas.

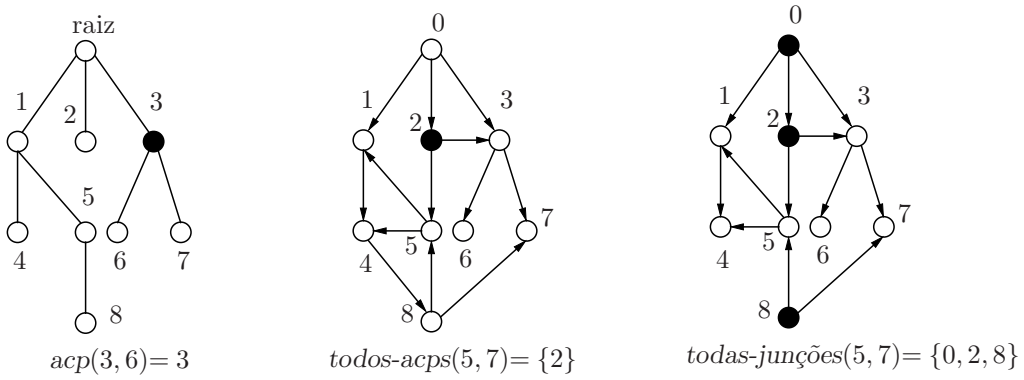


Figura 2.9: Diferentes consultas em diferentes digrafos.

2.6 Aspectos de Complexidade Computacional

O custo de cada passo de um algoritmo descrito neste trabalho está de acordo com o custo atribuído por Cormen et al. [CLRS09]. Discutimos a *complexidade computacional* clássica de alguns problemas tratados neste trabalho, dizendo quais pertencem às classes $\mathcal{P}$, isto é, a classe dos problemas decididos em tempo polinomial por uma *máquina de Turing determinística*, ou $\mathcal{NP}$, isto é, a classe dos problemas decididos em tempo polinomial por uma *máquina de Turing não-determinística*. Mostramos que alguns problemas são completos para a classe $\mathcal{NP}$ (problemas $\mathcal{NP}$-completos). Não descrevemos as definições e os teoremas dessa teoria e esperamos do leitor um conhecimento básico sobre o assunto. Indicamos o livro do Sipser [Sip96] para uma primeira leitura.

Também discutimos alguns aspectos da teoria da *complexidade parametrizada*. Os artigos de Downey e Fellow [DF95a] e [DF95b], o livro de Flum e Grohe [FG06] e o artigo de Slivkins [Sli03] foram as principais referências consultadas.

As definições e os resultados que aparecem neste texto estão relacionados a três assuntos fundamentais dessa teoria: os problemas parametrizados, a p-redução entre problemas parametrizados, e as classes $\mathcal{FPT}$ e $\mathcal{W}[1]$. As definições aparecem de maneira tão informal quanto possível para manter a precisão. Para uma discussão completa recomendamos o livro citado.

2.6.1 Problemas Parametrizados

Começamos com uma definição para *problema parametrizado*. Um problema P é parametrizado por um número natural k se qualquer entrada de P é um par (x, k). Podemos entender x como a entrada do problema e k é um número natural. Vamos exemplificar um problema parametrizado através de um problema importante para este trabalho.

> O problema *k-caminhos-vértice-disjuntos*: Dados um digrafo D e pares de vértices $(s_1, t_1), \ldots, (s_k, t_k)$ de D (o parâmetro k coincide com o número de pares de vértices dados), decidir se existem caminhos vértice-disjuntos P_i de s_i para t_i para $i = 1, \ldots, k$.

Em uma entrada (x, k) do problema *k-caminhos-vértice-disjuntos* a instância x é formada por um digrafo e uma lista de pares ordenados de vértices, e o parâmetro k é o tamanho desta lista. Como descrito anteriormente, desejamos saber se existem caminhos vértice-disjuntos conectando cada par da lista. Se tais caminhos existem para um entrada (x, k), então dizemos que (x, k) é uma *entrada válida* do problema P.

2.6.2 Redução entre Problema Parametrizados

Uma p-redução de um problema parametrizado P para um outro P' deve atender aos seguintes requisitos:

1. Dada uma entrada (x, k) para o problema P, transformar (x, k) em uma entrada (x', k') para o problema P' através de uma computação R, isto é, $R(x, k) = (x', k')$, de tal forma que (x, k) é uma entrada válida do problema P se e somente se (x', k') é uma entrada válida para o problema P'. A transformação deve ser feita em tempo $O(f(k)|x|^c)$, onde f é qualquer função computável, c é uma constante, e $|x|$ denota o tamanho da entrada; e

2. Dadas uma entrada (x, k) para o problema P e uma computação R tal que $R(x, k) = (x', k')$, existe uma função computável g tal que $k' \leq g(k)$.

2.6.3 A Classe $\mathcal{FPT}$

Dizemos que um problema parametrizado P é tratável quando k é fixo (FPT - Tratável com Parâmetro Fixado) se podemos decidir se uma entrada qualquer (x, k) para P é ou não uma entrada válida para P em tempo $O(f(k)|x|^c)$, onde f é uma função computável e c é uma constante. Assim, se k for razoavelmente pequeno, podemos decidir se (x, k) é uma entrada válida para P em tempo "polinomial". Para este caso, dizemos que o problema P está na classe $\mathcal{FPT}$. É interessante notar o seguinte. Se um problema parametrizado P com parâmetro k está em $\mathcal{FPT}$, então P com k fixo pode ser decidido em tempo polinomial. Porém, se P é $\mathcal{NP}$-completo para k fixo, então P não pode estar em $\mathcal{FPT}$ a menos que $\mathcal{P} = \mathcal{NP}$. Um problema parametrizado que provavelmente não está em $\mathcal{FPT}$ é o problema *k-caminhos-vértice-disjuntos*. Este problema é $\mathcal{NP}$-completo até mesmo quando $k = 2$ [FHW80]. Portanto, se ele estivesse em $\mathcal{FPT}$, teríamos $\mathcal{P} = \mathcal{NP}$.

2.6.4 As Classes $\mathcal{W}[P], \mathcal{W}[1], \mathcal{W}[2], \ldots$

Começamos com a definição de um problema sobre *circuitos booleanos*. Um circuito booleano pode ser representado usando um digrafo acíclico. O vértice com grau de saída igual a zero é chamado de *vértice de saída*. Se o grau de entrada de um vértice v é igual a zero ($\delta^-(v) = 0$), então v é chamado de *vértice de entrada*. Se o grau de entrada de v é igual a um ($\delta^-(v) = 1$), então v é chamado de *vértice* NÃO. Se o grau de entrada de v é maior ou igual que dois ($\delta^-(v) \geq 2$), então v é chamado de *vértice* E ou *vértice* OU. Se o grau de entrada de v é maior que dois ($\delta^-(v) > 2$), então v é chamado de *vértice irrestrito* E ou *vértice irrestrito* OU. Veja a Figura 2.10.

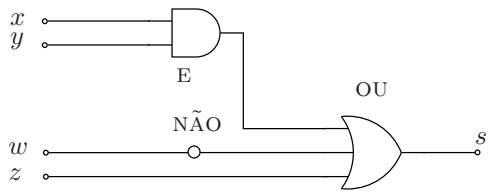


Figura 2.10: Um circuito booleano (ou lógico). Os vértices x, y e z são os vértices de entrada. O vértice s é o vértice de saída. O vértice OU é irrestrito.

A *profundidade de um circuito* é definida como sendo o número máximo de vértices NÃO, OU e E em um caminho de um vértice de entrada para o vértice de saída. A *teia (weft) de um circuito* é definida como sendo o número máximo de vértices irrestritos OU e E em um caminho de um vértice de entrada para o vértice de saída.

Dizemos que C é k-satisfatível se o vértice de saída de C possui valor 1 depois da simulação de C com exatamente k vértices de entrada com valor 1. Assim, temos o seguinte problema de decisão parametrizado.

> O problema *circuito-k-satisfatível*: Dados um circuito booleano C e um número natural k, decidir se C é k-satisfatível.

A classe $\mathcal{W}[P]$ foi definida por Downey e Fellows [DF95a] como sendo a classe dos problemas parametrizados p-redutíveis ao problema *circuito-k-satisfatível* [FG06, Notas do Capítulo 3]. Quando o circuito de entrada do problema *circuito-k-satisfatível* tem profundidade e teia limitados superiormente por dois inteiros positivos, respectivamente, h e t, os problemas parametrizados p-redutíveis ao problema *circuito-k-satisfatível* com tais limitações estão na classe $\mathcal{W}[t]$. As restrições mais fortes estão sobre $\mathcal{W}[1]$ e vão gradativamente enfraquecendo para $\mathcal{W}[2]$, $\mathcal{W}[3]$, ... As classes $\mathcal{W}[t]$ para todo $t \geq 1$ e $W[P]$ formam uma hierarquia chamada de $\mathcal{W}$-hierarquia. A Figura 2.11 ilustra como essas classes devem estar relacionadas.

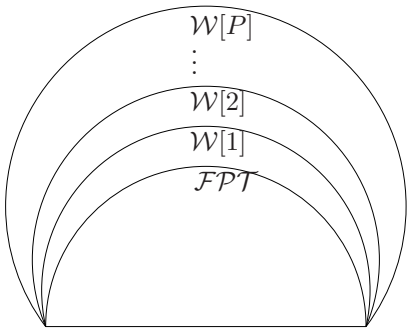


Figura 2.11: Relações entre as classes $\mathcal{FPT}, \mathcal{W}[1], \mathcal{W}[2], \ldots, \mathcal{W}[P]$ [FG06].

Da mesma forma que um problema $\mathcal{NP}$-completo provavelmente não está em $\mathcal{P}$, um problema parametrizado $\mathcal{W}[1]$-completo provavelmente não está em $\mathcal{FPT}$. Quando todos os problemas em $\mathcal{NP}$ são redutíveis a um problema P, dizemos que P é $\mathcal{NP}$-difícil. Analogamente, quando todos os problemas em $\mathcal{W}[1]$ são p-redutíveis a P, dizemos que P é $\mathcal{W}[1]$-difícil. Um exemplo de um problema $\mathcal{W}[1]$-difícil é o problema *k-caminhos-vértice-disjuntos* restrito a digrafos acíclicos [Sli03], [BJG08].

Indicamos as referências [DF95a], [DF95b] e [FG06] para o leitor que quiser saber mais sobre a teoria da complexidade parametrizada.

3

Algoritmos para Junções em Digrafos - Revisão da Literatura

Apresentamos neste capítulo alguns algoritmos que pré-processam um dado digrafo e respondem consultas tais como *acp*(u, v), *representante-acp*(u, v), *todos-acps*(u, v), *s-acp*(u, v), *representante-junção*(u, v), *todas-junções*(u, v) e *s-junção*(u, v). Em todos os trabalhos apresentados nesta resenha tanto os pré-processamentos como as consultas são realizados de maneira eficiente. Na Seção 3.1, definimos o problema *acp-todos-pares* e descrevemos três algoritmos eficientes para resolvê-lo. Nas Seções 3.2, 3.3, 3.4 e 3.5, definimos respectivamente os problemas: *representante-acp-todos-pares*, *todos-acps-todos-pares*, *representante-junção-todos-pares* e *todas-junções-todos-pares*. Em cada seção, descrevemos os principais algoritmos que resolvem o correspondente problema. Os algoritmos da literatura descritos na Seção 3.3 supõem um digrafo acíclico na entrada. No entanto, notamos que, para alguns algoritmos, essa restrição pode ser desconsiderada. Destacamos a nossa contribuição para uma versão mais fraca de cada um dos problemas citados, quando a entrada é composta por um digrafo acíclico e, adicionalmente, as correspondentes consultas. Estes problemas são: *representante-acp-k-pares*, *todos-acps-k-pares*, *representante-junção-k-pares* e *todas-junções-k-pares*. A entrada dos problemas que pedem por junções para todos os pares de vértices é constituída por apenas um digrafo, enquanto que a entrada dos problemas que pedem por junções para k pares de vértices é constituída por um digrafo e por uma lista de consultas. Isso deixa os problemas com natureza diferente e a comparação dos algoritmos é feita limitando o tamanho da entrada (o digrafo e/ou a lista de consultas). Os algoritmos para os problemas que pedem por junções para todos os pares de vértices consideram poder acessar qualquer vértice do digrafo em qualquer momento. Por isso, não é possível adaptar os algoritmos para a versão do problema que pedem por junções para k pares de vértices, a menos que rode o algoritmo original e responda somente as k consultas. Finalmente, na Seção 3.6 descrevemos um algoritmo da literatura que constrói a árvore de dominadores de um grafo fluxo redutível.

3.1 Ancestral Comum mais Próximo em Árvores Enraizadas

O problema tratado nesta seção é o problema *acp-todos-pares* em árvores enraizadas.

> O problema *acp-todos-pares*: Dada uma árvore enraizada T, pré-processe T de tal forma que uma consulta qualquer *acp*(u, v) seja respondida.

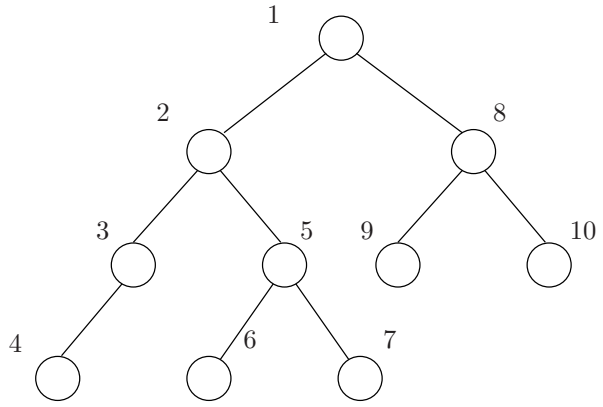
Existe uma extensa lista de trabalhos sobre este assunto. Citamos por exemplo os trabalhos de Aho, Hopcroft e Ullman [AHU76], de Harel e Tarjan [HT84], de Berkman e Vishkin [BV94], de Cole e Hariharan [CH05], de Nykänen e Ukkonen [NU94] e de Wen [Wen94]. Neste texto, daremos ênfase a três trabalhos, dois clássicos e um mais recente. O algoritmo de Aho, Hopcroft e Ullman [AHU76] considera que uma lista de consultas também é dada na entrada. Os algoritmos de Harel e Tarjan [HT84], e o algoritmo de Berkman e Vishkin [BV94] resolvem o problema *acp-todos-pares* como enunciado acima.

3.1.1 Algoritmo de Aho, Hopcroft e Ullman

Dadas uma árvore T com n vértices e uma lista L com $O(n)$ consultas $acp(u, v)$, o algoritmo de Aho, Hopcroft e Ullman [AHU76] constrói em tempo $O(n\alpha(n))$, estruturas de dados capazes de responder em tempo constante cada consulta de L, onde α é uma função inversa de Ackermann. Vamos descrever o algoritmo em seguida.

Cada vértice v de T armazena o valor pré-ordem de v, $pré(v)$, obtido através de uma varredura da árvore nesta ordem. Se i é o valor pré-ordem de um vértice v, então escrevemos $pré^{-1}(i) = v$. Cada consulta $acp(u, v)$ está associada a um par ordenado $X = (i, j)$ tal que $i = pré(u)$, $j = pré(v)$ e $i < j$. Para cada par ordenado X, o primeiro elemento do par é denotado por $esq(X)$ e o segundo por $dir(X)$. Primeiramente, o algoritmo gera uma sequência de instruções $A(X)$ e $R(i)$ ($Adicione(X)$ e $Remova(i)$), onde X é um par ordenado e i é o valor pré-ordem de um vértice. Depois, o algoritmo simula a sequência de instruções usando operações para conjuntos disjuntos[1] [CLRS09].

Considere a árvore enraizada e a lista de consultas na Figura 3.1. A geração da sequência de instruções é feita processando os vértices na ordem dada pela varredura em pós-ordem, isto é, na ordem 4, 3, 6, 7, 5, 2, 9, 10, 8, 1. No vértice com valor pré-ordem i o algoritmo cria instruções $A(X)$ para cada X tal que $dir(X) = i$ e uma instrução $R(i)$. Para a árvore enraizada da Figura 3.1, com a lista de consultas dada, teremos o seguinte conjunto de instruções: $\{R(4)$, $R(3)$, $A(X_1)$, $A(X_4)$, $R(6)$, $A(X_2)$, $A(X_3)$, $R(7)$, $A(X_7)$, $R(5)$, $R(2)$, $R(9)$, $A(X_6)$, $R(10)$, $A(X_5)$, $R(8)$, $R(1)\}$. Podemos imaginar que os pares X que foram adicionados ao conjunto ainda não foram tratados, e uma instrução $R(i)$ remove deste conjunto todos os pares X com $esq(X) \geq i$. Em [AHU76] é demonstrado que encontramos o ancestral comum mais próximo de u e v no momento em que o par $X = (pré(u), pré(v))$ é removido por uma instrução $R(i)$. O vértice $w = pré^{-1}(i)$ é o ancestral comum mais próximo dos vértices u e v.



$X_1 = (4, 6)$ $X_2 = (5, 7)$ $X_3 = (3, 7)$ $X_4 = (5, 6)$

$X_5 = (7, 8)$ $X_6 = (9, 10)$ $X_7 = (2, 5)$

Figura 3.1: Uma árvore enraizada e uma lista de consultas

A simulação da sequência de instruções requer alguns comentários. São criados conjuntos S_i cujos elementos são instruções A e R. No início:

1. S_i contém as instruções $A(X)$ tais que $dir(X) = i$; e

2. S_j contém $R(i)$ se $R(j)$ é a próxima instrução R depois da instrução $R(i)$.

[1] No artigo original, Aho, Hopcroft e Ullman usam o algoritmo *Set Merging* [HU73].

Cada vértice $u = pré^{-1}(i)$ é processado na ordem decrescente dos valores pré-ordem. Para o exemplo na Figura 3.1 temos o processamento dos vértices na ordem $10, 9, \ldots, 1$. Note que isso implica em um certo processamento das folhas para a raiz e dos vértices que estão mais à direita para os vértices que estão mais à esquerda. Para um i fixo, são analisados todos os pares X tais que $esq(X) = i$. Para cada X, uma atribuição $S_j \leftarrow encontre(A(X))$ é feita, e enquanto $i < j$ as seguintes atribuições são feitas: $S_h \leftarrow encontre(R(j))$, $S_h \leftarrow união(S_h, S_j)$ e $j \leftarrow h$. Por exemplo, quando $i = 9$, $X_6 = (9, 10)$ é analisado. Para este caso, $j = 10$, $h = 8$, o novo $S_8 = \{A(X_5),\ R(1),\ A(X_6),\ R(8)\}$ (inicialmente $S_8 = \{A(X_5),\ R(1)\}$ e $S_{10} = \{A(X_6),\ R(8)\}$) e o novo $j = 8$. Com isso, $i \geq j$ e o laço para. Note que na última iteração do laço encontramos $R(8)$ e que $esq(X_6) \geq 8$. Logo, o vértice $pré^{-1}(8)$ é o ancestral comum mais próximo dos vértices $pré^{-1}(9)$ e $pré^{-1}(10)$.

Uma importante observação é que ao processar um par ordenado $X = (i, k)$, temos que $i = esq(X)$ e que o vértice $u = pré^{-1}(i)$ corresponde ao vértice com maior valor pré-ordem ainda não processado. Certamente, o vértice $v = pré^{-1}(k)$ não é um ancestral de u. Ou v está à direita de u, ou v é um descendente de u. Considerando o conjunto S_j que contém $A(X)$ e que $i < j$, o vértice $w = pré^{-1}(j)$ ou está à direita ou é descendente de u. Quando $i \geq j$, o vértice w atual é um ancestral comum de u e v. No último caso temos o ancestral comum mais próximo dos vértices u e v. Resultados de [AHU76] garantem que o algoritmo está correto e que seu consumo de tempo é $O(n\alpha(n))$, onde α é uma função inversa de Ackermann. A criação do conjunto de instruções em pós-ordem e o item 2 descrito anteriormente são informações importantes para perceber que o algoritmo está certo.

3.1.2 Algoritmo de Harel e Tarjan

O algoritmo de Harel e Tarjan [HT84] resolve o problema *acp-todo-pares*. O pré-processamento da árvore enraizada T com n vértices é feito em tempo $O(n)$. Qualquer consulta $acp(u, v)$ é respondida em tempo $O(1)$. Os autores começam descrevendo o algoritmo considerando que T é uma árvore binária completa. Neste caso, o pré-processamento consiste na construção de três vetores: in, in^{-1} e h. O valor $in(v)$ é o valor do vértice v na varredura in-ordem de T. Se i é o valor de v, então $in^{-1}(i) = v$. O valor $h(v)$ é a altura do vértice v em T. Depois desse pré-processamento em T (uma árvore binária completa), o algoritmo é capaz de responder a qualquer consulta $acp(u, v)$ em tempo $O(1)$. O seguinte resultado garante a resposta de uma consulta $acp(u, v)$ em tempo $O(1)$ caso u seja um descendente de v ou vice-versa.

Lema 3.1 (Harel e Tarjan [HT84]). *Os descendentes de um vértice w de T são os vértices cujos valores in-ordem pertencem ao intervalo fechado $[in(w) - 2^{h(w)} + 1 : in(w) + 2^{h(w)} - 1]$.* □

Se u não é descendente e nem ancestral de v, isto é, u e v não se relacionam diretamente, então o seguinte resultado encontra a altura do ancestral comum mais próximo de u e v em tempo $O(1)$.

Lema 3.2 (Harel e Tarjan [HT84]). *Se u e v são dois vértices de T não relacionados diretamente, a altura do ancestral comum mais próximo de u e v é $\lfloor \lg(in(u) \oplus in(v)) \rfloor$, onde $i \oplus j$ é o inteiro cuja representação binária é o resultado da operação booleana ou exclusivo da representação binária de i e j.* □

Dada a altura h do ancestral comum mais próximo de u e v, o seguinte resultado encontra o ancestral de u (e de v) cuja altura é h em tempo $O(1)$.

Lema 3.3 (Harel e Tarjan [HT84]). *Se v é um vértice de T e h é uma altura tal que $h \geq h(v)$, então o ancestral de v de altura h é o vértice $in^{-1}(2^{h+1}\lfloor in(v)/2^{h+1} \rfloor + 2^h)$.* □

Assim, uma consulta $acp(u, v)$ é respondida em tempo $O(1)$ sobre uma árvore binária completa da seguinte forma. Se $in(u)$ está em $[in(v) - 2^{h(v)} + 1 : in(v) + 2^{h(v)} - 1]$, devolva v. Se

$in(v)$ está em $[in(u) - 2^{h(u)} + 1 : in(u) + 2^{h(u)} - 1]$, devolva u. Se os testes anteriores falharam, então faça $h = \lfloor \lg(in(u) \oplus in(v)) \rfloor$ e devolva $in^{-1}(2^{h+1} \lfloor in(v)/2^{h+1} \rfloor + 2^h)$.

Harel e Tarjan resolvem o problema *acp-todos-pares* em qualquer árvore enraizada T. A ideia é transformar uma consulta $acp(u, v)$ sobre T em uma consulta $acp(u, v)$ sobre uma subárvore binária completa com $O(n)$ vértices. Um número constante de passos no pré-processamento são necessários. Em cada um deles é consumido tempo linear. Não é uma tarefa fácil descrever este novo pré-processamento. Para o leitor interessado, indicamos a leitura das Seções 3, 4 e 5 de [HT84].

3.1.3 Algoritmo de Berkman e Vishkin

Descrevemos nesta seção o algoritmo de Berkman e Vishkin [BV93]. A nossa descrição é baseada na descrição feita por Bender et al. [BFCP+05]. Veja também a nota de aula de Erik Demaine [Dem12]. Vamos primeiro definir os problemas *rmq* e ± 1-*rmq*.

> O problema *rmq-todos-pares*: Dado um vetor de inteiros V, pré-processe V de tal forma que uma consulta qualquer $rmq(i, j)$ seja respondida.

Em uma consulta $rmq(i, j)$ (assumimos $i \leq j$) desejamos saber qual é o índice k em V tal que $V(k)$ é um menor valor no vetor $V(i..j)$. No problema ± 1-*rmq-todos-pares*, o vetor V tem a propriedade ± 1: para quaisquer dois índices consecutivos i e j de V, $|V(i) - V(j)| = 1$.

O algoritmo baseia-se em uma redução do problema *acp-todos-pares* para o problema ± 1-*rmq-todos-pares*. Dada uma árvore enraizada T com n vértices, cada vértice armazena um (e somente um) rótulo entre 0 e $n - 1$ e o seu nível em T. Uma varredura de Euler em T é realizada armazenando algumas informações em três vetores *euler*, *nível* e *míndice*. Os dois primeiros vetores possuem $2n - 1$ posições e o último possui n. Em $euler(i)$ é armazenado o rótulo do i-ésimo vértice da varredura, em $nível[i]$ é armazenado o nível em T do i-ésimo vértice da varredura. A varredura começa na raiz e continua explorando novos vértices como uma busca em profundidade. Quando chega em uma folha, volta para o vértice anterior e continua a varredura em um novo vértice. A varredura acaba quando chega novamente na raiz e não existe um novo vértice para ser explorado. Na Figura 3.2 apresentamos um exemplo com uma árvore enraizada e os rótulos de cada vértice da árvore. A varredura de Euler e os valores nos vetores *euler* e *nível* para a árvore desta figura é dado em seguida.

euler :	0	1	3	7	3	8	3	1	4	1	5	9	5	1	0	2	6	10	6	11	6	12	6	2	0
nível :	0	1	2	3	2	3	2	1	2	1	2	3	2	1	0	1	2	3	2	3	2	3	2	1	0

Note que *nível* possui a propriedade ± 1. Em $míndice[i]$ armazenamos o índice em *euler* (e consequentemente em *nível*) da primeira ocorrência do vértice de T com rótulo i na varredura de Euler.

míndice :

0	1	15	2	8	10	16	3	5	11	17	19	21

.

Dados dois vértices i e j, o vetor $nível(míndice(i)..míndice(j))$ (estamos supondo que $míndice(i) \leq míndice(j)$) contém o nível de todos os vértices da varredura de Euler começando no vértice com rótulo i e terminando no vértice com rótulo j. Note que o ancestral comum mais próximo dos vértices correspondentes aos rótulos i e j é o vértice com menor nível neste intervalo. Por exemplo, o ancestral comum mais próximo dos vértices 4 e 7 é o vértice 1[2]. Veja que a primeira ocorrência do vértice 4 na varredura de Euler é $míndice(4) = 8$ e do vértice 7 é $míndice(7) = 3$. O menor valor em $nível(3..8)$ corresponde ao nível do ancestral comum

[2]Dizemos que o vértice com o rótulo i é o vértice i.

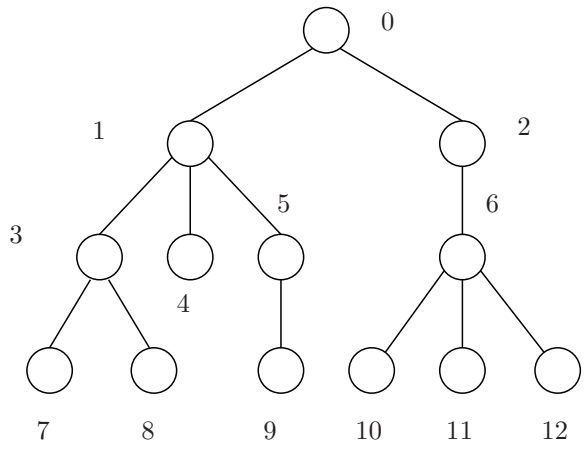


Figura 3.2: Uma árvore enraizada.

mais próximo desses vértices. Como o índice do menor nível em $nível(3..8)$ é $k = 7$, o ancestral comum mais próximo dos vértices 7 e 4 é $1 = euler(k = 7)$.

Dado um vetor V, vamos mostrar que é possível pré-processar V e responder a uma consulta ± 1-$rmq(i, j)$ consumindo espaço linear e tempo constante. Os primeiros passos são:

1. dividir o vetor *nível* em subvetores, cada um com tamanho $(\lg n)/2$ (o i-ésimo subvetor de *nível* é denotado por i-*nível*); e

2. criar um novo vetor $nível'$ com tamanho $n' = 2n/\lg n$, onde $nível'(i)$ armazena o menor valor do subvetor i-*nível*.

Considerando o exemplo anterior, teremos os seguintes novos vetores e subvetores:

$nível'$:
$\boxed{0|2|2|1|1|2|1|0|2|2|2|1|0}$
i-*níveis* :
$\boxed{0|1}\ \boxed{2|3}\ \boxed{2|3}\ \boxed{2|1}\ \boxed{2|1}\ \boxed{2|3}\ \boxed{2|1}\ \boxed{0|1}\ \boxed{2|3}\ \boxed{2|3}\ \boxed{2|3}\ \boxed{2|1}\ \boxed{0}$.

Considere que i e j são índices em *nível*. Uma consulta $rmq(i, j)$ em *nível* pode ser dividida em três consultas. Uma consulta no subvetor k'-*nível* que contém i (a partir de i até o final deste subvetor), uma consulta no subvetor l'-*nível* que contém j (do início deste subvetor até j) e uma consulta em $nível'(k' + 1..l' - 1)$. Por exemplo, uma consulta $rmq(3, 8)$ é dividida em uma consulta no subvetor 1-*nível* (começamos a contar a partir do 0), uma no subvetor 4-*nível* e uma no vetor $nível'(2..3)$.

$nível'$:
$\boxed{0|2|2|1|1|2|1|0|2|2|2|1|0}$
nível :
$\boxed{0|1}\ \boxed{2|3}\ \boxed{2|3}\ \boxed{2|1}\ \boxed{2|1}\ \boxed{2|3}\ \boxed{2|1}\ \boxed{0|1}\ \boxed{2|3}\ \boxed{2|3}\ \boxed{2|3}\ \boxed{2|1}\ \boxed{0}$.

Agora descrevemos como é obtido um menor valor em um subvetor i-*nível*. O que vamos fazer primeiro é mostrar que existem $O(\sqrt{n})$ diferentes subvetores. Lembre-se que qualquer subvetor tem a propriedade ± 1. Suponha que a cada elemento de cada subvetor foi adicionado o valor negativo do seu primeiro elemento. Isso não altera a resposta para qualquer consulta em qualquer subvetor. Com isso, cada subvetor começa com o valor 0. Considere um subvetor $V_{sub} = \boxed{0|1|0|-1|-2|-1}$. Note que o vetor $\boxed{0|1|-1|-1|-1|1}$ é uma representação de V_{sub}.

Isto é, V_{sub} começa com 0, o próximo valor é o valor anterior mais 1, o próximo é o valor anterior -1, e assim por diante. Com isso, fica fácil ver que o número total de diferentes subvetores é $2^{((\lg n)/2)-1} = O(\sqrt{n})$. Esse resultado sugere que apesar de todo subvetor ser "fisicamente distinto" um do outro, muitos deles podem ser vistos como um mesmo subvetor, se considerarmos os seus conteúdos. Veja no exemplo anterior que existem 6 subvetores $\boxed{2}\boxed{3}$ de um total de 13. Agora, dado que cada subvetor possui $O((\lg n)/2)$ elementos, o número total de consultas sobre cada subvetor é $O(((\lg n)/2)^2)$. Com isso, podemos construir uma tabela T_1 com $O(\sqrt{n})$ linhas e $O(\lg^2 n)$ colunas. Cada linha de T_1 representa uma possível combinação dos elementos de um subvetor e cada coluna representa uma possível consulta em um subvetor. É consumido $O(\lg \lg n)$ para armazenar um índice em uma entrada desta tabela pois o tamanho de cada subvetor é $O(\lg n)$. Portanto, esta tabela consome $O((\sqrt{n})(\lg^2 n)(\lg \lg n)) = o(n)$ bits.

Agora mostramos como é obtido um menor valor no vetor *nível'*. Para isso, construímos uma nova tabela T_2 que consome espaço linear. Lembre que o tamanho de *nível'* é $n' = 2n/\lg n$. Criamos T_2 com uma linha para cada índice i de *nível'* e uma coluna para cada valor i e $i + 2^k$ com $k = 0, 1, \ldots,$ enquanto $i + 2^k \leq n'$. Armazenamos em $T_2(i, i)$ o índice i de *nível'* e em $T_2(i, i + 2^k)$ o índice j de *nível'* tal que *nível'*(j) é um menor valor em *nível'*$(i, \ldots, i + 2^k)$. O número de entradas de T_2 é $O(n' \lg n') = O((2n/\lg n)\lg(2n/\lg n)) = O((2n/\lg n)(\lg 2n) - (2n/\lg n)(\lg \lg n)) = O(2n + (2n/\lg n)(1 - \lg \lg n)) = O(n)$. Usa-se programação dinâmica para preencher T_2. Por fim, considere que é pedido um menor valor em *nível'*$(i, \ldots, j)$. Denote por l o valor $j - i + 1$. Considere a maior potência de 2 menor ou igual que l, ou seja, $2^{\lfloor \lg l \rfloor}$. Para encontrar um menor valor no intervalo dado depois da construção de T_2, basta devolver o $\arg\min(\textit{nível'}(T_2(i, i + 2^{\lfloor \lg l \rfloor} - 1)), \textit{nível'}(T_2(j - 2^{\lfloor \lg l \rfloor} + 1, j)))$.

3.2 Um Ancestral Comum mais Próximo em Digrafos Acíclicos

Nesta seção descrevemos três algoritmos que resolvem o seguinte problema.

> O problema *representante-acp-todos-pares*: Dado um digrafo acíclico D, pré-processe D de tal forma que uma consulta qualquer *representante-acp*(u, v) seja respondida.

Bender et al. [BFCP+05] definiram o problema *representante-acp-todos-pares*. Baumgart et al. [BEG+07] mostraram como usar programação dinâmica para obter um pré-processamento com tempo $O(nm)$ para o problema. Depois disso, uma consulta *representante-acp*(u, v) é realizada em tempo constante. Bender et al. [BFCP+05] mostraram como pré-processar D em tempo $\tilde{O}(n^{2,68635})$[3] e então uma consulta pode ser feita em tempo constante. Mais tarde, Czumaj et al. [CKL07] mostraram como pré-processar D em tempo $O(n^{2,5719})$. Os dois últimos algoritmos usam um algoritmo para multiplicação de matrizes. Por isso, os resultados nestes casos têm mais valores teóricos que práticos. Por último, destacamos nossa contribuição para um problema mais fraco chamado de *representante-acp-k-pares*. Neste problema, além de um digrafo acíclico D como entrada, também são dados k pares de vértices para os quais desejamos saber um ancestral comum mais próximo. Obtemos um algoritmo rápido quando o número de arcos de D e o número de pares dados são limitados.

3.2.1 Algoritmo de Baumgart, Eckhardt, Griebsch, Kosub e Nowak

Baumgart et al. [BEG+07] apresentaram um algoritmo para o problema *representante-acp-todos-pares* que consome tempo $O(nm)$. Outro algoritmo com mesma complexidade de tempo pode ser obtido no trabalho de Czumaj et al. [CKL07]. Considere um digrafo acíclico D, o fecho transitivo D_{ft} de D, uma ordenação topológica dos vértices de D, um par de vértices x e y de D. A seguinte recorrência determina o vértice z que é o ancestral comum mais próximo com maior valor topológico de x e y: se $x \to y$ pertence a D_{ft}, então z é o próprio vértice x; caso

[3] $f(n) = \tilde{O}(g(n))$ se existe uma constante c tal que $f(n) = O(g(n) \log^c n)$

contrário, z é o vértice no conjunto $\{z_1, \ldots, z_k\}$ com maior valor topológico, onde $x_1, \ldots, x_k$ são os pais de x em D e z_i é o ancestral comum mais próximo com maior valor topológico de x_i e y para $i = 1, \ldots, k$.

O algoritmo de [BEG+07] é dado em seguida. Determine o fecho transitivo D_{ft} de D e uma ordenação topológica dos vértices de D. Inicialize uma matriz M $(n \times n)$ com um vértice fictício cujo valor topológico fictício é igual a $-\infty$. Para cada vértice v de D na ordem topológica crescente, e para cada filho x de v, faça a seguinte atribuição para cada vértice y topologicamente maior que v:

$$M(x,y) \leftarrow \begin{cases} x, & \text{se o arco } x \to y \text{ está em } D_{ft} \text{ ou} \\ M(v,y), & \text{se } M(v,y) \text{ é topologicamente maior que } M(x,y). \end{cases}$$

O algoritmo descrito está correto e consome tempo $O(nm)$ [BEG+07].

3.2.2 Algoritmo de Bender, Farach-Colton, Pemmasani, Skiena e Sumazin

O algoritmo de Bender et al. [BFCP+05] contém uma redução do problema *representante-acp-todos-pares* para o problema clássico *menor-distância-todos-pares*, um algoritmo que aproxima a menor distância entre todos os pares de vértices e um algoritmo que usa essa menor distância aproximada para encontrar um ancestral comum mais próximo para todo par de vértices. Em seguida definimos o problema *menor-distância-todos-pares*.

> O problema *menor-distância-todos-pares*: Dado um digrafo D com custo não negativo nos arcos, pré-processe D de tal forma que uma consulta qualquer *menor-distância*(u, v) seja respondida.

Em uma consulta *menor-distância*(u, v) desejamos saber qual é a menor distância de u para v em D. Em breve explicaremos como obter a resposta para uma consulta *representante-acp*(u, v) através de uma consulta *menor-distância*(u', v'). Agora mostramos a redução do problema *representante-acp-todos-pares* para o problema *menor-distância-todos-pares*. Considere um digrafo acíclico D. A primeira coisa a fazer é adicionar custos nos arcos de D. Para isso, é associado a cada vértice v de D a sua profundidade, denotada por $prof(v)$. A profundidade de um vértice em um digrafo acíclico é o tamanho de um maior caminho de um vértice fonte para v. Obtemos $prof(v)$ em tempo linear através de uma ordenação topológica e um algoritmo de programação dinâmica: o processamento dos vértices é feito na ordem crescente da ordenação topológica e um vértice v possui $prof(v) = prof(w) + 1$ onde w é um pai de v com maior valor $prof(w)$. Depois, é feita uma ordenação dos vértices considerando a profundidade obtida. Provavelmente aparecerão vértices u e v com $prof(u) = prof(v)$. Com $prof$ e uma ordenação topológica do vértices é fácil construir um novo vetor $prof'$ de tal forma que $prof'(u) \neq prof'(v)$ e, se $prof(u) \leq prof(v)$, então $prof'(u) \leq prof'(v)$ para qualquer par de vértices u e v. Assim, teremos valores distintos para qualquer par de vértices. Finalmente, atribuímos custos aos arcos de D. O custo do arco $u \to v$ é o valor $prof'(v) - prof'(u)$. Definimos a *distância* de u para v, denotada por $dist(u, v)$, como sendo um caminho de menor custo de u para v. Assim, as seguintes propriedades são satisfeitas.

Propriedade 3.1 (Bender et al. [BFCP+05]). *Se $u = v$, ou se existe um caminho de u para v então:*

1. $prof'(u) \leq prof'(v)$;
2. $dist(u, v) = prof'(v) - prof'(u)$;

Na verdade, qualquer caminho de u para v tem o mesmo custo. Vamos ver o porquê disso. Tome um caminho de u para v denotado por $P = u = u_1, u_2, \ldots, u_k = v$. O custo

dos arcos $u_i \to u_{i+1}$ é $prof'(u_{i+1}) - prof'(u_i)$ para todo $i = 1, \ldots, k-1$. Então, o custo do caminho P é $prof'(u_2) - prof'(u_1) + prof'(u_3) - prof'(u_2) + \cdots + prof'(u_k) - prof'(u_{k-1}) = prof'(u_k) - prof'(u_1) = prof'(v) - prof'(u)$.

Agora, suponha que dois vértices u e v tenham um ancestral comum. Considere que o vértice w é um ancestral comum mais próximo de u e v. Note a seguinte e muito importante propriedade.

Propriedade 3.2 (Bender et al. [BFCP+05]). *Se w é um ancestral comum mais próximo de u e v, então $dist(w,u) + dist(w,v) = prof'(u) + prof'(v) - 2prof'(w)$.*

Note que se u é um ancestral de v, isto é, existe um caminho de u para v, então a partir da Propriedade 3.2 obtemos a Propriedade 3.1 *2*. Se u e v não se relacionam, então dizemos que a *distância indireta* de u para v (ou de v para u) passando por w, um ancestral comum mais próximo de u e v, é $dist_{ind}(u,v) = dist_{ind}(v,u) = prof'(u) + prof'(v) - 2prof'(w)$. Agora, crie um novo digrafo D' que é formado por duas cópias D_1 e D_2 do digrafo de entrada D mais um arco $v_2 \to v_1$ com custo 0 para todo vértice v de D onde v_1 é o vértice que representa v em D_1 e v_2 é o vértices que representa v em D_2. Em [BFCP+05] é demonstrado que um caminho de menor custo de u_2 para v_1 em D' passa por um ancestral comum mais próximo de u e v em D com maior valor $prof'$. A redução termina aqui e agora podemos usar qualquer algoritmo para o problema *menor-distância-todos-pares* sobre D'. Resolvido o problema e dados quaisquer dois vértices u e v, usamos a equação $dist_{ind}(u,v) = prof'(u) + prof'(v) - 2prof'(w)$ para obter w que é um ancestral comum mais próximo de u e v em D pois $prof'$ é único para todo vértice em D. Não se conhece um algoritmo que resolva o problema *menor-distância-todos-pares* em tempo $o(n^3)$. Por isso, Bender et al. [BFCP+05] decidiram usar o algoritmo de Zwick [Zwi98] que, para este caso, aproxima $dist_{ind}(u,v)$ em tempo $\tilde{O}(n^\omega/\varepsilon)$. Com isso, obtemos uma aproximação para $prof'$ de um ancestral comum mais próximo de u e v com um erro de $\varepsilon > 0$. Temos $O(\varepsilon n)$ vértices candidatos a um ancestral comum mais próximo para um determinado par de vértices pois a profundidade máxima para qualquer vértice é $O(n)$ e não existe profundidade igual ($prof'$) para qualquer par de vértices. Logo, é consumido tempo $O(\varepsilon n^3)$ no total para escolhermos o vértice com maior valor $prof'$ dentre os $O(\varepsilon n)$ candidatos para cada par de vértices. Note que o melhor valor para ε é $n^{(w-3)/2}$, isto é, quando $O(\varepsilon n^3) = \tilde{O}(n^\omega/\varepsilon)$. Portanto, o algoritmo descrito resolve o problema *representante-acp-todos-pares* em tempo $\tilde{O}(n^{(\omega+3)/2})$. Atualmente este tempo é limitado superiormente por $O(n^{2,68635})$.

3.2.3 Algoritmo de Czumaj, Kowaluk e Lingas

Czumaj et al. [CKL07] realizam uma redução do problema *representante-acp-todos-pares* para o problema *máxima-testemunha-todos-pares*. Em seguida definimos tal problema. Este mesmo resultado aparece independentemente no trabalho de Baumgart et al. [BEG+07].

> O problema *máxima-testemunha-todos-pares*: Dadas duas matrizes booleanas quadradas $(n \times n)$ A e B, encontrar, se existir, o maior valor de k tal que $A(i,k)B(k,j) = 1$ para todo $i, j = 1, \ldots, n$.

Os seguintes passos realizam uma redução entre os problemas em tempo $\tilde{O}(n^\omega)$. Dado um digrafo acíclico D, compute o fecho transitivo D_{ft} de D em tempo $\tilde{O}(n^\omega)$. Faça uma ordenação topológica dos vértices de D_{ft} e considere que a ordem obtida dos vértices de D_{ft} é $v_1 \leq v_2 \leq \cdots \leq v_n$. Crie uma matriz A com n linhas e n colunas tal que $A(v_i, v_j) = 1$ se e somente se o vértice v_j é um ancestral do vértice v_i em D_{ft}. Crie uma matriz B que é igual a matriz transposta de A. Assim, $B(v_i, v_j) = 1$ se e somente se o vértice v_i é um ancestral do vértice v_j em D_{ft}. Agora considere uma entrada $C(v_i, v_j) = A(v_i, v_k)B(v_k, v_j) = 1$ tal que v_k é máximo. Sabemos que v_k é um ancestral comum de v_i e v_j pois $A(v_i, v_k) = B(v_k, v_j) = 1$ e que $v_{k'} \leq v_k$ para todo ancestral comum $v_{k'}$ de v_i e v_j. Como descrito em [BFCP+05] e [CKL07],

v_k é um ancestral comum mais próximo dos vértices v_i e v_j. Isso é verdade porque v_k é um ancestral comum dos vértices com valor topológico máximo. Portanto, qualquer vértice com valor topológico maior que v_k não é comum aos vértices v_i e v_j.

Para resolver o problema *máxima-testemunha-todos-pares*, os autores particionam as matrizes A e B em n/l submatrizes: submatriz A^i $(n \times l)$ e B^i $(l \times n)$ para $i = 1, \ldots, n/l$. As colunas da submatriz A^i são as colunas $(i-1)l+1, \ldots, il$ de A e as linhas de B^i são as linhas de $(i-1)l+1, \ldots, il$ de B (veja a Figura 3.3).

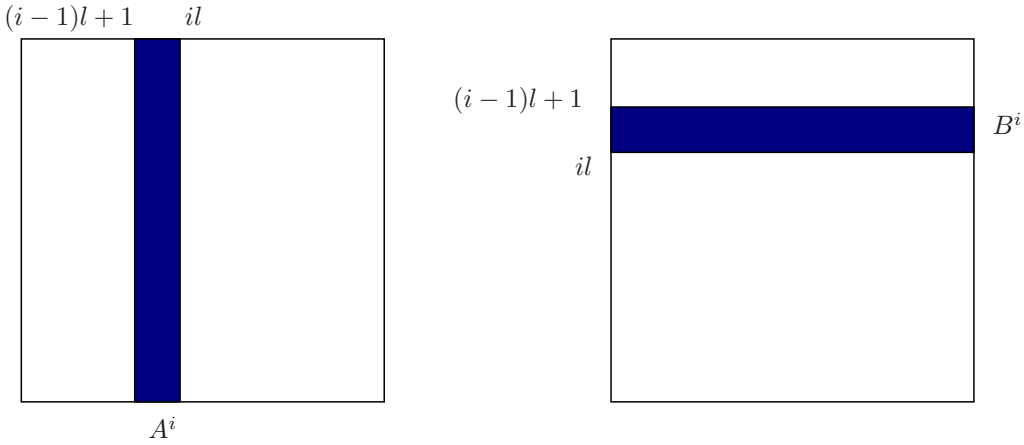


Figura 3.3: Matrizes retangulares A^i e B^i [CKL07].

Os autores computam o produto da matriz retangular $A^iB^i = C^i$ para $i = 1, \ldots, n/l$. Note que cada matriz C^i possui n linhas e n colunas e que $C^i(v_i, v_j) = 1$ se e somente se algum vértice v_k no intervalo $[(i-1)l+1 : il]$ é um ancestral comum de v_i e v_j. Lembre que $\omega(1,1,r)$ é o expoente que aparece no tempo do algoritmo mais rápido que multiplica matrizes retangulares $(n \times n^r)$ e $(n^r \times n)$. Como cada A^i possui n linhas e $l = n^r$ colunas, temos $r = \log_n l$. Logo, o algoritmo consome tempo $O(n^{\omega(1,1,\log_n l)}(n/l))$ para computar C^i para todo $i = 1, \ldots, n/l$, consome tempo $O(n^3/l)$ para encontrar o maior valor i tal que $C^i(v_i, v_j) = 1$ e tempo $O(n^2 l)$ para encontrar v_k no intervalo $[(i-1)l+1 : il]$ para todo par de vértices v_i e v_j. No total o algoritmo consome tempo $O(n^{\omega(1,1,\log_n l)}(n/l) + n^3/l + n^2 l)$. Escrevendo o consumo de tempo em função de r, temos $O(n^{1-r+\omega(1,1,r)} + n^{3-r} + n^{2+r})$. Se $r \geq 1/2$ então o consumo é $O(n^{1-r+\omega(1,1,r)} + n^{2+r})$. Igualando os expoentes $1 - r + \omega(1,1,r) = 2 + r$ chegamos em $\omega(1,1,r) = 2r + 1$. Com o limite inferior para $\omega(1,1,r) \geq 2$ temos a desigualdade $r \geq 1/2$ satisfeita. Portanto, quando $\omega(1,1,r) = 1 + 2r$, o consumo de tempo do algoritmo é $O(n^{2+r})$. Agora os autores usam o seguinte resultado de [LR83] que limita $\omega(1,1,r)$ superiormente.

Teorema 3.1 (Lotti e Romani [LR83])**.** *Considere* $\omega = \omega(1,1,1)$ *e* $\alpha = \sup\{r | \omega(1,1,r) = 2\}$. *Assim,*

$$\omega(1,1,r) = \begin{cases} 2, & \textit{se } 0 \leq r \leq \alpha \textit{ ou} \\ 2 + (\omega - 2)\frac{r-\alpha}{1-\alpha}, & \textit{se } \alpha \leq r \leq 1. \end{cases}$$

□

Substituindo $\omega(1,1,r)$ por $1+2r$ no Teorema 3.1 e usando os limitantes atuais para ω e α que são respectivamente $2,373$ [Wil12] e $0,30298$ [LG12], conseguimos isolar r e obter $r \leq 0,5719$. Logo, o consumo de tempo do algoritmo atualmente é $O(n^{2,5719}) = O(n^{2+r})$, onde r satisfaz a equação $\omega(1,1,r) = 1 + 2r$.

3.2.4 Contribuição

Considere a seguinte variação do problema *representante-acp-todos-pares* onde os pares são dados na entrada do problema:

> O problema *representante-acp-k-pares*: Dado um digrafo acíclico D e pares de vértices $\mathcal{P} = \{u_1, v_1\}, \ldots, \{u_k, v_k\}$, determinar um representante acp para cada par de vértices de $\mathcal{P}$.

Note que os algoritmos descritos anteriormente resolvem o problema proposto. Eles encontram um representante acp para todos os pares de vértices. Logo, encontram para k pares de vértices dados. No Capítulo 4, desenvolvemos um algoritmo simples e de fácil implementação capaz de resolver este problema em tempo $O(n(m + k))$. Com isso, conseguimos resolver o problema *representante-acp-k-pares* em tempo $o(n^{2,5719})$ caso $(m + k) = o(n^{1,5719})$.

3.3 Todos os Ancestrais Comuns mais Próximos em Digrafos Acíclicos

Nesta seção descrevemos alguns algoritmos para o problema *todos-acps-todos-pares*.

> O problema *todos-acps-todos-pares*: Dado um digrafo acíclico D, pré-processe D de tal forma que uma consulta qualquer *todos-acps*(u, v) seja respondida.

Baumgart et al. [BEG+07] desenvolveram vários algoritmos para este problema. Vamos descrever aqui dois dos seus algoritmos. O pré-processamento de um é feito em tempo $O(n^2m)$ e do outro em tempo $O(n^{3,5719})$. A resposta a uma consulta *todos-acps*(u, v) é feita em tempo $O(k')$, onde k' é o tamanho do conjunto $\mathcal{ACP}_{u,v}$. Eckhardt et al. [EMN07] mostraram como pré-processar D em tempo $O(n^{3,2567})$. Mais uma vez, no final desta seção destacamos nossa contribuição para um problema similar chamado de *todos-acp-k-pares*. Novamente, a entrada deste problema é composta por um digrafo acíclico D e k pares de vértices para os quais desejamos saber todos os acps. Ao limitar o número de pares dados, obtemos um bom algoritmo para o problema.

3.3.1 Algoritmos de Baumgart, Eckhardt, Griebsch, Kosub e Nowak

O primeiro algoritmo consome tempo $O(n^2m)$. Dado um digrafo acíclico D, primeiramente é computado o fecho transitivo D_{ft} de D. Em seguida, para cada vértice z e para cada par de vértices u e v, o vértice z é um ancestral comum mais próximo dos vértices u e v se e somente se os arcos $z \to u$ e $z \to v$ estão em D_{ft} e não existe filho x de z em D tal que $x \to u$ e $x \to v$ estão em D_{ft}.

O segundo algoritmo consome tempo $O(n^{3+r})$, onde $r \leq 0,5719$ (usamos aqui o mesmo resultado descrito na Seção 3.2.3). Portanto, o algoritmo atualmente consome tempo $O(n^{3,5719})$. O algoritmo descrito na Seção 3.2.3 resolve o problema *representante-acp-todos-pares*. Aqui ele é usado como subrotina. Lembre que este algoritmo encontra, para cada par de vértices, o representante acp com maior valor em uma ordenação topológica. Assim, considere um vértice fixo z de D. Considere que z é um ancestral comum mais próximo de um par de vértices u e v. Logo, se forçarmos z ter um valor topológico maior que todos os outros ancestrais comuns de u e v, o algoritmo auxiliar devolve corretamente z como um ancestral comum mais próximo de maior valor topológico para os vértices u e v. Para resolver o problema *todos-acps-todos-pares*, basta aplicar a ideia anterior para todo vértice de D. Portanto, para cada vértice z de D:

1. Compute uma ordenação topológica que maximiza o valor de z;
2. Resolva o problema *representante-acp-todos-pares* (Seção 3.2.3) encontrando, para todo par de vértices, o representante acp com valor topológico máximo; e

3. Para cada par de vértices u e v, adicione z a uma lista de ancestrais comuns mais próximos de u e v somente se z é o representante acp encontrado para este par.

Falta dizer como obter uma ordenação topológica que maximiza o valor de z. Primeiro remova de D o vértice z e todos os seus descendentes. Depois, encontre uma ordenação topológica sobre os vértices restantes de D. Por último, encontre uma ordenação topológica de z com os seus descendentes, e concatene ambas ordenações. Veja um exemplo na Figura 3.4.

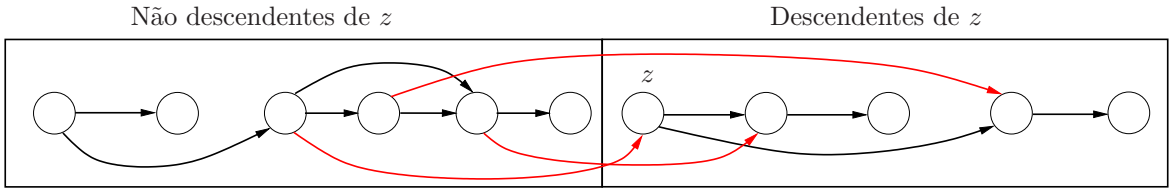


Figura 3.4: Ordenação topológica que maximiza o valor de z.

3.3.2 Algoritmos de Eckhardt, Mühling e Nowak

Eckhardt et al. [EMN07] descrevem um algoritmo para resolver o problema *s-acp-todos-pares* definido em seguida.

> O problema *s-acp-todos-pares*: Dado um digrafo acíclico D e um vértice s, pré-processe D de tal forma que uma consulta qualquer *s-acp*(u, v) seja respondida.

O tempo gasto para pré-processar o digrafo acíclico de entrada é $\tilde{O}(n^\omega)$, onde ω é o expoente que aparece no tempo do algoritmo mais rápido que multiplica matrizes quadradas. Vamos ver como isso é possível. Os autores consideram o fecho transitivo D_{ft} de D e várias matrizes booleanas quadradas $(n \times n)$: a matriz de adjacência A de D_{ft}, uma matriz C^s cuja entrada $C^s(u, v) = 1$ se e somente se s é um ancestral comum dos vértices u e v, uma matriz A^s cuja entrada $A^s(u, z) = 1$ se e somente se z é um filho de s e existe um caminho de z para u em D, e uma matriz de testemunhas W^s que descrevemos em breve.

A matriz C^s pode ser obtida em tempo $O(n^2)$ através de D_{ft} e a matriz A^s é a matriz transposta de A restrita às linhas que indexam os filhos de s em D. A matriz booleana de testemunhas W^s é obtida a partir do produto de duas matrizes, $W^s = A^s A$. Note que uma entrada $W^s(u, v) = 1$ se existe um filho z de s em D tal que $A^s(u, z) = A(z, v) = 1$. Neste caso, z (um filho de s) é um ancestral comum dos vértices u e v e portanto s não pode ser um ancestral comum mais próximo dos vértices u e v. O seguinte lema aparece em [EMN07].

Lema 3.4 (Eckhardt, Mühling e Nowak [EMN07])**.** *Considere as matrizes booleanas C^s, A^s e A e a matriz de testemunhas $W^s = A^s A$. Então, s é um ancestral comum mais próximo dos vértices u e v se e somente se $W^s(u, v) = 0$ e $C^s(u, v) = 1$.* □

Portanto, o problema *s-acp-todos-pares* pode ser resolvido em tempo $\tilde{O}(n^\omega)$. E assim, o problema *todos-acps-todos-pares* pode ser resolvido em tempo $O(n^{1+\omega})$. Isso porque gastamos $\tilde{O}(n^\omega)$ para calcular D_{ft}. Depois, gastamos tempo $O(n^\omega)$ para resolver o problema *s-acp-todos-pares* para cada vértice s de D. Com o limitante atual para ω temos o consumo de tempo igual a $O(n^{3,373})$. No entanto, os mesmos autores resolvem o problema *todos-acps-todos-pares* em tempo $O(n^{\omega(2,1,1)}) = O(n^{\omega(1,1,2)})$[4] com o uso de um algoritmo para multiplicar matrizes retangulares. Atualmente o melhor limitante superior para $\omega(1, 1, 2)$ é 3,2567 [LG12]. Para

[4]Como descrito em [LG12], Lotti e Romani [LR80] mostraram que a multiplicação de matrizes retangulares $(n \times n)$ e $(n \times m)$ ou $(m \times n)$ e $(n \times n)$ pode ser feito essencialmente no mesmo tempo que a multiplicação de matrizes retangulares $(n \times m)$ e $(m \times n)$

atingir tal limitante basta observar que as matrizes booleanas de testemunhas W^s podem ser todas obtidas através da seguinte multiplicação das matrizes $(n^2 \times n)$ e $(n \times n)$:

$$\begin{pmatrix} A^{s_1} \\ A^{s_2} \\ \vdots \\ A^{s_n} \end{pmatrix} A = \begin{pmatrix} W^{s_1} \\ W^{s_2} \\ \vdots \\ W^{s_n} \end{pmatrix}.$$

3.3.3 O Problema *todos-acps-todos-pares* em Digrafos Gerais

Os autores dos algoritmos descritos nesta seção supõem como entrada um digrafo acíclico. Porém, notamos que alguns algoritmos podem ser aplicados em digrafos gerais. Lembre que um ancestral comum mais próximo de um par de vértices aparece em digrafos gerais como ilustrado nos três últimos digrafos da Figura 2.6, Seção 2.3. Os algoritmos que resolvem o problema *todos-acps-todos-pares* em digrafos gerais são: o algoritmo $O(n^2m)$ de Baumgart, Eckhardt, Griebsch, Kosub e Nowak; e os algoritmos $O(n^{3,373})$ e $O(n^{3,2567})$ de Eckhardt, Mühling e Nowak. Os algoritmos funcionam essencialmente da seguinte forma. Dado um vértice s candidato a um ancestral comum mais próximo de dois vértices u e v, s é descartado se existe um arco $s \to w$ com caminhos de w para u e de w para v. Essa ideia pode ser perfeitamente aplicada aos digrafos gerais, sem supor aciclicidade.

3.3.4 Contribuição

Considere a seguinte variação do problema *todos-acps-todos-pares* onde as consultas são dadas na entrada do problema:

> O problema *todos-acps-k-pares*: Dado um digrafo acíclico D e pares de vértices $\mathcal{P} = \{u_1, v_1\}, \ldots, \{u_k, v_k\}$, determinar todos os acps para cada par de vértices de $\mathcal{P}$.

Os algoritmos das seções anteriores podem ser usados para resolver o problema proposto. No entanto, aproveitamos o fato de que as consultas são dadas na entrada para produzir um algoritmo simples e prático para este problema. Veremos no Capítulo 4 que o tempo gasto pelo novo algoritmo é $O(nm + n^2k)$. Se $k = o(n^{1,2567})$, então o novo algoritmo é mais rápido que o melhor algoritmo para o problema *todos-acps-todos-pares*.

3.4 Uma Junção em Digrafos Acíclicos

O problema tratado nesta seção é o problema *representante-junção-todos-pares*.

> O problema *representante-junção-todos-pares*: Dado um digrafo acíclico D, pré-processe D de tal forma que uma consulta qualquer *representante-junção*(u, v) seja respondida.

Note que qualquer algoritmo para o problema *representante-acp-todos-pares* pode ser também usado no problema *representante-junção-todos-pares*. No entanto, Yuster [Yus08] mostrou como pré-processar D tempo $\tilde{O}(n^\omega)$, onde ω é o expoente que aparece no tempo do algoritmo mais rápido que multiplica matrizes quadradas. Da mesma forma como foi feito nas seções anteriores, definimos e descrevemos a nossa contribuição para um problema similar chamado *representante-junção-k-pares*.

3.4.1 Algoritmo de Yuster

O algoritmo de programação dinâmica de Yuster resolve o problema *representante-junção-todos-pares* em tempo $\tilde{O}(n^\omega)$. O algoritmo considera uma ordenação topológica dos vértices de D e várias matrizes: a matriz de adjacência $A = A_0$ de D com 1 na diagonal principal, matrizes produto $A_{i+1} = A_i A_i$ para $i = 0, 1, \ldots, \lceil \lg(n-1) \rceil$ (note que $A_{\lceil \lg(n-1) \rceil}$ é o fecho transitivo de D), a matriz de testemunhas W_i para cada matriz A_i, uma matriz de ancestrais comuns C tal que $C(u, v) = w$ se w é um ancestral comum dos vértices u e v, e uma matriz dos menores índices F, onde $F(u, v) = r$ é o menor índice tal que $A_r(u, v) = 1$. Se u e v não têm ancestral comum, então $C(u, v) = 0$. Se não existe caminho de u para v, então $F(u, v) = \lceil \lg(n-1) \rceil + 1$. A matriz de testemunhas W_i pode ser obtida através do algoritmo de Alon et al. em tempo $\tilde{O}(n^\omega)$ [AN96]. Como temos $O(\log n)$ matrizes A_i, todas as matrizes W_i podem ser obtidas em tempo $\tilde{O}(n^\omega)$. A matriz C pode ser obtida em tempo $\tilde{O}(n^\omega)$ modificando o algoritmo de Bender et al. que resolve o problema da existência de um ancestral comum para todos os pares de vértices u e v. Neste algoritmo é considerado um grafo D'' composto por D, uma cópia D' de D com os arcos invertidos e uma mesclagem dos vértices sorvedouros de D' com os vértices fontes de D. Veja um exemplo na Figura 3.5.

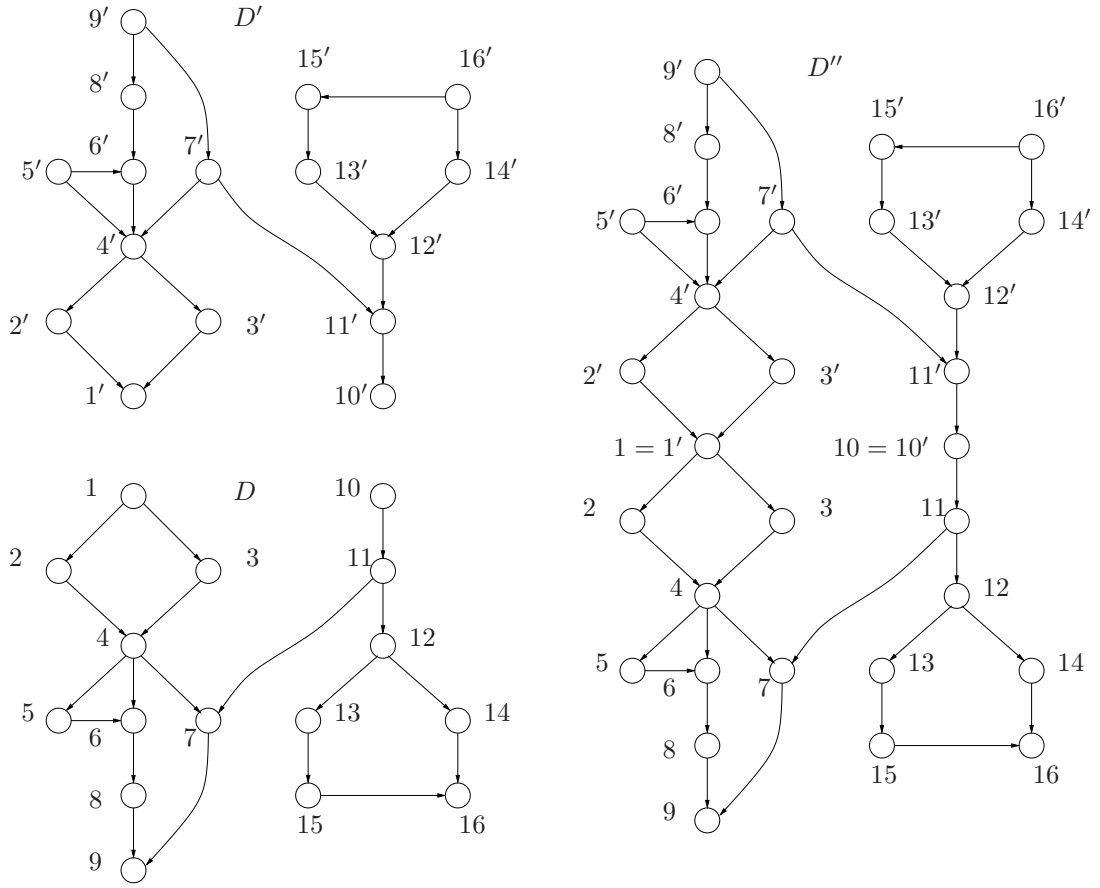


Figura 3.5: Digrafos D, D' e D''.

Pelo trabalho de Bender et al. [BFCP+05] sabemos que existe um ancestral comum dos vértices u e v em D se e somente se existe um caminho de u' para v em D'', onde u' é o vértice em D' correspondente ao vértice u de D e v é um vértice de D. Agora suponha que obtemos o fecho transitivo de D'' da mesma forma que foi obtido para D, isto é, defina B_0 como sendo a matriz de adjacência de D'' e calcule $B_{i+1} = B_i B_i$ para $i = 0, 1, \ldots, \lceil \lg(n-1) \rceil$. Assim, faça $C(u, v) = w$ no primeiro índice i tal que $B_i(u', w) B_i(w, v) = 1$, onde w é um vértice fonte em

D, u' é um vértice somente de D', e v um vértice somente de D.

Dito isso, vamos ao algoritmo de Yuster. Ele produz uma matriz M onde

$$M(u,v) = \begin{cases} 0, & \text{se os vértices } u \text{ e } v \text{ não têm junção ou} \\ w, & \text{se } w \text{ é uma junção dos pares } u \text{ e } v. \end{cases}$$

Para cada par de vértices distintos, u topologicamente menor que v, se u é um ancestral de v ($A_{\lceil \lg(n-1) \rceil}(u,v) = 1$), então u é uma junção dos vértices u e v ($M(u,v) \leftarrow u$) e o próximo par de vértices na ordem é considerado. Se u não é um ancestral de v ($A_{\lceil \lg(n-1) \rceil}(u,v) = 0$), então o algoritmo verifica a existência de um ancestral comum ao par u e v. Se não existe ancestral comum ao par u e v ($C(u,v) = 0$), então não existe junção para o par ($M(u,v) \leftarrow 0$) e o próximo par de vértices na ordem é considerado. Se existe um ancestral comum w ao par ($C(u,v) = w$), então os seguintes passos do algoritmo percorrem um caminho de w para u até que um pai de u seja atingido:

1. $s \leftarrow F(w,u)$ e

2. $w \leftarrow W_s(w,u)$.

Neste ponto, podemos supor que w é um pai de u, u não é um ancestral de v, e que uma junção para o par w e v já foi calculada em uma iteração anterior do algoritmo. Portanto, a junção obtida para o par w e v é também uma junção para o par u e v. O algoritmo faz a atribuição $M(u,v) \leftarrow M(w,v)$ e considera o próximo par de vértices na ordem. Uma prova feita em [Yus08] mostra que este algoritmo está correto e que são feitas $O(\log n)$ iterações para cada par de vértices u e v. Logo, o tempo gasto é $\tilde{O}(n^{\omega})$. Em seguida, simulamos uma iteração do algoritmo para um par de vértices da Figura 3.6. Acompanhe a simulação nas matrizes A_0, A_3 e F, W.

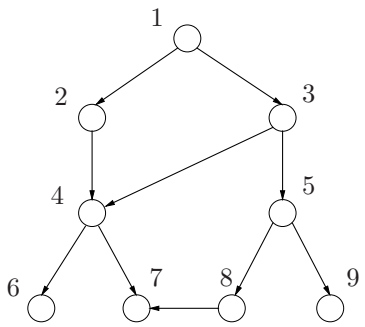


Figura 3.6: Digrafo acíclico com rótulo nos vértices.

A_0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	...
1	1	1	1	0	0	0	0	0	0	...
2	0	1	0	1	0	0	0	0	0	...
3	0	0	1	1	1	0	0	0	0	...
4	0	0	0	1	0	1	1	0	0	...
5	0	0	0	0	1	0	0	1	1	...
6	0	0	0	0	0	1	0	0	0	...
7	0	0	0	0	0	0	1	0	0	...
8	0	0	0	0	0	0	1	1	0	...
9	0	0	0	0	0	0	0	0	1	...

A_3	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	1	1	1	1	1	1	1	1	1
2	0	1	0	1	0	1	1	0	0
3	0	0	1	1	1	1	1	1	1
4	0	0	0	1	0	1	1	0	0
5	0	0	0	0	1	0	1	1	1
6	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	1	1	0
9	0	0	0	0	0	0	0	0	1

F, W	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	$W_0, 1$	$W_0, 1$	$W_0, 1$	$W_1, 2$	$W_1, 3$	$W_2, 2$	$W_2, 2$	$W_2, 3$	$W_2, 3$
2	-	$W_0, 2$	-	$W_0, 2$	-	$W_1, 4$	$W_1, 4$	-	-
3	-	-	$W_0, 3$	$W_0, 3$	$W_0, 3$	$W_1, 4$	$W_1, 4$	$W_1, 5$	$W_1, 5$
4	-	-	-	$W_0, 4$	-	$W_0, 4$	$W_0, 4$	-	-
5	-	-	-	-	$W_0, 5$	-	$W_1, 8$	$W_0, 5$	$W_0, 5$
6	-	-	-	-	-	$W_0, 6$	-	-	-
7	-	-	-	-	-	-	$W_0, 7$	-	-
8	-	-	-	-	-	-	$W_0, 8$	$W_0, 8$	-
9	-	-	-	-	-	-	-	-	$W_0, 9$

O vértice 1 é um ancestral comum de todo par de vértices u e v, isto é, $C(i, j) = 1$ para todo $i, j = 1, \ldots, 9$. As matrizes F e W foram descritas paralelamente. Se a posição $F, W(u, v) = W_i, z$, então $F(u, v) = i$ e $W_i(u, v) = z$. As matrizes intermediárias A_1 e A_2 não são descritas. Considere uma ordenação topológica dos vértices da Figura 3.6 em que o vértice 9 está na última posição. A simulação determinará uma junção para o par $(6, 9)$. Iterações anteriores do algoritmo determinaram uma junção para cada par de vértices (x, y) onde o vértice x vem antes que o vértice 6 na ordenação topológica. Como $A_3(6, 9) = 0$, não existe um caminho de 6 para 9. Isso pode ser verificado na Figura 3.6. Porém este par tem o vértice 1 como ancestral comum. Então o algoritmo percorre um caminho de 1 até 6 até encontrar um pai de 6. Os vértices 2 e 4 são os vértices internos percorridos neste caminho. Isso porque $F(1, 6) = 2, W_2(1, 6) = 2, F(2, 6) = 1, W_1(2, 6) = 4$. O algoritmo para quando o vértice 4 é atingido pois ele é um pai de 6. Por último, o algoritmo faz $M(6, 9) \leftarrow M(4, 9)$ pois uma junção dos vértices 4 e 9 foi determinada anteriormente.

3.4.2 Contribuição

Da mesma forma que anteriormente, considere a variação do problema *representante-junção-todos-pares*:

> O problema *representante-junção-k-pares*: Dado um digrafo acíclico D e pares de vértices $\mathcal{P} = \{u_1, v_1\}, \ldots, \{u_k, v_k\}$, determinar uma junção representante para cada par de vértices de $\mathcal{P}$.

No Capítulo 4, veremos que é possível resolver o problema *representante-junção-k-pares* em tempo $o(n^\omega)$ caso $(m + k) = o(n^{\omega-1}) = o(n^{1,373})$.

3.5 Todas Junções em Digrafos Acíclicos

Nesta seção descrevemos algoritmos para o problema *todas-junções-todos-pares*.

> O problema *todas-junções-todos-pares*: Dado um digrafo acíclico D, pré-processe D de tal forma que uma consulta qualquer *todas-junções*(u, v) seja respondida.

Eckhardt, Mühling e Nowak definiram o problema *s-acp-todos-pares* objetivando uma solução para o problema *todos-acps-todos-pares*. Vamos aplicar a mesma ideia definindo o seguinte problema.

> O problema *s-junção-todos-pares*: Dado um digrafo acíclico D e um vértice s, pré-processe D de tal forma que uma consulta qualquer *s-junção*(u, v) seja respondida.

Não temos conhecimento da definição explícita para esses problemas na literatura e tampouco abordagens para resolvê-los. É direto perceber que, se o problema *s-junção-todos-pares* pode ser resolvido em tempo polinomial, então o problema *todas-junções-todos-pares* também pode. Por isso, na próxima seção discutimos três abordagens diretas que resolvem o problema *s-junção-todos-pares* em tempo polinomial.

3.5.1 Algoritmos Polinomiais para o Problema *s-junção-todos-pares*

Nesta seção descrevemos três abordagens simples e diretas que resolvem o problema *s-junção-todos-pares* em tempo polinomial. O leitor que não é familiar com os algoritmos e estruturas de dados desta seção pode consultar os seguintes trabalhos [GT86], [ST84] e [AU72].

A primeira abordagem faz uso de uma redução bem conhecida do problema caminhos vértice-disjuntos para caminhos arco-disjuntos e de um algoritmo que encontra um fluxo máximo em uma rede. Dado um digrafo acíclico D, primeiro construímos um novo digrafo acíclico D'_i para cada par de vértices u_i e v_i. Para um par fixo de vértices u_i e v_i, criamos dois novos vértices v' e v'' e um arco $v' \to v''$ em D'_i para cada vértice v em D e um arco $u'' \to v'$ em D'_i para cada arco $u \to v$ em D. Criamos mais um novo vértice t' e dois novos arcos $u''_i \to t'$ e $v''_i \to t'$. Por último, fixamos capacidade igual a 2 para o arco $s' \to s''$ e capacidade igual a 1 para os arcos restantes. Veja exemplos de dois digrafos na Figura 3.7 supondo que o digrafo de entrada é o da Figura 3.6. Usamos um algoritmo para determinar o valor de um fluxo máximo de s' para t' em D'_i. Se este valor é 2, então existem caminhos vértice-disjuntos de s para u_i e de s para v_i em D. Esta abordagem pode ser implementada usando, por exemplo, o algoritmo de fluxo máximo de Goldberg e Tarjan [GT86] que consome tempo $O(nm \log(n^2/m))$ por digrafo D'_i.

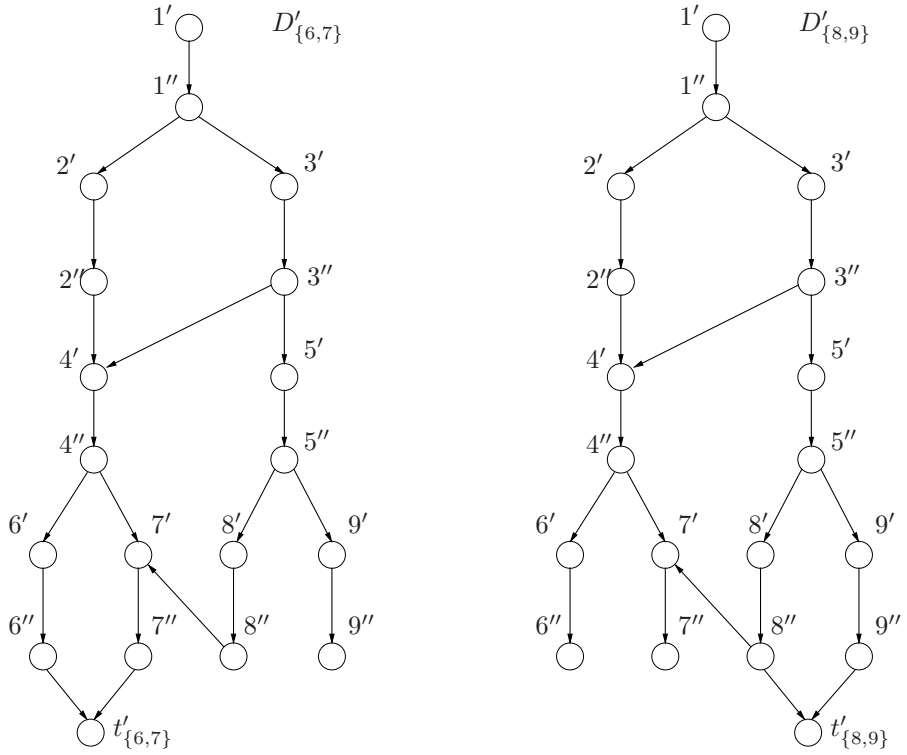


Figura 3.7: Digrafos $D'_{6,7}$ e $D'_{8,9}$ correspondentes ao digrafo da Figura 3.6. Os arcos $1' \to 1''$ têm capacidade igual a 2. Os demais arcos têm capacidade igual a 1.

A segunda abordagem também usa uma redução do problema caminhos vértice-disjuntos para caminhos arco-disjuntos e depois disso usa o algoritmo de Suurballe e Tarjan [ST84]. Dados um digrafo D com custos não negativos nos arcos e um vértice s em D, o algoritmo de Suurballe e Tarjan encontra, para cada vértice v, um par de caminhos arco-disjuntos de s para v tal que a soma dos custos dos caminhos é mínima. Neste caso, construímos um digrafo acíclico D'. Para cada vértice v em D, temos em D' dois vértices v' e v'' e um arco $v' \to v''$. Para cada arco $u \to v$ em D, criamos um arco $u'' \to v'$ em D'. Para cada dois vértices u_i e v_i criamos um vértice t'_i e dois arcos $u''_i \to t'_i$ e $v''_i \to t'_i$. Colocamos custo igual a 1 para todo arco em D'. A Figura 3.7 dá uma ideia de como fica este novo digrafo. Observe que existem caminhos arco-disjuntos de s''

para t'_i em D' se e somente se existem caminhos vértice-disjuntos de s para u_i e de s para v_i no digrafo original. O algoritmo de Suurballe e Tarjan resolve este problema sobre D' em tempo $O(m' \log_{1+m'/n'} n')$, onde n' e m' são os números de vértices e arcos em D'.

A terceira abordagem usa duas estruturas de dados: uma chamada árvores de dominadores (veja Aho e Ullman, [AU72]) e outra capaz de responder consultas $acp(u, v)$ em árvores enraizadas dinâmicas. Primeiro construímos uma árvore de dominadores T enraizada em s em tempo linear considerando o digrafo $D[S]$, onde S é o conjunto dos descendentes de s. Isto pode ser feito usando, por exemplo, o algoritmo de Georgiadis e Tarjan [GT04]. Depois, construímos uma estrutura de dados para consultas $acp_T(u, v)$ em tempo linear. Por último, todas as consultas $acp_T(u_i, v_i)$ possíveis são realizadas. Note que s é uma junção dos vértices u_i e v_i em D se e somente se $s = \mathcal{ACP}^T_{u_i,v_i}$. Essa abordagem produz um algoritmo linear para resolver o problema. Porém, a implementação do algoritmo mais simples para construir uma árvore de dominadores (veja Apêndice C em [AHLT99]) não é tarefa trivial pois é necessário manter uma estrutura de dados para responder consultas $acp(u, v)$ em árvores enraizadas dinâmicas (veja Cole e Hariharan, [CH05]).

3.5.2 Contribuição

A nossa principal contribuição neste trabalho é um algoritmo linear para o problema *s-junção-todos-pares* apresentado no Capítulo 4. Com isso, consumimos tempo $O(n^2)$ para construir uma matriz booleana M^s $(n \times n)$ tal que $M^s(u, v) = 1$ se e somente se s é uma junção dos vértices u e v. Também conseguimos resolver o problema *todas-junções-todos-pares* em tempo $O(n^3)$ e espaço ótimo, isto é, uma lista de junções para cada par de vértices. Observe que qualquer algoritmo que deseja listar todas as junções de todos os possíveis pares de vértices deve consumir tempo $\Omega(n^3)$, no pior caso. Para ver isso, considere o exemplo simples na Figura 3.8. Observe que todos os $\Omega(n^2)$ pares de vértices na segunda linha possui $\Omega(n)$ junções na primeira linha, garantindo assim o limitante inferior no pior caso.

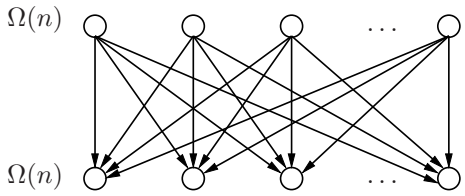


Figura 3.8: Cada vértice na primeira linha é uma junção de todo par de vértices na segunda. Assim, listar todas as junções de todos os $\Omega(n^2)$ pares poderia consumir tempo $\Omega(n^3)$.

Desenvolvemos um algoritmo para o problema *todas-junções-todos-pares* com tempo e espaço ótimos no pior caso. Porém, não temos conhecimento de um algoritmo que consiga resolver este problema em tempo e espaço ótimos em qualquer caso. De maneira mais específica, não conhecemos um algoritmo para este problema cujo tempo seja limitado superiormente pelo tempo do armazenamento explícito de todas as junções de todos os pares de vértices.

No Capítulo 5 veremos uma aplicação da Antropologia e, motivados por essa aplicação, definimos o problema *todas-junções-k-pares*.

> O Problema *todas-junções-k-pares*: Dados um digrafo acíclico D e pares de vértices $\mathcal{P} = \{u_1, v_1\}, \ldots, \{u_k, v_k\}$, determinar todas as junções para cada par de vértices de $\mathcal{P}$.

No próximo capítulo mostraremos como resolver este problema em tempo $O(n(m + k))$ e espaço $O(m + k)$.

3.6 Árvores de Dominadores

Nesta seção, consideramos o seguinte problema.

> O problema *árvore-de-dominadores*: Dado um grafo fluxo D com raiz em s, construir a árvore de dominadores de D.

Lembre da definição de árvore de dominadores na Seção 2.1. Muitos algoritmos foram desenvolvidos para resolver este problema. Citaremos alguns em seguida. Lengauer e Tarjan [LT79] desenvolveram um algoritmo que consome tempo $O(m\alpha(m,n))$, onde α é uma função inversa de Ackermann. Uma implementação mais simples consome tempo $O(m \log n)$. Alstrup, Harel, Lauridsen e Thorup [AHLT99] descreveram um algoritmo linear para o problema. Porém, como os próprios autores descrevem, este resultado é puramente teórico pois é usada uma estrutura de dados chamada *Q-heap* devido a Fredman e Willard [FW94] que requer $n \geq 2^{12^{20}}$ [AHLT99]. Um algoritmo linear implementável foi dado por Georgiadis e Tarjan [GT04]. No entanto, Georgiadis, Tarjan, e Werneck [GTW06] fizeram uma análise experimental não incluindo os algoritmos lineares (inclusive o de Georgiadis e Tarjan [GT04]) alegando que estes algoritmos são complexos e que seria raro algum desses ser na prática mais rápido que os algoritmos $O(m\alpha(m,n))$ e $O(m \log n)$ de Lengauer e Tarjan [LT79]. Assim, podemos supor que um algoritmo linear mais simples e eficiente na prática para o problema ainda é requerido.

Quando a entrada do problema acima é um grafo fluxo redutível, temos o problema *árvore-de-dominadores-gfr*. Qualquer algoritmo para o problema *árvore-de-dominadores* resolve também o problema *árvore-de-dominadores-gfr*. Porém, para o problema *árvore-de-dominadores-gfr* existem algoritmos mais simples. Aho, Hopcroft e Ullman [AHU76] desenvolveram um algoritmo que consome tempo $O(m \log n)$. Ramalingam e Reps [RR94] desenvolveram um algoritmo que utiliza uma estrutura de dados para consultas $acp(u,v)$ em árvores dinâmicas. Se a estrutura de dados de Gabow [Gab90] é usada, então o algoritmo consome tempo amortizado $O(m)$. Se a estrutura de dados de Cole e Hariharan [CH05] é usada, então o algoritmo consome tempo $O(m)$ no pior caso. Infelizmente, implementar qualquer uma destas estrutura de dados não é uma tarefa fácil. Portanto, também para este caso ainda é interessante fornecer um algoritmo linear para o problema que tenha um bom desenvolvimento na prática e que seja de fácil implementação.

Uma aplicação importante para árvore de dominadores ocorre na área de compiladores ([ALSU07], [GT04]). Um grafo fluxo pode ser usado para representar o fluxo de um programa, e os compiladores desejam identificar qual é a estrutura do grafo fluxo com que estão trabalhando encontrando pontos no programa correspondente que podem ser otimizados. Essa fase da compilação de um programa chama-se *otimização do código* [ALSU07]. Também é interessante notar a frequência dos grafos fluxos redutíveis nesta aplicação. Knuth [Knu71] fez um estudo empírico com 50 programas escritos em Fortran e os grafos fluxos correspondentes de todos os programas eram redutíveis. Allen e Cocke [AC72] analisaram 72 programas escritos em Fortran e 5 deles eram irredutíveis. Recentemente, Stanier e Watson [SW12] analisaram 15 projetos da GNU escritos em linguagem C. A análise feita em 2009 considerou os códigos mais antigos e mais atuais disponíveis para cada projeto. Para os códigos mais antigos, 20 funções foram irredutíveis de um total de 4772. Para os códigos mais atuais, 5 foram irredutíveis de um total de 10427. Hecht e Ullman [HU74] também declararam que a classe de grafos fluxos redutíveis é comum na prática quando considerando esta aplicação. Portanto, um algoritmo simples, rápido e de fácil implementação para o problema *árvore-de-dominadores-gfr* é importante para tal aplicação.

Neste texto descrevemos o algoritmo linear de Ramalingam e Reps [RR94] para o problema *árvore-de-dominadores-gfr* devido à sua simplicidade. No entanto, como já dissemos anteriormente, este algoritmo assume uma estrutura de dados sofisticada para responder consultas $acp(u,v)$ em árvores enraizadas dinâmicas em tempo constante. No Capítulo 4, descrevemos e analisamos um algoritmo para este problema que não usa tal estrutura.

3.6.1 Algoritmo de Ramalingam e Reps

Dado um grafo fluxo redutível G com raiz em s, considere o digrafo acíclico D de G e uma ordenação topológica de D. Lembre que as árvores de dominadores T_G de G e T_D de D são iguais. Inicialmente, faça s a raiz da árvore de dominadores de T_D. Em seguida, repita os seguintes passos para cada vértice v na ordem topológica crescente:

1. se v tem um único pai w em D, então $di(v) \leftarrow w$, onde $di(v)$ denota o dominador imediato de v; e

2. se v tem mais que um pai em D, então $di(v) \leftarrow w_k$, onde $v_1, \ldots, v_k$ são os pais de v em D, $w_1 = v_1$, $w_i = acp_{T_D}(w_{i-1}, v_i)$ para $i = 2, \ldots, k$.

A ordenação topológica é feita em tempo linear e para cada passo do algoritmo é feita uma adição de uma folha em T_D. Para cada arco de D é feito no máximo uma consulta acp_{T_D}. Portanto, o algoritmo é linear se usarmos uma estrutura de dados que responde em tempo constante a uma consulta acp_{T_D} em árvores dinâmicas [Gab90], [CH05].

3.6.2 Contribuição

No Capítulo 4, mostramos como construir uma árvore de dominadores de um grafo fluxo redutível em tempo $O(hm)$, onde h é a altura da árvore. Se a altura da árvore é limitada, então temos um algoritmo rápido para o problema.

4

Algoritmos para Junções em Digrafos Acíclicos - Contribuições

Descrevemos neste capítulo um algoritmo linear para resolver um problema que trata de caminhos vértice-disjuntos sobre digrafos acíclicos. O problema, chamado de *s-junção-k-pares*, está definido na Seção 4.1, e a descrição do algoritmo linear está na Seção 4.2. Na Seção 4.3, justificamos as contribuições descritas no capítulo anterior. Na Seção 4.4, aplicamos a ideia usada para resolver o problema *s-junção-k-pares* para resolver o problema *s-arco-junção-k-pares*. Na Seção 4.5, mostramos como usar o algoritmo da Seção 4.2 para obter a multiplicação de duas matrizes booleanas. Na última seção deste capítulo, descrevemos uma relação min-max em grafos fluxos e um algoritmo para construir uma árvore de dominadores de um grafo fluxo redutível. No Apêndice A, encontram-se os pseudocódigos dos algoritmos apresentados neste capítulo.

4.1 Definição e Complexidade do Problema s-junção-k-pares

Dados um digrafo acíclico D, um vértice s em D e k pares de vértices $\{u_1, v_1\}, \ldots, \{u_k, v_k\}$, queremos determinar para quais pares o vértice s é uma junção. Este problema é declarado a seguir.

> O Problema *s-junção-k-pares*: Dados um digrafo acíclico D, um vértice s de D e pares de vértices $\mathcal{P} = \{\{u_1, v_1\}, \ldots, \{u_k, v_k\}\}$, determinar quais pares de vértices em $\mathcal{P}$ têm s como junção.

Este problema pode ser resolvido em tempo $O(m + k)$. Primeiro construa em tempo $O(m)$ [GT04] a árvore de dominadores T de $D[S]$, onde S é o conjunto de descendentes de s em D. A raiz de T é s. Depois construa em tempo $O(m)$ uma estrutura de dados para responder consultas $acp(u, v)$ em T. Finalmente, resolva em tempo $O(1)$ cada consulta $acp(u_i, v_i)$ para $i = 1, \ldots, k$. Quando uma consulta $acp(u_i, v_i)$ devolve s como resposta, temos que s é uma junção dos vértices u_i e v_i.

Em [Tho05], Tholey descreve como resolver o problema *s-junção-k-pares* consumindo tempo $O(n \log^2 n + (m + k) \log_{2+(m+k)/(n+k)} n)$. O autor propõe uma modificação da estrutura de dados criada por Suurballe e Tarjan [ST84]. Recentemente, Tholey [Tho12] publicou um trabalho considerado por ele como uma versão completa de [Tho05]. Uma das contribuições do autor que aparece neste novo trabalho é um algoritmo que resolve o problema *s-junção-k-pares* em tempo linear, porém usando a estrutura de dados árvore de dominadores. Nas próximas seções, apresentamos um algoritmo linear para o problema que não usa tal estrutura deixando o algoritmo mais simples.

4.2 Um Algoritmo Linear para o Problema s-junção-k-pares

Nesta seção apresentamos dois algoritmos que resolvem o problema *s-junção-k-pares*. O primeiro é um algoritmo de programação dinâmica com consumo de tempo cúbico, no pior caso. O segundo é um algoritmo recursivo linear. Apresentamos uma análise de sua correção e complexidade de tempo.

De fato, o algoritmo recursivo constrói uma estrutura de dados sem considerar os k pares de vértices da entrada (veja o Teorema 4.1). Por isso, este algoritmo resolve o problema mais geral *s-junção-todos-pares*. No entanto, desde o princípio do nosso trabalho, estamos interessados em resolver um problema real da Antropologia (veja o Capítulo 5) onde faz mais sentido considerar o problema *s-junção-k-pares*.

No final desta seção, descrevemos um algoritmo que obtém (se existir) dois caminhos vértice-disjuntos de um vértice s para dois vértices u e v (baseado em um algoritmo descrito em [ST84]).

Ao longo desta seção, considere um digrafo acíclico D e um vértice s de D. Começamos com a seguinte recorrência.

Lema 4.1. *Seja T^s uma árvore geradora de busca em profundidade (agbp) de D com raiz em s. Sejam u e v vértices descendentes de s. Denote por $z = \mathcal{ACP}^{T^s}_{u,v}$ o acp em T^s de u e v. Então, o vértice s é uma junção dos vértices u e v se e somente se vale uma das seguintes afirmações:*

1. *u é igual a s ou v é igual a s.*
2. *u está em T^{s_1} e v está em T^{s_2}, onde s_1 e s_2 são dois filhos distintos de s em T^s, e T^{s_1} e T^{s_2} são subárvores de T^s com raízes em s_1 e s_2.*
3. *u e v estão em uma mesma subárvore (digamos T^{s_1}), z não é igual a u ou a v e, ou s é uma junção de z e u, ou s é uma junção de z e v.*
4. *u e v estão em uma mesma subárvore (digamos T^{s_1}), z é igual a u (ou v) e s é uma junção de z e t para algum t em $\delta^-_D(v)$ (ou $\delta^-_D(u)$) e t diferente de z.*

Demonstração: Se o primeiro caso é verdade, então, s é uma junção degenerada do par. Para o segundo caso, observe que os caminhos em T^s de s para u e para v fornecem um certificado para o fato de que s é uma junção. O terceiro caso é demonstrado pelo Lema 4.3 e Corolário 4.1. O último caso é demonstrado pelo Lema 4.4.

Agora, considere que nenhum caso é verdade, isto é, s não é uma junção degenerada, u e v estão em uma mesma subárvore e as condições 3 e 4 são falsas. Então, s não pode ser uma junção de u e v. □

Antes de demonstrar os Lemas 4.3 e 4.4, e o Corolário 4.1, note a seguinte propriedade de uma árvore geradora de busca em profundidade.

Lema 4.2. *Considere uma agbp T^s de D, um filho s_1 de s em T^s e um caminho R de s para u em D, onde u é um descendente de s_1 em T^s. Suponha que x é o primeiro vértice em R que está na subárvore T^{s_1}. Assim, todos os vértices do caminho $R[x,u]$ pertencem à subárvore T^{s_1} de T^s.* □

Adicionalmente, se s é uma junção dos vértices u e v e os vértices u e v estão em T^{s_1}, então existe um par de caminhos vértice-disjuntos em D de s para u e de s para v, tal que todos os vértices internos de um dos caminhos estão em T^{s_1}. Podemos demonstrar isso por contradição. Considere dois caminhos internamente vértice-disjuntos em D, P de s para u e Q de s para v (veja Figura 4.1). Suponha que ambos P e Q não têm todos os vértices em T^{s_1}. Considere os primeiros vértices de P e de Q que pertencem a T^{s_1}, x em P e y em Q. Sabemos que os caminhos $P[x,u]$ e $Q[y,v]$ são internos à subárvore T^{s_1}. Considere o caminho P' de s_1 para x em T^{s_1}. Se não existe vértice comum nos caminhos P' e $Q[y,v]$, então os caminhos $sP'P[x,u]$ e Q são internamente vértice-disjuntos e todos os vértices internos de um deles estão em T^{s_1}. Portanto, P' e $Q[y,v]$ têm pelo menos um vértice comum. Seja v' o vértice comum aos caminhos P' e $Q[y,v]$ mais próximo de v. Analogamente, considere o caminho Q' de s_1 para y em T^{s_1}. Se ele é disjunto ao caminho $P[x,u]$, então os caminhos $sQ'Q[y,v]$ e P são caminhos internamente vértice-disjuntos e todos os vértices internos de um deles estão em T^{s_1}. Portanto, Q' e $P[x,u]$ têm pelo menos um vértice comum. Seja u' o vértice comum mais próximo de u.

Assim, encontramos um ciclo com os caminhos de y para v', de v' para x, de x para u' e de u' para y. Isto é uma contradição, pois D é um digrafo acíclico. Portanto, todo vértice interno de P ou de Q estão em T^{s_1}. Agora podemos mostrar o seguinte lema.

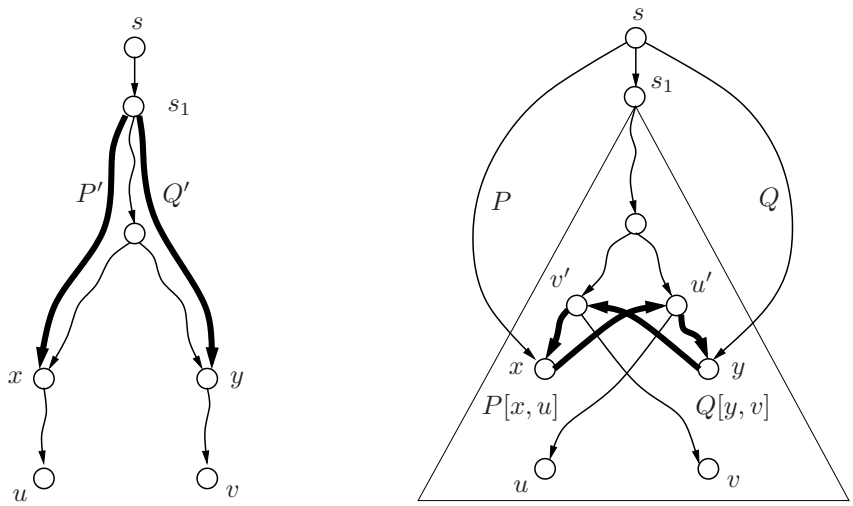


Figura 4.1: Uma agbp e um ciclo composto pelos caminhos de y para v', de v' para x, de x para u' e de u' para y. Isso não pode ocorrer em um digrafo acíclico.

Lema 4.3. *Considere u e v vértices na subárvore T^{s_1} de T^s, onde s_1 é um filho de s em T^s. Denote por z o acp de u e v em T^s e considere que z não é igual a u ou v. O vértice s é uma junção dos vértices u e v se e somente se z pertence a um par de caminhos internamente vértice-disjuntos de s para u ou de s para v.*

Demonstração: Como s é uma junção de u e v, tome um par de caminhos internamente vértice-disjuntos P de s para u e Q de s para v. Suponha que todos os vértices internos de P estão na subárvore T^{s_1} e Q tem um arco externo entrando na subárvore T^{s_1}. Considere R o caminho de s para z em T^s. Sejam P' e Q' os caminhos em T^s de z para u e de z para v, respectivamente (veja Figura 4.2 (a)). Aqui é importante notar que, pela construção de T^s, qualquer caminho que intercepta um dos caminhos P' ou Q' e depois intercepta o outro, não pode voltar a interceptar o primeiro caminho. Vamos dividir a demonstração em dois casos:

Caso 1. Não existe vértice de Q em R. Se Q é também disjunto de P', então os caminhos RP' e Q são internamente vértice-disjuntos (veja Figura 4.2 (b)). Se existe algum vértice de Q em P', então considere x o vértice comum de Q e P' mais próximo de z. Então, os caminhos $Q[s,x]P'[x,u]$ e RQ' são internamente vértice-disjuntos (veja Figura 4.2 (c)).

Caso 2. Existem vértices de Q em R. Neste caso, R poderia ter vértices em P, em Q ou em ambos. De todos os vértices de R que estão em P ou Q, considere que y é o vértice mais próximo ao vértice z. Sem perda de generalidade, considere que y está em Q (o caso quando y está em P é simétrico).

Caso 2.1. Não existe vértice de P em Q'. Então, os seguinte caminhos são internamente vértice-disjuntos: P e $Q[s,y]R[y,z]Q'$ (veja Figura 4.3 (b)).

Caso 2.2. Existem alguns vértices de P em Q'. Considere x, o vértice comum em P e Q' mais próximo ao vértice z. Então, os seguintes caminhos são internamente vértice-disjuntos: $Q[s,y]R[y,z]P'$ e $P[s,x]Q'[x,v]$ (veja Figura 4.3 (c)).

Portanto, se s é uma junção dos vértices u e v, então existe um par de caminhos internamente vértice-disjuntos com z pertencendo a um deles. Isso termina a primeira parte da demonstração. A demonstração do outro lado é trivial. Por definição, s é uma junção dos vértices u e v se existe um par de caminhos internamente vértice-disjuntos, independentemente se z pertence a um deles. □

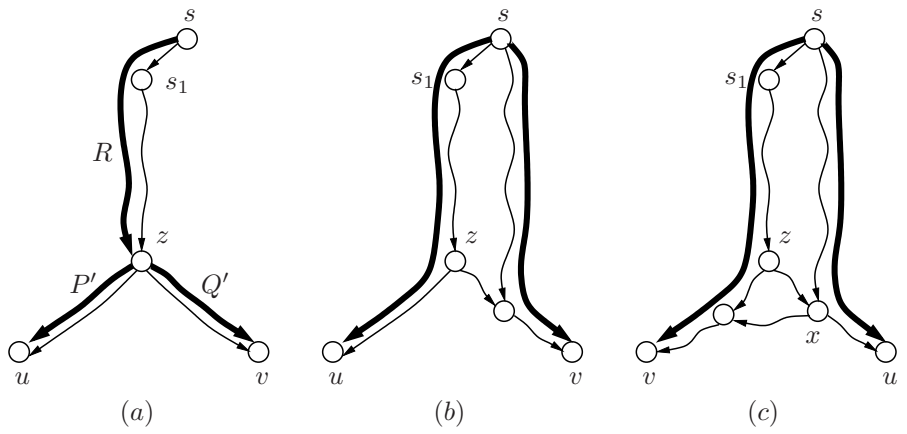


Figura 4.2: Uma agbp e novos caminhos internamente vértice-disjuntos – Lema 4.3. Caso 1. Um dos caminhos possui o vértice $z = \mathcal{ACP}^{T^s}_{u,v}$.

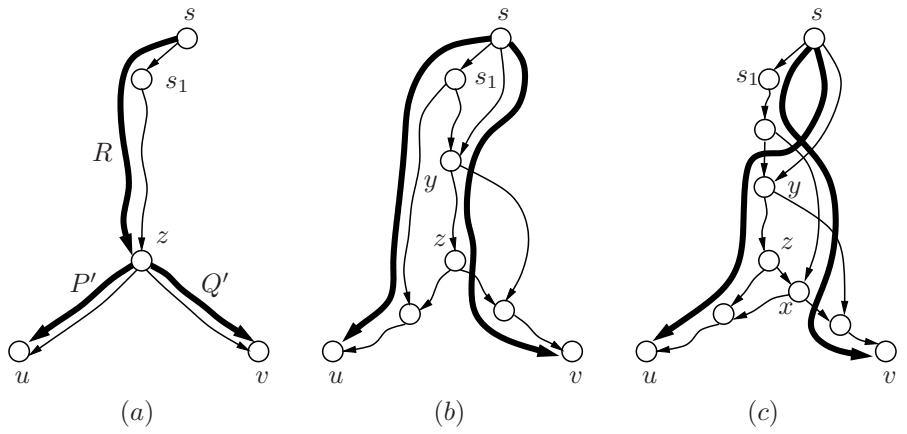


Figura 4.3: Uma agbp e novos caminhos internamente vértice-disjuntos – Lema 4.3. Caso 2. Um dos caminhos possui o vértice $z = \mathcal{ACP}^{T^s}_{u,v}$. O vértice $y \in Q$ é o mais próximo de z em $(P \cup Q) \cap R$.

Com isso, se existem caminhos internamente vértice-disjuntos de s a u e de s a v, então existe um par desses caminhos em que z pertence a um deles. Uma consequência do Lema 4.3 é o seguinte corolário.

Corolário 4.1. *Sejam u e v vértices na subárvore T^{s_1} de T^s, onde s_1 é um filho de s em T^s. Chame de z o acp de u e v na subárvore T^{s_1} e considere que z não é igual a u ou v. O vértice s é uma junção dos vértices u e v se e somente se s é uma junção dos vértices z e u ou s é uma junção dos vértices z e v.*

Demonstração: Considere um par de caminhos internamente vértice-disjuntos P de s para u e Q de s para v tal que z está em P ou Q como dado pelo Lema 4.3. Se z está em Q, então os caminhos $Q[s, z]$ e P são internamente vértice-disjuntos. Portanto, s é uma junção dos vértices z e u. Se z está em P, então os caminhos $P[s, z]$ e Q são internamente vértice-disjuntos. Portanto, s é uma junção dos vértices z e v.

Para provar o outro lado, considere um par de caminhos internamente vértice-disjuntos P de s para z e Q de s para v. Considere P' e Q' os caminhos em T^s de z para u e de z para v, respectivamente. Mais uma vez, dividimos a demonstração em dois casos:

Caso 1. Não existe vértice de Q em P'. Portanto, os caminhos PP' e Q são internamente

vértice-disjuntos (veja Figura 4.4 (b)).

Caso 2. Existem vértices de Q em P'. Como D é um digrafo acíclico, não existe vértice interno de P em P'. Seja x o vértice comum em Q e P' mais próximo de z. Assim, os caminhos $Q[s,x]P'[x,u]$ e PQ' são internamente vértice-disjuntos (veja Figura 4.4 (c)).

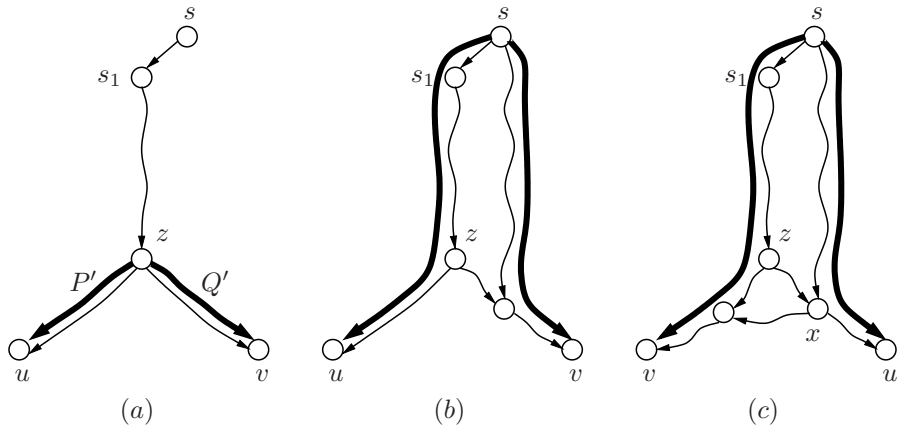


Figura 4.4: Uma agbp e novos caminhos internamente vértice-disjuntos – Corolário 4.1. Casos 1 e 2.

Com uma demonstração similar podemos mostrar que é possível construir um par de caminhos vértice-disjuntos de s para u e de s para v quando pegamos um par de caminhos vértice-disjuntos de s para u e de s para z. Portanto, se s é uma junção dos vértices z e u ou s é uma junção dos vértices z e v, então s é uma junção dos vértices u e v. □

O corolário anterior caracteriza s como uma junção dos vértices u e v quando o acp de u e v em T^s é diferente de u e de v. No próximo lema consideramos o caso onde o acp é igual a um dos vértices.

Lema 4.4. *Considere z e u vértices na subárvore T^{s_1} de T^s e considere z um ancestral próprio de u em T^s, isto é, z é o acp de z e u em T^s e z é diferente de u. O vértice s é uma junção dos vértices z e u se e somente se s é uma junção dos vértices z e t, para algum t em $\delta_D^-(u)$ e t diferente de z.*

Demonstração: Para demonstrar a primeira parte, consideramos dois caminhos internamente vértice-disjuntos Q de s para z e P de s para u. Considere o último vértice t antes de u em P. Sabemos que t está em $\delta_D^-(u)$ pois ele é um pai de u em D e t é diferente de z pois Q e P são caminhos vértice-disjuntos. Portanto, o caminho $P[s,t]$ e Q são vértice-disjuntos, e então s é uma junção dos vértices z e t.

Para demostrar o outro lado, consideramos dois caminhos internamente vértice-disjuntos P de s para t e Q de s para z. Se u está em P, então o subcaminho $P[u,t]$ mais o arco $t \to u$ é um ciclo. Contudo D é um digrafo acíclico. Assim, u não está em P. Se u está em Q, então u é um ancestral próprio de z. Por hipótese, z é um ancestral próprio de u. Assim, podemos novamente produzir um ciclo contradizendo o fato de que D é um digrafo acíclico. Então, u também não está em Q. Como t é diferente de z, podemos usar o arco $t \to u$ estendendo o caminho P e construindo dois caminhos internamente vértice-disjuntos P mais o arco $t \to u$ e Q. Portanto, s é uma junção dos vértices z e u. □

4.2.1 Algoritmos de Programação Dinâmica e Recursivo

Uma matriz binária e quadrada M^s $(n \times n)$ é uma forma natural para representar todos os pares de vértices que têm um vértice s como uma junção. O vértice s é uma junção dos vértices u e v se e somente se $M^s(u,v) = M^s(v,u) = 1$. Assim, podemos descrever um algoritmo de

programação dinâmica que considera uma agbp T^s e uma ordenação topológica dos vértices de D para preencher a matriz M^s. Primeiro, consideramos que os elementos da matriz M^s foram inicializados com 0. Depois, $M^s(s,u) = M^s(u,s) = 1$ para cada vértice u em D. Finalmente, considere o vértice $z = \mathcal{ACP}^{T^s}_{u,v}$ para cada par de vértices u e v em D tal que u precede v na ordem topológica. Faça $M^s(u,v) = M^s(v,u) = 1$ se $z = s$ (isso implica $u \in T^{s_1}$ e $v \in T^{s_2}$); ou se $z \neq u$ e ($M^s(z,u) = 1$ ou $M^s(z,v) = 1$); ou se $z = u$ e $M^s(z,t) = 1$ para algum $t \in \delta^-_D(v)$. Nessa última parte usamos os resultados do Lema 4.1. Infelizmente, este algoritmo poderia gastar tempo $\Omega(n^3)$. Veja na Figura 4.5 que muitos arcos poderiam ser processados por um número quadrático de pares de vértices.

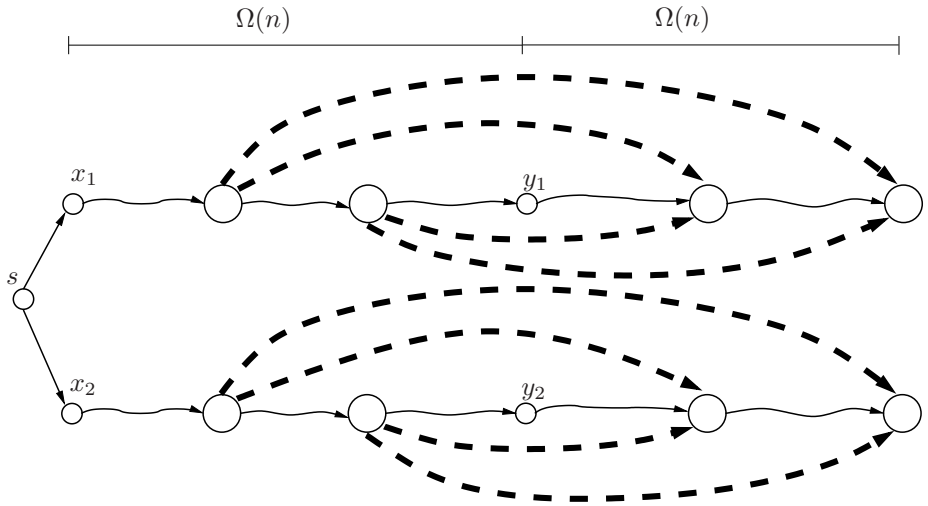


Figura 4.5: Existem $\Omega(n)$ vértices nos caminhos de x_1 para y_1 e em T^{y_1} (resp. x_2, y_2 e T^{y_2}). Existe um arco ligando todo vértice destes caminhos para todos os descendentes próprios de y_1 (resp. y_2) representado pelos arcos mais largos e tracejados ligando os vértices maiores. Para um número quadrático de descendentes próprios de y_1 (resp. y_2), o algoritmo de programação dinâmica descrito anteriormente processa $\Omega(n)$ arcos. Neste caso, o tempo total gasto é $\Omega(n^3)$.

Contudo, podemos fornecer uma solução melhor para o problema. Primeiro apresentamos uma representação compacta para a matriz M^s usando um vetor unidimensional. O algoritmo que resolve o problema *s-junção-k-pares* tem duas fases:

Fase 1. Construir em tempo $O(m)$ uma partição dos descendentes de s, $\mathcal{B}^s$, que obedece a seguinte propriedade: *Dois vértices u e v estão em subconjuntos diferentes de $\mathcal{B}^s$ se e somente se s é uma junção de u e v.*

Fase 2. Resolver em tempo $O(1)$ as consultas *s-junção*(u_i, v_i) para $i = 1, \ldots, k$.

Denotamos este algoritmo por **SJKP**. O tempo total gasto por ele para resolver o problema *s-junção-k-pares* é $O(m + k)$, isto é, linear no tamanho da entrada. Vimos no capítulo anterior que uma representação matricial é usada em vários problemas envolvendo acps e junções. Se tal representação é requerida, então pode-se usar a partição $\mathcal{B}^s$ para obtê-la em tempo $O(n^2)$.

Agora mostramos como resolver a **Fase 1**. A estrutura de dados usada para representar a partição $\mathcal{B}^s$ como descrita acima é um vetor r, similar à estrutura de dados para representar subconjuntos disjuntos (veja Cormen, Leiserson, Rivest e Stein [CLRS09]). Cada conjunto de $\mathcal{B}^s$ tem um vértice representante, e para tais vértices $r(v) = v$. Para os vértices restantes $r(v)$ aponta para um outro vértice no mesmo conjunto da partição. Estas ligações conduzem para o representante de cada conjunto. O algoritmo começa construindo uma agbp T^s de $D[S]$, onde S é o conjunto dos descendentes de s. O vetor r é então inicializado de tal forma que cada vértice

v aponta para seu pai em T^s, e o vértice raiz s tem $r(s) = s$. Isso representa uma partição com um único conjunto e s é o representante (veja a Figura 4.6 (b)). Durante a construção de T^s cada vértice v recebe um rótulo $pós(v)$ com o número do vértice em uma varredura pós-ordem de T^s. Além disso, é armazenado para cada vértice v o valor $minpós(v)$ que é o valor pós-ordem mínimo de um vértice na subárvore T^v de T^s. Não é difícil construir uma agbp e manter estes valores. O leitor interessado pode consultar o livro de Sedgewick [Sed02]. O algoritmo para resolver a **Fase 1** é recursivo. Em cada chamada ele recebe um vértice representante z e modifica corretamente os valores de r para os vértices em T^z de T^s. As chamadas iniciais para o algoritmo recursivo são feitas sobre cada filho s_j de s em T^s, $r(s_j) = s_j$ e $z = s_j$. Em cada chamada recursiva os seguintes casos são analisados:

Caso 1: T^z tem um único vértice. Nada deve ser feito no vetor r.

Caso 2: T^z tem pelo menos dois vértices. Considere que w é o vértice tal que $pós(w) = pós(z) - 1$. Observe que w deve ser inicialmente um filho de z. Considere todos os vértices t em $\delta_D^-(w)$. Se, para algum desses vértices, $r(z) \neq r(t)$ (isto é, s é uma junção dos vértices z e t) podemos modificar w tornando-o representante, resolver o problema recursivamente para a subárvore T^w e, depois do fim da chamada recursiva, atualizar w para o vértice cujo valor pós-ordem é $minpós(w) - 1$. No outro caso, $r(z) = r(t)$ para todo t em $\delta_D^-(w)$, então (pelo Lema 4.4) podemos fazer $r(w)$ igual a z e atualizar w para o vértice com valor pós-ordem igual a $pós(w) - 1$. Isso é feito até w ser um vértice fora de T^z, isto é, $pós(w) < minpós(z)$.

Note que a **Fase 1** do algoritmo SJKP explora um arco de $D[S]$ no máximo duas vezes (uma vez na construção de T^s e no máximo uma vez no **Caso 2** do algoritmo recursivo). Portanto, o tempo gasto por ela é linear em m. A Figura 4.6 (c) ilustra a partição $\mathcal{B}^s$ para o digrafo da Figura 4.6 (a).

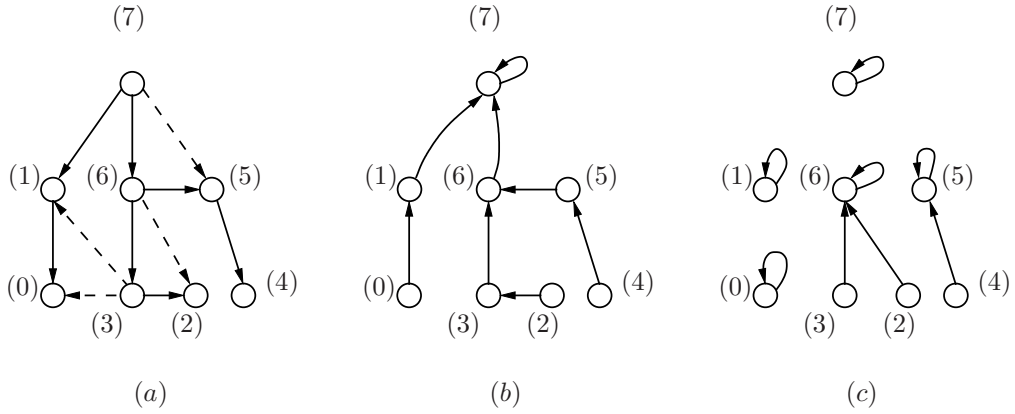


Figura 4.6: Um digrafo acíclico, uma agbp, e a primeira e a última configuração da partição dos vértices. O valor pós-ordem de cada vértice está entre parênteses.

Este algoritmo fornece um certificado que prova que s não é uma junção dos vértices u e v, quando este for o caso. Na Seção 4.6, veremos que se $p = r(u) = r(v)$, então p é um dominador dos vértices u e v. Com isso, p é o vértice que certifica que s não é junção de u e v pois p separa s de u e s de v.

Agora mostramos que o algoritmo está correto. Note que $r(u)$ é alterado no máximo uma vez para todo u descendente de s. A recursão considera um caminho Z de z para w em T^z. Uma importante observação é que $r(w') = z$ para todo w' em Z. Portanto temos que mostrar que s não é uma junção dos vértices w' e w'' para todo w' e w'' em Z. Uma demonstração segue se assumimos que s não é uma junção dos vértices z e w' para todo w' em Z.

Lema 4.5. *Considere Z o caminho de z para w em T^s. Se s não é uma junção dos vértices z e w' para todo w' em Z, então s não é uma junção dos vértices w' e w'' para todo w' e w'' em Z.*

Demonstração: A demonstração é por contradição. Suponha que s é uma junção dos vértices w' e w'' onde ambos os vértices estão no caminho Z. Considere P e Q, os caminhos vértice-disjuntos de s para w' e de s para w'', respectivamente. Considere o caminho R de s para z em T^s. Considere os vértices de R que são comuns aos caminhos P ou Q, e considere z' um tal vértice mais próximo de z. Suponha que z' está em Q (o outro caso, quando z' está em P, é análogo). Então, os caminhos $Q[s, z']R[z', z]$ e P são internamente vértice-disjuntos. Portanto, s é uma junção dos vértices z e w', uma contradição. □

Considere novamente o caminho Z em T^z de z para w. Agora, suponha que s não é uma junção dos vértices z e w' para todo w' em Z diferente de w. Contudo, s é uma junção dos vértices z e w. Do algoritmo anterior, $r(u)$ é diferente de $r(v)$ para todo vértice u em $T^z \setminus T^w$ e v em T^w. Isto implica que s é uma junção dos vértices u e v. O próximo lema demonstra isso em dois passos.

Lema 4.6. *Se s não é uma junção dos vértices z e w' para todo w' em Z diferente de w, e s é uma junção dos vértices z e w, então*

a. *s é uma junção dos vértices w' e w para todo w' em Z; e*

b. *s é uma junção dos vértices u e v para todos os vértices u em $T^z \setminus T^w$ e v em T^w.*

Demonstração: **a.** Considere os caminhos P de s para w e Q de s para z que são internamente vértice-disjuntos. Sabemos que o único vértice comum em Q e Z é z, pois Z contém descendentes de z e Q contém ancestrais de z. Note também que o único vértice comum em P e Z é w. Se esse não é o caso, então s é uma junção dos vértices z e w' para algum w' diferente de w em Z, contradizendo nossa hipótese. Portanto podemos estender o caminho Q usando o caminho Z de z até qualquer w' em Z e então s é uma junção dos vértices w' e w pois os caminhos $QZ[z, w']$ e P são internamente vértice-disjuntos (veja a Figura 4.7 (a)).

b. Considere u em $T^z \setminus T^w$ e v em T^w. Considere w'' o acp em T^s dos vértices u e v. Pelo Lema 4.6 a, temos, em particular, que s é uma junção de w'' e w. Portanto, existem caminhos internamente vértice-disjuntos P de s para w'' e Q de s para w. Note que podemos estender o caminho Q adicionando o caminho sobre T^w de w para v e este novo caminho, denotado por R, não intersecta P ou $Q \setminus \{w\}$. Caso contrário, existiria um ancestral e um descendente de w. Assim, o único vértice comum em R e P é s. Portanto, s é uma junção de w'' e v. Se u é igual ao w'', então acabou. Se u é diferente de w'', então pelo Corolário 4.1, temos que s é uma junção de u e v (veja a Figura 4.7 (b)). □

Ainda considerando o algoritmo recursivo descrito anteriormente, podemos rapidamente demonstrar algumas propriedades sobre o vértice z e os vértices que não estão em T^z.

Proposição 4.1. *Considere uma chamada para o procedimento recursivo onde inicialmente $z = s_i$ para algum filho s_i de s em T^s. Em qualquer chamada recursiva (z é o parâmetro) derivada desta chamada inicial, os seguintes invariantes são verdadeiros:*

- *O vértice z é um representante,*
- *Para todo vértice $x \in T^{s_i} \setminus T^z$, $r(z) \neq r(x)$.*

Demonstração: O primeiro invariante é verdadeiro pois na primeira chamada $z = s_i$ é um representante e antes de qualquer chamada recursiva do procedimento passando w como parâmetro, fazemos $r(w) = w$.

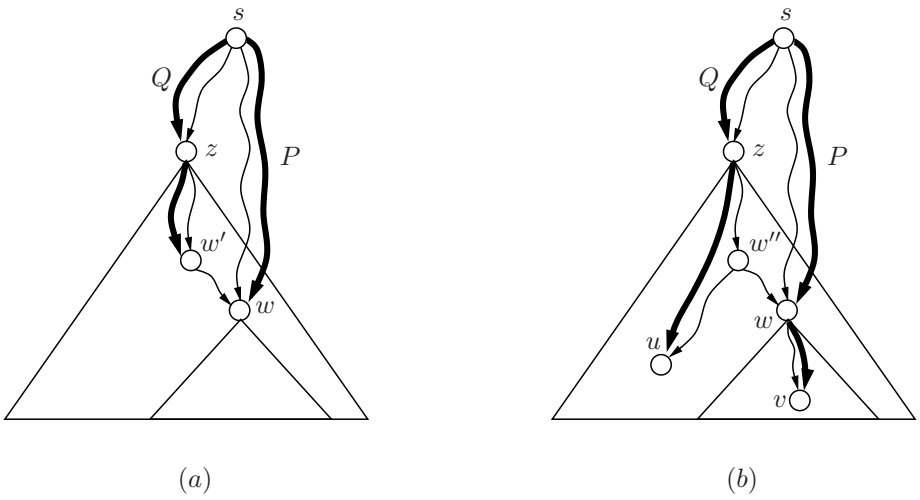


Figura 4.7: Novos caminhos internamente vértice-disjuntos – Lema 4.6.

O segundo invariante é verdadeiro pois para qualquer vértice $v \in T^s$, $r(v)$ aponta para um representante que é seu ancestral em T^s. Pelo primeiro invariante, $r(z) = z$. Mas o vértice z não é um ancestral de x em T^s. Portanto, $r(z) \neq r(x)$. □

Agora vamos demonstrar que o algoritmo recursivo está correto.

Lema 4.7. *Considere uma chamada para o procedimento recursivo onde inicialmente z é um filho s_i de s em T^s. Para todo vértice u e v em T^{s_i} tal que* $\min\{pós(u), pós(v)\} \geq pós(w) + 1$ *os seguintes invariantes são verdadeiros:*

- *$r(u)$ e $r(v)$ apontam para um representante,*
- *s é uma junção dos vértices u e v se e somente se $r(u) \neq r(v)$.*

Demonstração: O primeiro invariante é verdadeiro pois $r(w)$ aponta ou para z ou para w (quando ele se torna um representante).

Finalmente, o último invariante é demonstrado por indução no número de comparações "$pós(w) \geq minpós(z)$" realizadas. Na primeira comparação temos $z = s_i$ e w é o filho de maior valor pós-ordem, $pós(w) = pós(s_i) - 1$. Isso significa que s_i é o único vértice em T^{s_i} com valor pós-ordem maior que $pós(w)$. O vértice s não é uma junção dos vértices s_i e s_i e $r(s_i) = s_i$. Agora suponha que este invariante é verdade na i-ésima comparação $pós(w) \geq minpós(z)$, e vamos demonstrar que este invariante continua verdade na próxima comparação. Considere o caminho Z em T^z de z para w na i-ésima comparação. Este caminho foi formado pelo algoritmo nas iterações anteriores. Pela ordem que os vértices são processados e pelo Lema 4.4, sabemos que s não é uma junção dos vértices z e w' para todo w' em Z diferente de w. Sabemos que, $\min\{pós(z), pós(t)\} \geq pós(w) + 1$, para todo z e para todo t em $\delta_D^-(w)$. Assim o invariante é verdade para o par de vértices z e t, e portanto s é uma junção de z e t se e somente se $r(z) \neq r(t)$. Aplicando o Lema 4.4 temos duas possibilidades:

Caso 1. s não é uma junção dos vértices z e t, para todo t em $\delta_D^-(w)$. Então, $r(z) = r(t)$, para todo t em $\delta_D^-(w)$ e pelo Lema 4.4, s não é uma junção dos vértices z e w. Neste caso, o algoritmo faz $r(w) = z$. Tome x em T^{s_i} com $pós(x) \geq pós(w) + 1$. Denote por y o acp em T^s dos vértices w e x (possivelmente $x = y$). Vamos mostrar que exatamente antes de atualizar w para o próximo vértice, s é uma junção dos vértices x e w se e somente se $r(x) \neq r(w)$. Suponha que x está em $T^{s_i} \setminus T^z$. Então, y é algum vértice no caminho de s_i para z em T^{s_i} e, pelo segundo invariante da Proposição 4.1, $r(y) \neq r(z)$ e $r(x) \neq r(z) = z = r(w)$. Portanto,

temos que mostrar que s é uma junção dos vértices x e w. Sabemos que s é uma junção dos vértices y e z pois $r(y) \neq r(z)$. Como D é um digrafo acíclico, sabemos que s é uma junção dos vértices y e w porque podemos usar o caminho de z para w em T^{s_i}. Se $y = x$, então acabou. Caso contrário, pelo Corolário 4.1, temos que s é uma junção dos vértices x e w.

Agora suponha que x está em T^z. Então o vértice y está em Z, isto é, o acp de w e x em T^s está em Z (possivelmente $x = y$).

Caso 1.1. s não é uma junção dos vértices x e z. Neste caso, $r(x) = r(z)$. Além disso, $r(w) = z = r(z) = r(x)$. Então temos que mostrar que s não é uma junção dos vértices x e w. Como mencionado, s não é uma junção dos vértices z e w', onde w' está em Z. Pelo Lema 4.5, em particular, s não é junção dos vértices y e w. Assim, $r(y) = r(w)$. Portanto, se $x = y$, então s não é uma junção dos vértices x e w. Se $x \neq y$, então por indução e porque $r(y) = r(w) = r(x)$, s não é uma junção dos vértices y e x. Finalmente, pelo Corolário 4.1, s não é uma junção dos vértices x e w porque s não é uma junção dos vértices y e w, e y e x.

Caso 1.2. s é uma junção dos vértices x e z. Então, $r(x) \neq r(z)$. Além disso, $r(w) = z = r(z) \neq r(x)$. Então temos que mostrar que s é uma junção dos vértices x e w. Como y está em Z, sabemos que s não é uma junção dos vértices z e y. Então nosso invariante diz que $r(z) = r(y)$. Portanto, $r(y) = r(z) \neq r(x)$. Isso significa que s é uma junção dos vértices y e x. Pelo Corolário 4.1, s é uma junção dos vértices x e w.

Depois disso, o algoritmo atualiza w para ser o próximo vértice com valor pós-ordem igual a $pós(w) - 1$ e então nosso invariante é restaurado.

Caso 2. s é uma junção dos vértices z e t, para algum t em $\delta_D^-(w)$. Então, $r(z) \neq r(t)$, para algum t em $\delta_D^-(w)$ e pelo Lema 4.4, s é uma junção dos vértices z e w. Neste caso, o algoritmo faz $r(w) = w$. Tome $x \in T^{s_i}$ com $pós(x) \geq pós(w) + 1$. Vamos mostrar que exatamente antes de realizar a chamada recursiva, s é uma junção dos vértices x e w se e somente se $r(x) \neq r(w)$. Como w não é ancestral de x em T^s e $r(x)$ aponta para um representante que é seu ancestral em T^s, o valor $r(x)$ é diferente do valor $r(w)$. Assim, temos que mostrar que s é uma junção dos vértices x e w. Se x está em $T^z \setminus T^w$, então o Lema 4.6 b garante que s é uma junção dos vértices x e w. Se x está em $T^{s_i} \setminus T^z$, então y, o ancestral comum mais próximo dos vértices x e w em T^s, é um vértice no caminho de s_i para z em T^{s_i} (possivelmente $x = y$). Pelo segundo invariante na Proposição 4.1, $r(y) \neq r(z)$. Assim, s é uma junção dos vértices y e z, e podemos usar o caminho de z para w em T^{s_i} para dizer que s é uma junção dos vértices y e w. Se $x = y$, então acabou. Se, $x \neq y$, então, pelo Corolário 4.1, s é uma junção dos vértices x e w. Depois da chamada recursiva, temos que s é uma junção dos vértices u e v se e somente se $r(u) \neq r(v)$ para todos os vértices u e v em T^{s_i} tais que seus valores pós-ordem são maiores ou iguais ao valor pós-ordem do vértice $minpós(w)$. Depois disso, fazemos w ser o vértice com valor pós-ordem igual a $minpós(w) - 1$ e então nosso invariante é restaurado. □

Agora vamos considerar a **Fase 2**, isto é, queremos resolver em tempo $O(1)$ as consultas *s-junção*(u_i, v_i) para $i = 1, \ldots, k$. Isto pode ser feito verificando os valores do vetor r, lembrando que s é uma junção de u_i e v_i se e somente se $r(u_i) \neq r(v_i)$. Agora podemos declarar o seguinte resultado.

Teorema 4.1. *Dados um digrafo acíclico D com n vértices e m arcos, e um vértice s em D, podemos construir em tempo $O(m)$ uma estrutura de dados que responde em tempo constante para qualquer par de vértices u e v em D, se s é uma junção dos vértices u e v. Além disso, depois de tal construção, todos os k pares de vértices que têm s como uma junção podem ser obtidos em tempo $O(k)$.*

Demonstração: A demonstração de que a construção da estrutura de dados é feita em tempo $O(m)$ e a resposta à consulta *s-junção*(u, v) é feita em tempo $O(1)$ foi descrita ao longo deste capítulo. Falta mostrar como obter todos os pares de vértices que têm s como junção em tempo $O(k)$, supondo haver k pares de vértices com s como uma junção. Para isso, basta manter os vértices representantes e combinar todos os pares u e v com $r(u) \neq r(v)$. □

Vamos supor que nos é dado um digrafo acíclico D com um vértice fonte s tal que existe um caminho de s para qualquer outro vértice de D, isto é, D é um grafo fluxo com raiz em s. Vamos chamar a estrutura de dados descrita anteriormente (o vetor r) de *partição de D em estrelas*. O problema *s-junção-k-pares* com D como entrada pode ser resolvido em tempo $O(m+k)$: $O(m)$ para construir uma partição de D em estrelas e $O(1)$ para consultar se s é ou não junção para cada par.

4.2.2 Obtendo um Par de Caminhos Vértice-Disjuntos

Dado um par de vértices u e v vamos mostrar como obter dois caminhos internamente vértice-disjuntos de s para u e de s para v. A ideia é adaptar o algoritmo de Suurballe e Tarjan [ST84]. Os autores mostram como obter caminhos arco-disjuntos de um vértice fixo s para qualquer outro vértice u. Aqui queremos obter caminhos internamente vértice-disjuntos de um vértice fixo s para qualquer par de vértices u e v, inclusive quando $u = v$.

Antes de descrever como obter tais caminhos, devemos alterar o algoritmo recursivo descrito anteriormente. Primeiro vamos considerar dois novos vetores indexados pelos vértices e denotados por r' e q. O vetor r' é um vetor booleano. O valor $r'(u)$ é igual a 1 se e somente se existem caminhos internamente vértice-disjuntos de s a u. O vetor r' pode ser obtido da seguinte forma. Inicialize $r'(u) = 0$ para todo u em T^s e $r'(s) = 1$. Para cada filho s_i de s em T^s, faça $r'(s_i) = 1$ se existir um arco $t \to s_i$ onde $t \neq s$. Antes de cada chamada recursiva do algoritmo SJKP, faça $r'(w) = 1$. Note que uma chamada recursiva foi causada porque encontramos $r(z) \neq r(t)$, ou seja, s é uma junção de z e t. Pelo resultado do Lema 4.6 a (quando $w' = w$) sabemos que existem caminhos vértice-disjuntos de s para o par w e w. Vamos abusar da linguagem e dizer que s é uma junção dos vértices w e w. Agora descrevemos o vetor q. No momento em que descobrirmos que o vértice u é um representante, armazenamos em $q(u)$ um vértice t (pai de u em D) que causou tal propriedade (u ser um representante). No início, armazenamos um vértice "falso" em todas as posições de q. Depois armazenamos s em $q(s)$ e em $q(s_i)$. Se $r'(s_i) = 1$, então existe um arco $t \to s_i$, onde $t \neq s$ e assim $q(s_i)$ vale t. No algoritmo recursivo, quando encontramos um t tal que $r(z) \neq r(t)$, armazenamos t em $q(w)$.

Agora vamos usar os vetores r, r', q e p ($p(u)$ é o pai de u em T^s) para construir caminhos P de s para u e Q de s para v tal que P e Q são internamente vértice-disjuntos. A construção é dada passo a passo e os caminhos são obtidos em ordem inversa. Dados dois vértices u e v, se $u \neq v$ e $r(u) \neq r(v)$, então s é uma junção de u e v e os caminhos invertidos de u para $r(u)$ (o caminho P inicial) e de v para $r(v)$ (o caminho Q inicial) ambos sobre T^s são vértice-disjuntos. Atualizando u para $r(u)$ e v para $r(v)$, temos que os novos vértices u e v são representantes. Se $u = v$ e $r'(u) = 1$, então s é uma junção do par u, u e o caminho invertido de u para $r(u)$ sobre T^s (o caminho P inicial) e o arco invertido $u \to q(u)$ mais o caminho invertido de $q(u)$ para $r(q(u))$ sobre T^s (o caminho Q inicial) são internamente vértice-disjuntos. Atualizando u para $r(u)$ e v para $r(q(u))$ temos novamente u e v representantes. Em qualquer caso, depois desses primeiros passos temos os seguintes invariantes:

1. Os vértices u e v são representantes distintos.

2. Os caminhos P e Q são internamente vértice-disjuntos.

O algoritmo recebe dois vértices representantes u e v, e dois caminhos internamente vértice-disjuntos P e Q. Enquanto ambos os vértices u e v não são iguais a s, se $pós(u) < pós(v)$, então se $r(p(u)) \neq v$, então aumentamos o caminho P adicionando o caminho invertido de $p(u)$ para $r(p(u))$ sobre T^s e atualizamos $u = r(p(u))$. Se $r(p(u)) = v$, então aumentamos P adicionando o arco invertido $u \to q(u)$ mais o caminho invertido de $q(u)$ para $r(q(u))$ sobre T^s e atualizamos $u = r(q(u))$. Note que os invariantes são mantidos na próxima iteração. De maneira simétrica, aumentamos o caminho Q quando $pós(v) < pós(u)$. Quando $u = v = s$, P e Q são caminhos internamente vértice-disjuntos de s para u e de s para v. Na Figura 4.8 vemos o funcionamento

deste algoritmo em um exemplo. A Figura 4.8 (a) mostra o digrafo acíclico do nosso exemplo. Os vértices em negrito são os representantes. A inicialização dos caminhos P e Q estão na Figura 4.8 (b). Note que cada caminho termina em um vértice que é um representante. Nas Figuras 4.8 (c) e (d) o caminho Q é atualizado duas vezes pois temos $pós(v) < pós(u)$. Na Figura 4.8 (e) o caminho P é atualizado pois $pós(u) < pós(v)$. Na Figura 4.8 (e), primeiro Q e depois P são atualizados, ambos terminam em s e o algoritmo para.

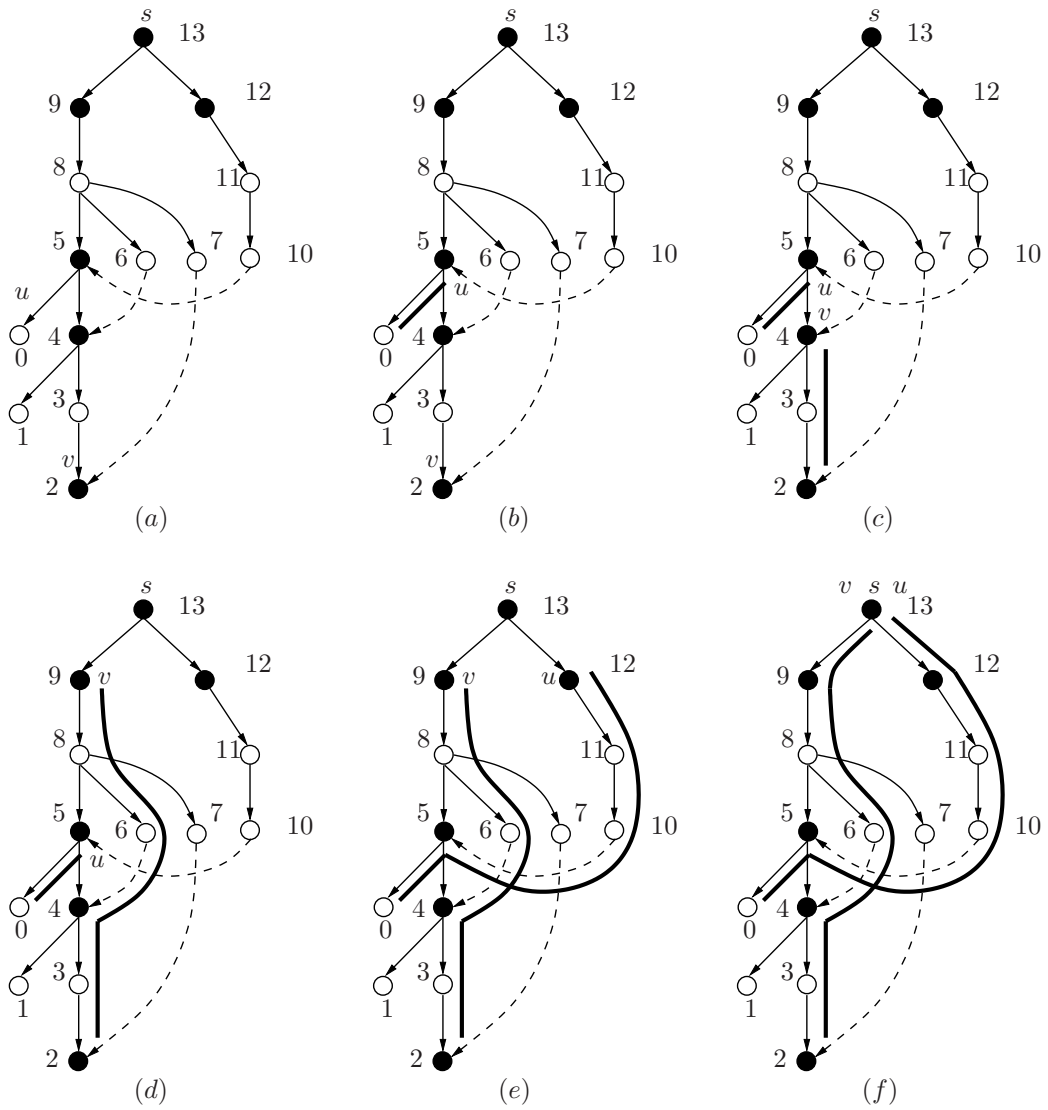


Figura 4.8: Algoritmo que encontra dois caminhos internamente vértice-disjuntos. Os vértices em negrito são representantes e o número em cada vértice é o valor pós-ordem do vértice em T^s.

Dados dois vértices u e v cujo vértice s é uma junção, o tempo gasto para obter um par de caminhos internamente vértice-disjuntos P de s para u e Q de s para v é linear no número de arcos de P e Q. Na próxima seção, vamos mostrar como uma partição de um digrafo acíclico em estrelas pode ser usada na solução de diversos problemas.

4.3 Usando uma Partição em Estrelas nos Problemas de Junções

O objetivo desta seção é justificar as contribuições descritas no Capítulo 3. Todos os algoritmos descritos são simples e usam uma partição em estrelas de um digrafo acíclico. Para cada caso,

são dados um digrafo acíclico D e pares de vértices $\mathcal{P} = \{\{u_1, v_1\}, \ldots, \{u_k, v_k\}\}$. A letra K representa o número total de acps ou junções para todo par em $\mathcal{P}$.

O Problema *representante-acp-k-pares*. Definido na Seção 3.2.4, este problema pode ser resolvido em tempo $O(n(m+k))$ e espaço $O(m+k)$ da seguinte forma:

1. determine uma ordenação topológica de D;
2. inicialize uma lista $RepACP(u_i, v_i)$ para todo par $\{u_i, v_i\}$ de $\mathcal{P}$;
3. para cada vértice s em D crescente topologicamente;
   (a) determine o conjunto de descendentes S de s;
   (b) particione $D[S]$ em estrelas (r^s); e
   (c) para cada par $\{u_i, v_i\}$ em $\mathcal{P}$, se $r^s(u_i) \neq r^s(v_i)$, $Substitua(RepACP(u_i, v_i), s)$.

O Problema *todos-acps-k-pares*. Definido na Seção 3.3.4, este problema pode ser resolvido em tempo $O(n(m+kn))$ porém não inferior ao tempo necessário para calcular o fecho transitivo de D. O algoritmo é descrito seguinte forma:

1. determine uma ordenação topológica de D;
2. determine o fecho transitivo D_{ft} de D;
3. inicialize uma lista $TodosACPs(u_i, v_i)$ para todo par $\{u_i, v_i\}$ de $\mathcal{P}$;
4. para cada vértice s de D decrescente topologicamente;
   (a) determine o conjunto de descendentes S de s;
   (b) particione $D[S]$ em estrelas (r^s); e
   (c) para cada par $\{u_i, v_i\}$ em $\mathcal{P}$, se $r^s(u_i) \neq r^s(v_i)$ e não existe arco $s \to s'$ em D_{ft} para todo s' em $TodosACPs(u_i, v_i)$, então $Insira(TodosACPs(u_i, v_i), s)$.

O Problema *representante-junção-k-pares*. Definido na Seção 3.4.2, este problema pode ser resolvido em tempo $O(n(m+k))$ e espaço $O(m+k)$ da seguinte forma:

1. inicialize uma lista $RepJ(u_i, v_i)$ para todo par $\{u_i, v_i\}$ de $\mathcal{P}$;
2. para cada vértice s em D;
   (a) determine o conjunto de descendentes S de s;
   (b) particione $D[S]$ em estrelas (r^s); e
   (c) para cada par $\{u_i, v_i\}$ em $\mathcal{P}$, se $r^s(u_i) \neq r^s(v_i)$, então $Substitua(RepJ(u_i, v_i), s)$.

O Problema *todas-junções-k-pares*. Definido na Seção 3.5.2, este problema pode ser resolvido em tempo $O(n(m+k))$ e espaço $O(m+K)$ da seguinte forma:

1. inicialize uma lista $TodasJs(u_i, v_i)$ para todo par $\{u_i, v_i\}$ de $\mathcal{P}$;
2. para cada vértice s de D;
   (a) determine o conjunto de descendentes S de s;
   (b) particione $D[S]$ em estrelas (r^s); e
   (c) para cada par $\{u_i, v_i\}$ em $\mathcal{P}$, se $r^s(u_i) \neq r^s(v_i)$, então $Insira(TodasJs(u_i, v_i), s)$.

O Problema *todas-junções-todos-pares*. Definido na Seção 3.5.2, este problema pode ser resolvido em tempo $O(n^3)$ e espaço $O(m + K)$ aplicando o algoritmo para o problema *todas-junções-k-pares* porém considerando todos os pares de vértices de D.

Note que o problema *todos-acps-todos-pares* pode ser resolvido em tempo $O(n^{3,2567})$ [EMN07] e que mostramos como resolver o problema *todas-junções-todos-pares* em tempo $O(n^3)$. Algumas questões ainda estão em aberto. Descrevemos duas que consideramos importantes. É possível resolver o problema *todos-acps-todos-pares* em tempo $O(n^3)$ e o problema *todas-junções-todos-pares* em tempo $O(m + K)$?

4.4 Um Algoritmo Linear para o Problema s-arco-junção-k-pares

Até agora a palavra junção foi usada somente no contexto de caminhos vértice-disjuntos. Nesta seção vamos usar a palavra junção também no contexto de caminhos arco-disjuntos. Dizemos que s é um *arco-junção* de u e v se existem caminhos arco-disjuntos de s a u e de s a v. Definimos o problema *s-arco-junção-k-pares* da mesma forma que o problema *s-junção-k-pares*.

> O problema *s-arco-junção-k-pares*: Dados um digrafo acíclico D, um vértice s de D e pares de vértices $\mathcal{P} = \{\{u_1, v_1\}, \ldots, \{u_k, v_k\}\}$, determinar quais pares de vértices em $\mathcal{P}$ têm s como arco-junção.

É natural considerar, para este caso, o digrafo linha D_l de D (mais uma cópia do vértice s (s') e os arcos $s' \to u$ onde u é um arco em D com ponta inicial igual a s) e usar o algoritmo recursivo descrito na Seção 4.2. É claro que s' é uma junção de u' e v' em D_l se e somente se as pontas de u' e v' em D têm s como junção. Com isso, temos que fazer algumas alterações na fase de consultas, já que $r(u') \neq r(v')$ em D_l significa que em D, partindo de s, atingimos as pontas dos arcos u' e v' com caminhos arco-disjuntos. A construção de D_l e o pré-processamento em D_l é linear mas não é óbvio que uma consulta *s-junção*(u, v) em D é feita em tempo constante pois existem $g_D^-(u)$ e $g_D^-(v)$ arcos em D com pontas finais em u e em v.

Ao invés de usar o digrafo linha vamos alterar o algoritmo recursivo SJKP, descrito na Seção 4.2, para responder a uma consulta *s-arco-junção*(u, v). Da mesma forma que o algoritmo SJKP, o novo algoritmo (SAJKP) resolve o problema *s-arco-junção-k-pares* em duas fases. A **Fase 1** constrói uma estrutura de dados (os vetores r e r' para tratar inclusive consultas onde $u = v$) em tempo linear. A **Fase 2** resolve consultas em tempo constante. O algoritmo recebe um vértice z e devolve, para todo par de vértices u e v internos à árvore T^z, os vetores r e r' de tal forma que as seguintes propriedades são satisfeitas. Se $u \neq v$, então $r(u) \neq r(v)$ se e somente se existem caminhos arco-disjuntos de s para u e de s para v. Se $u = v$, então $r'(u) = 1$ se e somente se existem caminhos arco-disjuntos de s para u. Assim, para verificar se s é um arco-junção dos vértices u e v, basta fazer no máximo duas comparações. Antes de descrever a **Fase 1** do SAJKP observamos o seguinte fato.

Observação 4.1. *Considere o algoritmo SJKP (vértice-disjunto) e tome um vértice w com $r'(w) = 1$ e com pelo menos um filho w_i em T^s. No contexto vértice-disjunto sabemos que s é uma junção do par w e w, pois $r'(w) = 1$ (Lema 4.6 a). Portanto, no contexto arco-disjunto, além de w ser um representante, cada filho de w também é um representante, isto é, $r(w) = w$ e $r(w_i) = w_i$ (veja a Figura 4.9).*

Agora vamos descrever o algoritmo. O vetor r é inicializado como em SJKP. No início, o vetor $r'(u) = 0$ para todo $u \neq s$ e $r'(s) = 1$. Continuamos considerando os vetores *pós* e *minpós*. A primeira chamada ao algoritmo recursivo SAJKP é feita com o parâmetro $z = s$ e em cada chamada recursiva analisamos novamente dois casos:

Caso 1: T^z tem um único vértice. Nada é feito.

Caso 2: T^z tem pelo menos dois vértices. Os seguintes passos devem ser feitos para cada filho z_i de z em T^s em ordem decrescente do valor pós-ordem de T^s. Transformamos z_i em

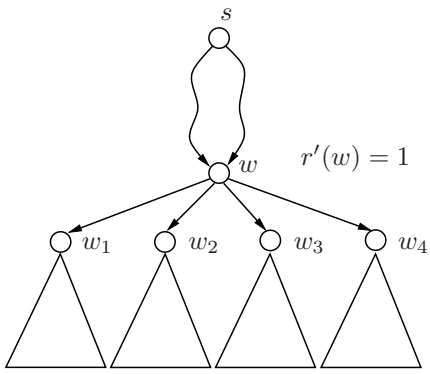


Figura 4.9: Os filhos do vértice w serão representantes no contexto arco-junção pois qualquer par de vértices u e v com u em T^{w_i} e v em T^{w_j} tem s como arco-junção.

um representante, isto é, $r(z_i) = z_i$. Se existir um arco $t \to z_i$ que não está em T^s, então modificamos $r'(z_i) = 1$, resolvemos o problema recursivamente para a subárvore T^{z_i} e o algoritmo continua considerando o próximo filho de z em T^s. Se não existir tal arco, então continuamos o algoritmo com o mesmo filho z_i de z. Denote por w o vértice com $pós(w) = pós(z_i) - 1$. No início, w é um filho de z_i. Considere todos os arcos $t \to w$ que não estão em T^s e faça os seguintes passos enquanto w é um vértice de T^{z_i}, isto é, $pós(w) \geq minpós(z_i)$. Se para algum t temos $r'(t) = 1$ ou $r(z_i) \neq r(t)$ (isto é, s é um arco-junção do par t e t ou dos vértices z_i e t, respectivamente), então modificamos $r(w) = w$ e $r'(w) = 1$, resolvemos o problema recursivamente para a subárvore T^w e, depois da chamada recursiva, atualizamos w para $minpós(w) - 1$. Se para todo t temos $r'(t) = 0$ e $r(z_i) = r(t)$, então modificamos $r(w) = z_i$ e atualizamos w para o vértice com valor pós-ordem igual a $pós(w) - 1$.

4.5 Multiplicação de Matrizes Booleanas Retangulares

Dadas duas matrizes retangulares A $(m \times r)$ e B $(r \times n)$, o *método linha-coluna* [SS98] obtém a matriz produto $C = AB$ $(m \times n)$ em tempo $O(W + mr + rn)$ onde W é o número total de testemunhas para as entradas em C iguais a 1. Vamos descrever o funcionamento deste método como feito em [Lin09]. São considerados conjuntos S_A^k e S_B^k para $k = 1, \ldots, r$. O índice-linha i da matriz A está em S_A^k se e somente se $A(i,k) = 1$. De forma simétrica, o índice-coluna j da matriz B está em S_B^k se e somente se $B(k,j) = 1$. Desta forma, para $k = 1, \ldots, r$, e para cada índice i em S_A^k e j em S_B^k, temos $C(i,j) = 1$ com testemunha k pois $A(i,k) = 1$ e $B(k,j) = 1$. Nesta seção descrevemos como implementar uma versão do método linha-coluna usando o algoritmo SJKP.

Dadas matrizes A $(m \times r)$ e B $(r \times n)$, primeiro construímos um digrafo acíclico D a partir de A e B, depois usamos o algoritmo SJKP, e por fim obtemos a matriz $C = AB$. A construção do digrafo acíclico é feita da seguinte forma. São criados um vértice denotado por s; um vértice denotado por c; m vértices denotados por Al_i para $i = 1, \ldots, m$; r vértices denotados por Ac_i para $i = 1, \ldots, r$; rn vértices denotados por $b_{i,j}$ para $i = 1, \ldots, r$ e $j = 1, \ldots, n$. Isso totaliza $2 + m + r + rn$ vértices. São criados um arco de s para c; um arco de s para $b_{i,j}$, para $i = 1, \ldots, r$ e $j = 1, \ldots, n$ onde a entrada $B(i,j) = 1$ ($O(rn)$ arcos); um arco de c para Al_i para $i = 1, \ldots, m$ onde a linha i de A não é nula ($O(m)$ arcos); um arco de Al_i para Ac_k para $i = 1, \ldots, m$ e $k = 1, \ldots, r$ onde a entrada $A(i,k) = 1$ ($O(mr)$ arcos); um arco de Ac_k para $b_{k,j}$ para $k = 1, \ldots, r$ e $j = 1, \ldots, n$ onde a entrada $B(k,j) = 1$ ($O(rn)$ arcos). Assim, o número total de arcos é $O(mr + rn)$. Veja um exemplo na Figura 4.10.

Depois da construção de D, usamos o algoritmo SJKP com os parâmetros $D[S]$ e s onde S é

$$A = \begin{array}{|c|c|c|c|} \hline 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 1 \\ \hline \end{array} \qquad B = \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline 0 & 0 & 0 \\ \hline 0 & 1 & 1 \\ \hline 0 & 0 & 1 \\ \hline \end{array}$$

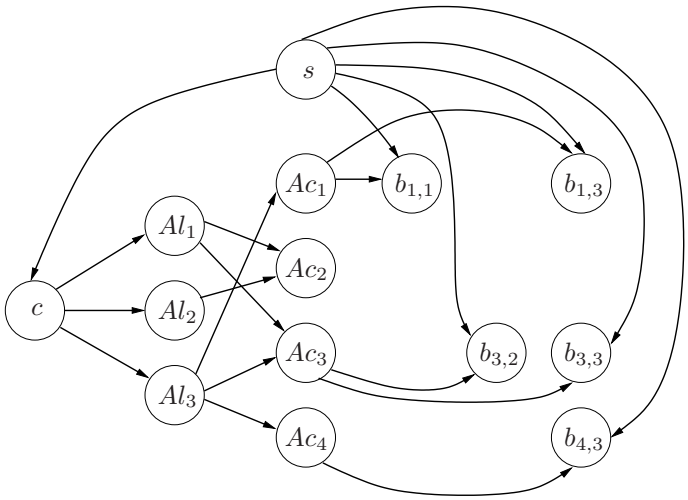


Figura 4.10: A construção de um digrafo acíclico D a partir das matrizes A e B dadas. Os vértices com grau de entrada iguais a zero foram omitidos.

o conjunto dos descendentes de s em D. Agora, suponha que s é uma junção do par $\{b_{k,j}, b_{k,j}\}$. Pela construção de D, os pais de $b_{k,j}$ são s e Ac_k, e a entrada $B(k,j) = 1$. Tome um vértice Al_i pai de Ac_k. Pela construção do digrafo, a entrada $A(i,k) = 1$. Com isso, a entrada $C(i,j) = 1$. Portanto, s é uma junção do par $\{b_{k,j}, b_{k,j}\}$ se, e somente se, $C(i,j) = 1$. Este algoritmo é uma forma de implementar o método linha-coluna pois um arco $Al_i \to Ac_k$ existe se e somente se i está em S_A^k, um arco $Ac_k \to b_{k,j}$ existe se e somente se j está em S_B^k, e quando s é uma junção do par $\{b_{k,j}, b_{k,j}\}$, os passos seguintes simulam exatamente a combinação dos índices pertencentes aos conjuntos S_A^k e S_B^k. Como a construção do digrafo acíclico consome tempo $O(mr + rn)$, o consumo total do algoritmo é igual ao do método linha-coluna, ou seja, $O(W + mr + rn)$ onde W é o número total de testemunhas para as entradas em C iguais a 1.

4.6 Uma Relação min-max *em Grafos Fluxos*

No prefácio do seu livro, Schrijver [Sch03] diz que algoritmos eficientes para um problema podem derivar relações min-max relacionadas. Foi exatamente isso o que aconteceu no nosso caso. Depois de obter um algoritmo eficiente para o problema *s-junção-k-pares* derivamos a relação min-max do Corolário 4.2. Schrijver ainda diz que o contrário também pode ocorrer: relações min-max podem dar origem a algoritmos eficientes. Assim, a importância de relações min-max é óbvia. Em seguida exemplificamos estes fatos com alguns subconjuntos especiais de vértices, arcos ou arestas de um grafo e suas relações min-max.

Um *grafo bipartido* é um grafo cujo conjunto de vértices pode ser particionado em dois subconjuntos de tal forma que toda aresta tem exatamente uma ponta em cada subconjunto. Dado um grafo G (não necessariamente bipartido), um *emparelhamento* em G é um conjunto

de arestas de G sem vértices em comum. Uma *cobertura por vértices* em G é um conjunto de vértices de G onde toda aresta é incidente a pelo menos um vértice da cobertura. O teorema a seguir, devido a Kőnig [Kőn31], relaciona em grafos bipartidos o tamanho máximo de um emparelhamento com o tamanho mínimo de uma cobertura por vértices.

Teorema 4.2 (Kőnig [Kőn31]). *Dado um grafo bipartido G, o tamanho máximo de um emparelhamento em G é igual ao tamanho mínimo de uma cobertura por vértices em G.* □

Dado um grafo G, um conjunto *independente* em G é um conjunto de vértices de G tal que não existe aresta em G que liga qualquer par de vértices neste conjunto. Uma *cobertura por arestas* em G é um conjunto de arestas de G onde todo vértice é adjacente a pelo menos uma aresta da cobertura. O próximo teorema, devido a Kőnig e Rado [Rad33], relaciona em grafos bipartidos o tamanho máximo de um conjunto independente com o tamanho mínimo de uma cobertura por arestas.

Teorema 4.3 (Kőnig e Rado [Rad33]). *Dado um grafo bipartido G, o tamanho máximo de um conjunto independente em G é igual ao tamanho mínimo de uma cobertura por arestas em G.* □

Dado um digrafo D, denote por $V(D)$ o conjunto de vértices de D e considere um subconjunto próprio e não vazio X de $V(D)$. Se o conjunto de arcos $\mathcal{E}$ é *dirigido*, isto é, todo arco em $\mathcal{E}$ tem ponta inicial em X e ponta final em $V(D) \setminus X$, então dizemos que $\mathcal{E}$ é um *corte dirigido* em D. Um conjunto de arcos de D que cobre todos os cortes dirigidos de D é uma *dijunção*.

Um teorema famoso em digrafos relaciona o número máximo de cortes dirigidos arco-disjuntos com o tamanho mínimo de uma dijunção.

Teorema 4.4 (Lucchesi e Younger [LY78]). *Dado um digrafo D conexo, o número máximo de cortes dirigidos arco-disjuntos em D é igual ao tamanho mínimo de uma dijunção em D.* □

A conjectura de Woodall define uma relação min-max que inverte os objetos otimizados do Teorema 4.4.

Conjectura 4.1 (Woodall [Woo78]). *Dado um digrafo D, o número máximo de dijunções disjuntas em D é igual ao tamanho mínimo de um corte dirigido em D.*

A conjectura de Woodall é verdadeira para algumas classes de digrafos como, por exemplo, para os digrafos *fonte-sorvedouro-conexo*, isto é, digrafos para os quais existe um caminho dirigido de cada vértice fonte para cada vértice sorvedouro [FY87] e [Sch82].

Para terminar essa parte onde listamos algumas relações min-max em grafos, os dois próximos teoremas descrevem relações min-max em grafos fluxos redutíveis.

Teorema 4.5 (Frank e Gyárfás [FG76]). *Dado um grafo fluxo redutível D, o número máximo de ciclos vértice-disjuntos em D é igual ao tamanho mínimo de um conjunto de vértices que intersectam todos os ciclos em D.* □

Frank e Gyárfás conjecturaram a versão para arcos da relação min-max do Teorema 4.5. Essa conjectura é verdadeira como mostra Ramachandran.

Teorema 4.6 (Ramachandran [Ram90]). *Dado um grafo fluxo redutível D, o número máximo de ciclos arco-disjuntos em D é igual ao tamanho mínimo de um conjunto de arcos que intersectam todos os ciclos em D.* □

Uma curiosidade é que antes de publicar o artigo [Ram90] em que o Teorema 4.6 é apresentado, Ramachandran publicou o artigo [Ram88] com um algoritmo polinomial para o problema onde é dado um grafo fluxo redutível com peso nos arcos, e deseja-se encontrar um conjunto de arcos de retorno com peso mínimo, o que nos faz pensar que Ramachandran primeiro teria

descoberto um algoritmo para o seu problema e depois uma relação min-max, como no nosso caso.

Agora começamos a descrever uma relação min-max sobre o tamanho de dois conjuntos definidos a seguir. Dados um grafo fluxo D com raiz em s, para todo vértice $u \neq s$ dizemos que o vértice s *domina trivialmente* o vértice u. Para cada vértice $u \neq s$ em D, definimos $\mathcal{A}_u^s$ como sendo um conjunto não vazio de dominadores não triviais de u. Se existem caminhos vértice-disjuntos de s para u (s é uma junção do par $\{u, u\}$), então o conjunto $\mathcal{A}_u^s$ é único e unitário ($\mathcal{A}_u^s = \{u\}$). Além disso, definimos $\mathcal{A}_s^s = \{s\}$. Com exceção do conjunto $\mathcal{A}_s^s$, nenhum outro conjunto possui o vértice s. Com isso, podemos definir o primeiro conjunto da relação min-max. O conjunto é definido formalmente como:

$$\mathcal{A}^s = \bigcup_{u \in D} \mathcal{A}_u^s,$$

onde $\mathcal{A}_s^s = \{s\}$, e $\mathcal{A}_u^s$ é um conjunto não vazio de dominadores não triviais para todo $u \neq s$ em D.

Em palavras, o conjunto $\mathcal{A}^s$ é formado pelo vértice s, e por pelo menos um vértice dominador não trivial para todo vértice $u \neq s$ em D. Note que $\mathcal{A}^s$ não é único. Por exemplo, supondo um grafo fluxo com raiz em s, como ilustra a Figura 4.11, um possível $\mathcal{A}^s$ é o conjunto $\{s, x, v, u\}$, pois podemos ter $\mathcal{A}_s^s = \{s\}$, $\mathcal{A}_x^s = \{x\}$, $\mathcal{A}_v^s = \{x, v\}$ e $\mathcal{A}_u^s = \{x, v, u\}$. No entanto, $\mathcal{A}^s$ pode ser o conjunto $\{s, x\}$, pois $\mathcal{A}_s^s = \{s\}$, $\mathcal{A}_x^s = \{x\}$, $\mathcal{A}_v^s = \{x\}$ e $\mathcal{A}_u^s = \{x\}$.

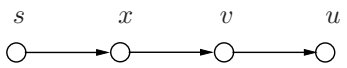


Figura 4.11: Podemos ter $\mathcal{A}^s = \{s, x\}$, ou $\mathcal{A}^s = \{s, x, v\}$, ou $\mathcal{A}^s = \{s, x, v, u\}$.

Denote por α o número de elementos de $\mathcal{A}^s$, e $\alpha^\star$ o número de elementos de um conjunto $\mathcal{A}^s$ de tamanho mínimo. É de nosso interesse encontrar um $\mathcal{A}^s$ com tamanho mínimo.

Agora vamos para o segundo conjunto da relação min-max. Dado um grafo fluxo D com raiz em s, denote por $V(D)$ o conjunto de vértices de D. Defina uma partição de $V(D)$, $\mathcal{B}^s = \{\{\mathcal{B}_1^s\}, \ldots, \{\mathcal{B}_\beta^s\}\}$, que atende à seguinte propriedade[1]:

Propriedade 4.1. *Dados dois vértices u e v, se $u \in \mathcal{B}_i^s$ e $v \in \mathcal{B}_j^s$, $i, j \in \{1, \ldots, \beta\}$, e $i \neq j$, então s é uma junção dos vértices u e v.*

Como $\mathcal{B}^s$ é uma partição dos vértices de D, temos que $\bigcup_{i=1}^{\beta} \mathcal{B}_i^s = V(D)$ e $\mathcal{B}_i^s \cap \mathcal{B}_j^s = \emptyset$ para quaisquer dois conjuntos diferentes $\mathcal{B}_i^s$ e $\mathcal{B}_j^s$. Vamos denotar por β o número de elementos de $\mathcal{B}^s$, e por $\beta^\star$ o número de elementos de uma partição $\mathcal{B}^s$ máxima. É de nosso interesse encontrar um $\mathcal{B}^s$ com tamanho máximo. O Lema 4.8 descreve uma importante propriedade de uma partição $\mathcal{B}^s$.

Lema 4.8. *Dados um grafo fluxo D com raiz em s, e uma partição $\mathcal{B}^s$ do conjunto de vértices de D que satisfaz a Propriedade 4.1. Para qualquer vértice $u \neq s$ em um específico $\mathcal{B}_i^s$ para algum $i \in \{1, \ldots, \beta\}$, todos os dominadores não triviais do vértice u estão em $\mathcal{B}_i^s$.*

Demonstração: Tome um vértice $u \neq s$ em um $\mathcal{B}_i^s$ para algum $i \in \{1, \ldots, \beta\}$. Dividimos a prova em dois casos.

1. Existem pelo menos dois caminhos vértice-disjuntos de s para u. Assim, o único dominador não trivial de u é o próprio u que está em $\mathcal{B}_i^s$.

[1]Note a diferença com a propriedade da partição $\mathcal{B}^s$ da **Fase 1** do algoritmo SJKP (Seção 4.2.1).

2. Não existem dois caminhos vértice-disjuntos de s para u. Tome um dominador w não trivial de u e diferente de u. Se não é possível tomar um w diferente de u (ou seja, u é filho de s), então w está em $\mathcal{B}_i^s$, e isso termina a demostração. Como w é um dominador de u, todos os caminhos de s para u passam por w. Assim, concluímos que s não pode ser uma junção dos vértices w e u. Por outro lado, se w está em $\mathcal{B}_j^s$, $j \in \{1, \ldots, \beta\}$, e $j \neq i$, então s seria uma junção dos vértices w e u, pois $\mathcal{B}^s$ satisfaz a Propriedade 4.1. Mas isso contradiz a conclusão anterior. Portanto, w só pode estar no mesmo conjunto que u, ou seja, w está em $\mathcal{B}_i^s$.

□

Os dois lemas e o corolário a seguir descrevem uma relação min-max em grafos fluxos redutíveis.

Lema 4.9. *Dados um grafo fluxo D com raiz em s, e conjuntos $\mathcal{A}^s$ e $\mathcal{B}^s$ como definidos anteriormente. Então vale a seguinte desigualdade: $|\mathcal{A}^s| \geq |\mathcal{B}^s|$.*

Demonstração: Tome $\mathcal{A}^s$ e $\mathcal{B}^s$ conjuntos como definidos anteriormente. Considere uma parte $\mathcal{B}_i^s$ de $\mathcal{B}$. Temos dois casos.

1. $|\mathcal{B}_i^s| \geq 2$. Tome u e v em $\mathcal{B}_i^s$, $u \neq v$. Considere mais dois casos.

   (a) O vértice s é uma junção dos vértices u e v. Neste caso, u e v não têm dominadores comuns e, como $\mathcal{A}_u^s$ e $\mathcal{A}_v^s$ são ambos diferentes de vazio, temos $|\mathcal{A}_u^s \cup \mathcal{A}_v^s| \geq 2$. Pelo Lema 4.8, os vértices de $\mathcal{A}_u^s$ e $\mathcal{A}_v^s$ devem estar em $\mathcal{B}_i^s$.

   (b) O vértice s não é uma junção dos vértices u e v. Então existe um dominador comum não trivial de u e v, e assim $|\mathcal{A}_u^s \cup \mathcal{A}_v^s| \geq 1$. Mais uma vez, pelo Lema 4.8, os vértices de $\mathcal{A}_u^s$ e $\mathcal{A}_v^s$ devem estar em $\mathcal{B}_i^s$.

2. $|\mathcal{B}_i^s| = 1$. Considere $\mathcal{B}_i^s = \{u\}$. Pelo Lema 4.8, $\mathcal{A}_u^s = \{u\} = \mathcal{B}_i^s$.

Com isso, para cada elemento da partição $\mathcal{B}^s$, mostramos que existe um vértice diferente em $\mathcal{A}^s$. Portanto, $\mathcal{A}^s \geq \mathcal{B}^s$. □

Para o próximo lema precisamos da seguinte definição. Considere um grafo fluxo D com raiz em s e dois vértices w e u em D. O vértice w é o *dominador mais alto não trivial* de u somente quando w é diferente de s, w domina u e qualquer ancestral próprio de w não domina u, exceto pela raiz s do grafo fluxo. Os próximos lema e corolário provam a relação min-max para grafos fluxos redutíveis.

Lema 4.10. *Dados um grafo fluxo redutível D com raiz em s e um conjunto máximo $\mathcal{B}^s$, é possível construir um conjunto $\mathcal{A}^s$ tal que $|\mathcal{A}^s| = \alpha = \beta^\star = |\mathcal{B}^s|$.*

Demonstração: Suponha que o conjunto máximo $\mathcal{B}^s$ foi obtido através do algoritmo SJKP, isto é, a partição de D em estrelas é a própria partição $\mathcal{B}^s$. Pelo algoritmo, temos $\beta^\star$ vértices representantes, denotados por $w_1, \ldots, w_{\beta^\star}$. Sem perda de generalidade, suponha que $w_1 = s$. Vamos mostrar que o conjunto $\{w_1, \ldots, w_{\beta^\star}\}$ é um conjunto válido para $\mathcal{A}^s$. Tome um vértice w_i em $\{w_2, \ldots, w_{\beta^\star}\}$ e tome um vértice $u \neq w_i$ em $\mathcal{B}_i^s$. Se não é possível tomar um $u \neq w_i$ em $\mathcal{B}_i^s$, então pelo Lema 4.8 o único dominador de u está em $\mathcal{B}_i^s$ e é o próprio u. Como $w_i \neq s$, temos que w_i é o dominador mais alto não trivial de u. Então estamos supondo que foram tomados um w_i e um $u \neq w_i$. Vamos mostrar que w_i é o dominador mais alto não trivial de u. Lembre que w_i é um ancestral próprio de u, e que s não é uma junção dos vértices w_i e u pois ambos estão em $\mathcal{B}_i^s$ (propriedade da partição $\mathcal{B}^s$ devolvida pelo algoritmo SJKP). Dividimos a prova em dois casos.

1. Não existem caminhos internamente vértice-disjuntos de s para w_i. Então, pelo algoritmo SJKP, w_i é um filho de s[2]. Se existe um caminho de s para u que não passa por w_i, então s é uma junção dos vértices w_i e u, uma contradição. Portanto, todos os caminhos de s para u passam por w_i, e assim w_i domina u. Para este caso, não pode existir um ancestral próprio de w_i diferente de s que domina u. Logo, o vértice w_i é o dominador mais alto não trivial de u.

2. Existem caminhos internamente vértice-disjuntos de s para w_i. Tome dois caminhos internamente vértice-disjuntos P e Q de s para w_i. Tais caminhos garantem não existir um ancestral próprio de w_i diferente de s que domina u. Com isso, nos falta mostrar que w_i domina u. Tome um caminho R de s a u que não passa por w_i. Se os caminhos R e P ou R e Q têm somente o s como vértice comum, então s é uma junção dos vértices w_i e u, uma contradição. Portanto, os caminho R e P têm vértices comuns diferentes de s, e R e Q também têm vértices comuns diferentes de s. Denote os vértices comuns a R e P por s, $p_1, \ldots, p_k$, e os vértices comuns a R e Q por s, $q_1, \ldots, q_l$. Como P e Q são internamente vértice-disjuntos, temos $p_i \neq q_j$ para todo $i = 1, \ldots, k$ e $j = 1, \ldots, l$ (Figura 4.12).

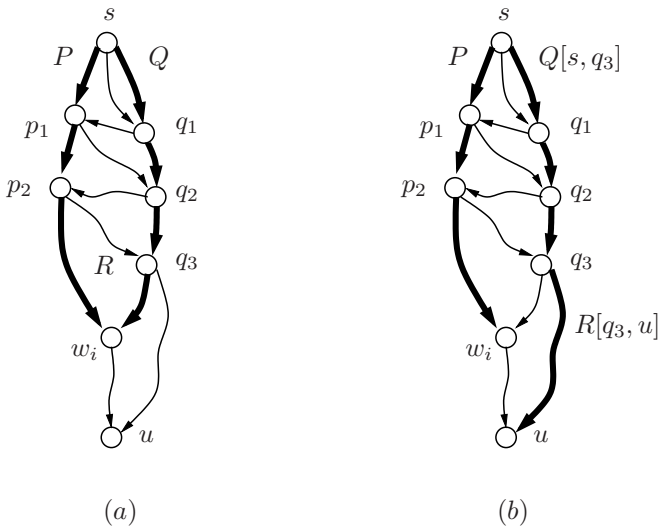


Figura 4.12: O vértice q_3 é o último vértice em R comum aos caminhos P e Q. Em destaque aparecem caminhos vértice-disjuntos.

Sem perda de generalidade, suponha que o vértice q_l é o último vértice em R comum a P ou Q (Figura 4.12 (a)). Assim, os caminhos P e $Q[s, q_l]R[q_l, u]$ são internamente vértice-disjuntos (Figura 4.12 (b)). Mas isso contradiz o fato de que s não é uma junção dos vértices w_i e u. Portanto concluímos que todos os caminhos de s para u passam por w_i, e assim w_i domina u.

Agora, fazemos $\mathcal{A}_s^s = \{w_1 = s\}$, e para cada $i = 2, \ldots, \beta^\star$ e para todo vértices u em $\mathcal{B}_i^s$ fazemos $\mathcal{A}_u^s = \{w_i\}$. Com isso, $\mathcal{A}^s = \bigcup_{u \in D} \mathcal{A}_u^s = \{w_1, \ldots, w_{\beta^\star}\}$. Isso conclui a demonstração. □

Pelos Lemas 4.9 e 4.10, segue o Corolário 4.2.

Corolário 4.2. *Dado um grafo fluxo redutível D com raiz em s, o tamanho de um conjunto mínimo $\mathcal{A}^s$ de D é igual ao tamanho de um conjunto máximo $\mathcal{B}^s$ de D, isto é, $\alpha^\star = \beta^\star$.* □

Portanto, o Corolário 4.2 nos diz que em um grafo fluxo redutível D com raiz em s, o número máximo de partes que o conjunto de vértices de D pode ser particionado, tal que dois vértices

[2]Para perceber isso, é importante lembrar que quando um w_i se torna um representante pelo **Caso 2** do algoritmo SJKP, existem caminhos vértice-disjuntos de s para w_i (veja o Lema 4.6 **a** quando $w' = w$).

em partes diferentes têm s como junção é igual ao número mínimo de vértices que dominam não trivialmente todos os vértices do grafo.

Por outro lado, fica fácil obter um conjunto $\mathcal{A}^s$ de tamanho mínimo para um grafo fluxo geral D se nos é dado a árvore de dominadores T de D. Basta fazer $\mathcal{A}^s = \{s, s_1, \ldots, s_k\}$, onde $s_1, \ldots, s_k$ são filhos de s em T. Assim, podemos obter $\mathcal{B}^s$ a partir de $\mathcal{A}^s$, fazendo $\mathcal{B}^s_s = \{s\}$ e $\mathcal{B}^s_u = \{s_i\}$, para todo $i = 1, \ldots, k$, e para todo vértice u descendente de s_i em T. Como a árvore de dominadores de um grafo fluxo D com raiz em s é única, teremos únicos conjuntos $\mathcal{A}^s$ de tamanho mínimo e $\mathcal{B}^s$ de tamanho máximo. Veja na Figura 4.13 como podemos obter $\mathcal{B}^s$, se tivermos $\mathcal{A}^s$.

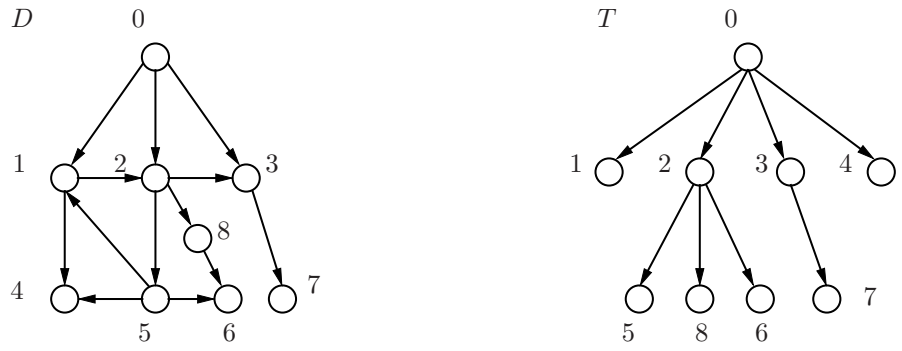


Figura 4.13: Um grafo fluxo D com raiz em 0 e sua árvore de dominadores T. A raiz de T mais os filhos da raiz são os elementos do conjunto mínimo de dominadores não triviais para todo vértices de D, isto é, $\mathcal{A}^0 = \{0, 1, 2, 3, 4\}$. O conjunto $\mathcal{B}^0_0 = \{0\}$ e $\mathcal{B}^0_u = \{i\}$ para $i = 1, 2, 3, 4$ e para todo vértice u descendente de i em T.

Seguem desses fatos os resultados do Lema 4.10 e do Corolário 4.2 para grafos fluxos em geral.

Lema 4.11. *Dados um grafo fluxo D com raiz em s e o conjunto mínimo $\mathcal{A}^s$, é possível construir o conjunto $\mathcal{B}^s$ tal que $|\mathcal{A}^s| = \alpha^\star = \beta = |\mathcal{B}^s|$.* □

Corolário 4.3. *Dado um grafo fluxo D com raiz em s, o tamanho do conjunto mínimo $\mathcal{A}^s$ de D é igual ao tamanho do conjunto máximo $\mathcal{B}^s$ de D, isto é, $\alpha^\star = \beta^\star$.* □

4.6.1 Construindo uma Árvore de Dominadores para Grafos Fluxos Redutíveis

Os resultados do Lema 4.11 e do Corolário 4.3 generalizam os resultados do Lema 4.10 e do Corolário 4.2. No entanto, a demonstração do Lema 4.10 sugere um algoritmo para construir uma árvore de dominadores de um grafo fluxo redutível em tempo $O(hm)$ onde h é a altura da árvore de dominadores do grafo fluxo redutível dado. O procedimento é recursivo e recebe um digrafo acíclico[3] D e um vértice s. Se D tem mais que 1 vértice, então obtenha uma partição $\mathcal{B}^s$ através do algoritmo SJKP com parâmetros D e s onde o conjunto $\{w_1, w_2, \ldots, w_{\beta^\star}\}$ é o conjunto dos representantes de cada subconjunto de $\mathcal{B}^s$, e que $w_1 = s$. O procedimento continua, para $i = 2, \ldots, \beta^\star$: chame o procedimento recursivamente com parâmetros $D[\mathcal{B}^s_i]$ e w_i; e atribua $p_T(w_i) \leftarrow s$ (o pai de w_i na árvore de dominadores T é s).

O tempo gasto por este procedimento é $O(hm)$ onde h é a altura da árvore de dominadores pois consome-se tempo $O(m)$ para construir cada nível da árvore.

[3]Estamos considerando que o grafo fluxo redutível dado foi pré-processado produzindo o seu digrafo acíclico.

5

Uma Aplicação na Antropologia

Neste capítulo definimos alguns problemas da área de Antropologia e discutimos as dificuldades em resolvê-los, até mesmo quando o digrafo associado à entrada é acíclico e com grau limitado nos vértices. A grosso modo, nestes problemas desejamos encontrar estruturas formadas por caminhos vértice-disjuntos que relacionam ancestrais comuns de indivíduos casados. Essas estruturas são chamadas de k-anéis e a sua definição do ponto de vista computacional encontra-se na Seção 2.4. Na Seção 5.1, definimos os problemas *todos-k-anéis* e *um-k-anel* e apresentamos suas complexidades. Na Seção 5.2, descrevemos a relação entre o problema *um-k-anel* e o problema da área de digrafos chamado de *subdigrafo-homeomorfo*. Na Seção 5.3, apresentamos um algoritmo polinomial baseado em uma redução de Eppstein para o caso quando k é fixo e o digrafo de entrada é acíclico, e alguns melhoramentos que experimentalmente aceleram a computação. Na Seção 5.4, descrevemos alguns experimentos realizados sobre sociedades de diferentes regiões do Brasil.

5.1 Uma Aplicação na Antropologia - Definição e Complexidade

O primeiro objetivo desta seção é falar sobre uma aplicação da Antropologia que motivou nosso trabalho. Fomos apresentados ao problema por dois pesquisadores da área de Antropologia: o professor Dr. Marcio Ferreira da Silva do Departamento de Antropologia da Faculdade de Filosofia, Letras e Ciências Humanas da Universidade de São Paulo e o professor Dr. João dal Poz Neto do Departamento de Ciências Sociais do Instituto de Ciências Humanas da Universidade Federal de Juiz de Fora. Temos trabalhado nos últimos anos em parceria no problema que descrevemos a seguir do ponto de vista computacional.

> O problema *todos-k-anéis*: Dados um grafo misto H e um conjunto ordenado $\mathcal{E}$ com k arestas de H, encontrar em H todos os k-anéis $\mathcal{R}_1 = (\mathcal{E}, \mathcal{P}_1), \mathcal{R}_2 = (\mathcal{E}, \mathcal{P}_2), \ldots, \mathcal{R}_t = (\mathcal{E}, \mathcal{P}_t)$.

Lembramos que a definição de um k-anel encontra-se na Seção 2.4. Note que o primeiro elemento do par que define um k-anel é dado do problema.

Com o intuito de fazer uma ligação entre as disciplinas Computação e Antropologia, algumas considerações devem ser feitas. Usamos um grafo misto para modelar uma rede de parentesco de uma sociedade juntamente com os casamentos desta sociedade. Assim, dada uma rede de parentesco de uma sociedade com seus relacionamentos de filiação (quem é filho de quem) e de afinidade (quem é casado com quem), podemos considerar os indivíduos da rede como vértices, os relacionamentos de filiação como arcos do vértice pai para o vértice filho, e os relacionamentos de afinidade como arestas entre os vértices casados obtendo assim um grafo misto[1].

O estudo de redes de parentesco e como as alianças matrimoniais se formam de acordo com regras e tabus de cada sociedade tem sua origem no trabalho de Lévi-Strauss [LS69], a quem recomendamos o leitor interessado. Em qualquer sociedade existem regras que possibilitam ou

[1]Como prefere o professor Marcio, um modelo reduzido para o estudo das alianças matrimoniais [dPFdS13].

impossibilitam alianças matrimoniais. Lévi-Strauss foi o primeiro antropólogo a verificar que a circulação (o "vai e vem") das pessoas em uma sociedade, e assim sua vida social, é um produto das regras que incentivam ou impossibilitam as alianças matrimoniais. Na Terceira Parte do livro de Louis Dumont [Dum75] vemos que em uma sociedade onde um homem, não podendo ter relações sexuais com parentes próximos como a sua irmã, ou a sua mãe (proibição do incesto), procura um outro homem para ser esposo de sua parente próxima, recebendo assim, como troca, a sua esposa. As regras de proibições produziriam uma *regra de troca*. Descobrir uma regra de troca de uma sociedade é um dos objetivos dos antropólogos seguidores de Lévi-Strauss. A teoria das alianças matrimoniais evoluiu com o passar dos anos e estruturas mais complexas, envolvendo casamentos entre indivíduos que estão ligados por outros casamentos, passaram a ganhar destaque na teoria das alianças [VJZ70], [HW96], [HHD+04], [dPdS09]. Por exemplo, vemos nas Figuras 5.1 e 5.2 algumas estruturas complexas para o caso da sociedade Enawenê-Nawês. Os vértices são divididos em homens (triângulos) e mulheres (círculos) e em até quatro grupos (ou clãs) representados pelas cores dos vértices. No cenário da Figura 5.1, dizemos que o casamento do indivíduo 1 *redobra*[2] o casamento do tio, o indivíduo 4.

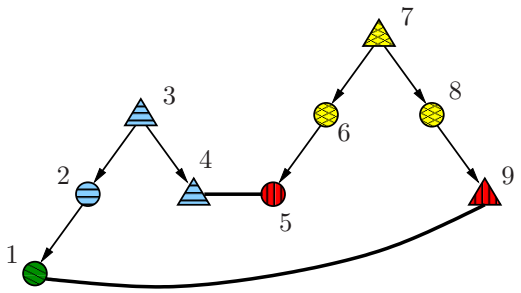


Figura 5.1: Existem quatro clãs na figura: $\mathcal{C}_1 = \{1\}$, $\mathcal{C}_2 = \{2, 3, 4\}$, $\mathcal{C}_3 = \{5, 9\}$ e $\mathcal{C}_4 = \{6, 7, 8\}$. O casamento da mulher 1 com um homem do clã $\mathcal{C}_3$ redobra o casamento do seu tio, o indivíduo 4, com uma mulher do clã $\mathcal{C}_3$ [dPFdS13].

No cenário da Figura 5.2, vemos três clãs diferentes e dizemos que o indivíduo 1 do clã $\mathcal{C}_1$ *reencadeia*[3] as alianças entre os clãs ao casar com o indivíduo 14 do clã $\mathcal{C}_3$.

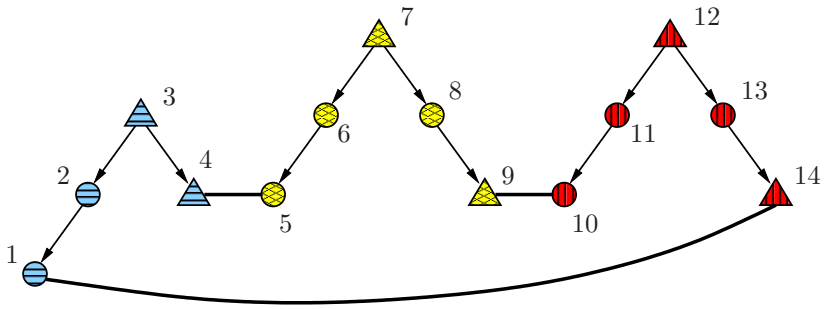


Figura 5.2: Os conjuntos $\mathcal{C}_1 = \{1, 2, 3, 4\}$, $\mathcal{C}_2 = \{5, 6, 7, 8, 9\}$ e $\mathcal{C}_3 = \{10, 11, 12, 13, 14\}$ são três clãs. O casamento da mulher 1 do clã $\mathcal{C}_1$ com o homem 14 do clã $\mathcal{C}_3$ reencadeia as alianças 4 e 5, e 9 e 10 [dPFdS13].

Observando essas estruturas, confirma-se o que se constata em entrevista com indivíduos desta sociedade: para os Enawenê-Nawês é um clã quem determina a regra de troca [dS13].

[2]Termo usado pelos antropólogos.
[3]Idem 2.

As estruturas complexas com k casamentos são aquelas que chamamos de k-anéis. Para um antropólogo descobrir uma regra de troca seria necessário observar todos os k-anéis de todos os conjuntos ordenados com k arestas e tirar suas conclusões[4].

Dito isso, voltemos ao nosso problema do ponto de vista computacional. O problema *todos-k-anéis* é um problema computacionalmente "intratável" pois pode existir um número exponencial de k-anéis de um conjunto ordenado com k arestas no grafo misto da entrada. De fato, até mesmo para listar todos os 1-anéis de um conjunto com uma aresta podemos levar tempo exponencial. Veja na Figura 5.3 que o número de 1-anéis depende exponencialmente do tamanho da entrada, até mesmo quando a entrada é um digrafo acíclico com limites nos graus de entrada e saída de cada vértice. Para ver o número exponencial de 1-anéis, usamos os vértices não marcados para obter qualquer subconjunto de vértices marcados pertencentes a caminhos de s a u_1 e de s a v_1. Como existem $\Omega(n)$ vértices marcados, temos um número exponencial de 1-anéis.

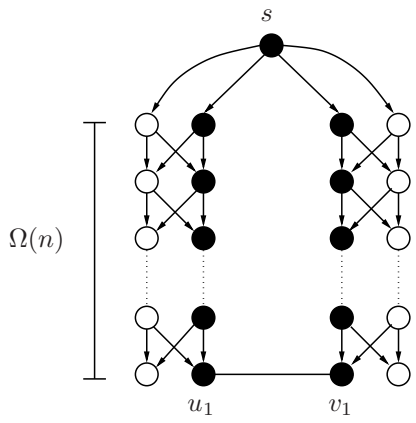


Figura 5.3: Tempo exponencial para listar todos os 1-anéis.

Como este problema não pode ser resolvido eficientemente, isto é, não existe algoritmo polinomial para listar todas essas estruturas, direcionamos nossa atenção para o problema de decidir se existe um k-anel em D.

> O problema *um-k-anel*: Dados um grafo misto H e um conjunto ordenado $\mathcal{E}$ com k arestas de H, decidir se existe um k-anel $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ em H.

Porém, decidir se existe um k-anel em H continua sendo uma tarefa difícil pois este problema é $\mathcal{NP}$-completo.

Lema 5.1. *O problema um-k-anel é $\mathcal{NP}$-completo, até mesmo quando o digrafo induzido pelos arcos do grafo misto é acíclico.*

Demonstração: Primeiro, verificamos que o problema está em $\mathcal{NP}$. Suponha que são dados um grafo misto H tal que o digrafo D induzido pelos arcos de H é acíclico, um subconjunto ordenado de arestas $\mathcal{E} = (w_1 = u_1 - v_k, w_2 = u_2 - v_1, \ldots, w_k = u_k - v_{k-1})$ de H, e um par $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ onde $\mathcal{P} = (P_1^{\mathcal{R}} Q_1, \ldots, P_k^{\mathcal{R}} Q_k)$ e $P_i^{\mathcal{R}}$ é um caminho de u_i para s_i e Q_i é um caminho de s_i para v_i para $i = 1, \ldots, k$. Queremos verificar em tempo polinomial se $\mathcal{R}$ é um k-anel em H. Primeiro, verificamos se s_i é o único vértice em comum nos caminhos $P_i^{\mathcal{R}}$ e Q_i para $i = 1, \ldots, k$. Isso mostra que s_i é uma junção dos vértices u_i e v_i para $i = 1, \ldots, k$. Depois, verificamos se os caminhos são dois a dois vértice-disjuntos (exceto os caminhos com um mesmo índice i que tem s_i como vértice único comum devem ser internamente vértice-disjuntos). Isso completa nossa

[4]Nossos colaboradores trabalham com $k = 1, 2, 3$, pois não encontram explicação antropológica para anéis maiores.

verificação de que $\mathcal{R}$ é um k-anel em H. Portanto, o problema *um-k-anel* está em $\mathcal{NP}$. Agora mostramos que ele é $\mathcal{NP}$-difícil. Vamos reduzir o problema *k-caminhos-vértice-disjuntos* (Even, Itai e Shamir [EIS75] mostraram que este problema é $\mathcal{NP}$-completo para digrafos acíclicos) para o problema *um-k-anel*.

> O problema *k-caminhos-vértice-disjuntos*: Dados um digrafo acíclico D e k pares de vértices $(s_1, t_1), \ldots, (s_k, t_k)$ de D, decidir se existem caminhos vértice-disjuntos P_i de s_i para t_i para $i = 1, \ldots, k$.

Dados um digrafo acíclico D e k pares ordenados de vértices $(s_1, t_1), \ldots, (s_k, t_k)$ de D, construímos um novo digrafo acíclico D' que é uma cópia de D mais k novos vértices $t'_1, \ldots, t'_k$ e k novos arcos $s_i \to t'_i$, para $i = 1, \ldots, k$. Criamos um conjunto ordenado de arestas $\mathcal{E} = (w_1, \ldots, w_k,)$, onde cada elemento é uma aresta $w_1 = t_1 - t'_k$ e $w_i = t_i - t'_{i-1}$ para $i = 2, \ldots, k$. Incluímos em D' o conjunto de arestas $\mathcal{E}$, tornando-o um grafo misto. Agora mostramos que existem caminhos vértice-disjuntos P_i de s_i para t_i para $i = 1, \ldots, k$ em D se e somente se existe um k-anel $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ em D'.

Considere que $P_1, \ldots, P_k$ são caminhos vértice-disjuntos de s_i para t_i em D. Portanto, se fizermos $\mathcal{P} = (P_1^{\mathcal{R}} Q_1, \ldots, P_k^{\mathcal{R}} Q_k)$ onde Q_i é o arco $s_i \to t'_i$ para todo $i = 1, \ldots, k$, então o par $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ é um k-anel em D',(veja Figura 5.4).

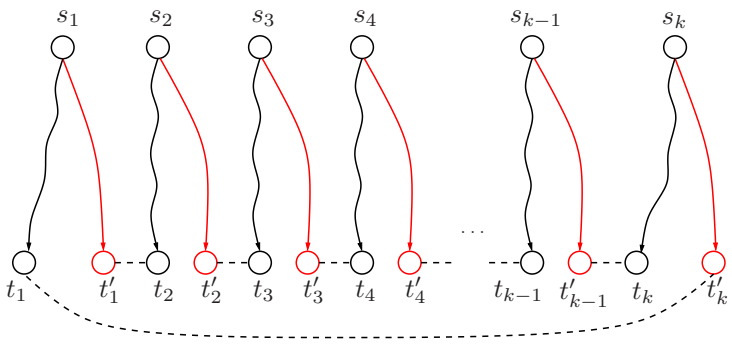


Figura 5.4: Um k-anel em D'.

Agora considere que $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ é um k-anel em D' onde $\mathcal{P} = (P_1^{\mathcal{R}} Q_1, \ldots, P_k^{\mathcal{R}} Q_k)$. Pela definição de um k-anel, os caminhos P_i e Q_i começam em uma junção dos vértices t_i e t'_i para $i = 1, \ldots, k$. Pela construção de D', a única junção dos vértices t_i e t'_i é s_i. Portanto, os caminhos $P_1, \ldots, P_k$ em D' são os caminhos vértice-disjuntos desejados que conectam os terminais $(s_1, t_1), \ldots, (s_k, t_k)$ em D. □

Se é permitido um grafo misto de entrada qualquer (o digrafo induzido pelos arcos pode ter ciclos), então o problema *um-2-anel* é $\mathcal{NP}$-completo. A demonstração desse fato é praticamente a mesma que a do Lema 5.1. Basta substituir k por 2 e o problema *k-caminhos-vértice-disjuntos* para digrafos acíclicos pelo problema *2-caminhos-vértice-disjuntos* para digrafos em geral que é $\mathcal{NP}$-completo devido a Fortune, Hopcroft e Wyllie [FHW80].

Lema 5.2. *Quando o digrafo induzido pelos arcos do grafo misto não é necessariamente acíclico, o problema um-2-anel é $\mathcal{NP}$-completo.* □

5.1.1 Limitando o Digrafo de Entrada

Note que o digrafo D induzido pelos arcos do grafo misto vindo da aplicação da área de Antropologia é acíclico e o grau de entrada de cada vértice de D é limitado (um indivíduo tem normalmente no máximo dois pais). Essa restrição no grau de entrada de cada vértice não deixa o problema mais fácil. Vamos mostrar que o problema *um-k-anel* continua $\mathcal{NP}$-completo, até

mesmo se cada vértice de D tem grau de entrada no máximo 1 e grau de saída no máximo 2 ou grau de entrada no máximo 2 e grau de saída no máximo 1.

Tome um grafo misto H tal que o digrafo D induzido pelos seus arcos é acíclico e um subconjunto ordenado de arestas $\mathcal{E} = (u_1 - v_k, u_2 - v_1, \ldots, u_k - v_{k-1})$ de H. Vamos construir em tempo polinomial um novo grafo misto H' tal que o digrafo D' induzido pelos arcos de H' contém as restrições citadas anteriormente, e que existe um k-anel $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ em H se e somente se existe um k-anel $\mathcal{R}' = (\mathcal{E}', \mathcal{P}')$ em H'. A construção pode ser acompanhada na Figura 5.5.

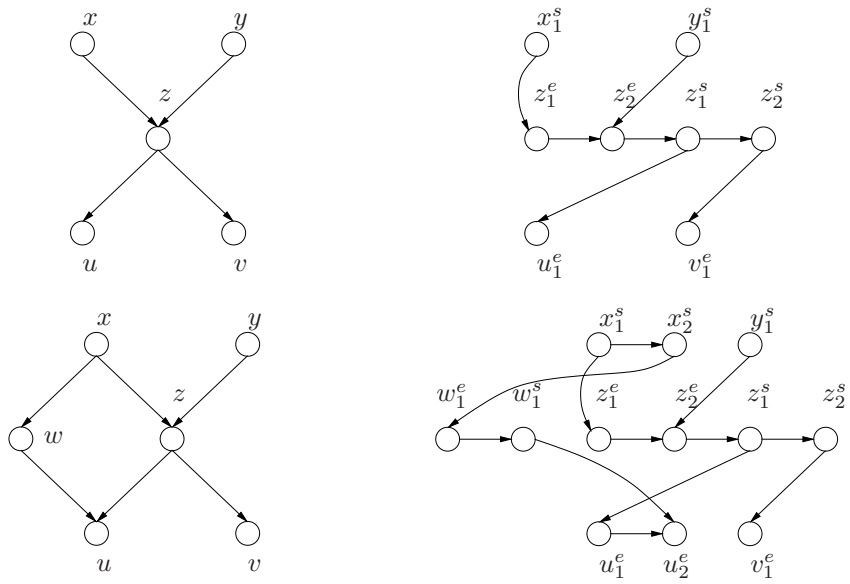


Figura 5.5: Estamos olhando somente para os digrafos acíclicos. À esquerda temos dois exemplos de digrafos acíclicos de entrada e à direita temos os dois digrafos correspondentes com as restrições nos graus de entrada e saída em cada vértice.

Para cada vértice x de D crie um dispositivo em D', denotado por $Disp(x)$, com $g_D^-(x) + g_D^+(x)$ vértices, $x_1^e, x_2^e, \ldots, x^e_{g_D^-(x)}$, $x_1^s, x_2^s, \ldots, x^s_{g_D^+(x)}$, e arcos $x_i^e \to x_{i+1}^e$ para $i = 1, \ldots, g_D^-(x) - 1$, $x^e_{g_D^-(x)} \to x_1^s$ se os graus de entrada e saída de x são ambos diferentes de 0, e $x_i^s \to x_{i+1}^s$ para $i = 1, \ldots, g_D^+(x) - 1$. Chame de $a_1^e(x), a_2^e(x), \ldots, a^e_{g_D^-(x)}(x)$ e $a_1^s(x), a_2^s(x), \ldots, a^s_{g_D^+(x)}(x)$ os arcos que entram e saem, respectivamente, de cada vértice x em D. Perceba que existem únicos i e j tais que o arco $a_i^s(x)$ corresponde ao arco $a_j^e(y)$ para todo arco $x \to y$ em D (vamos escrever $a_i^s(x) = a_j^e(y)$ neste caso). Crie um arco $x_i^s \to y_j^e$ em D' se e somente se existe um arco $x \to y$ em D e $a_i^s(x) = a_j^e(y)$. Este arco sai de $Disp(x)$ pelo vértice x_i^s e entra em $Disp(y)$ pelo vértice y_j^e. Por fim, crie um conjunto ordenado de arestas $\mathcal{E}' = \left({u_1}^e_{g_D^-(u_1)} - {v_k}^e_{g_D^-(v_k)}, {u_2}^e_{g_D^-(u_2)} - {v_1}^e_{g_D^-(v_1)}, \ldots, {u_k}^e_{g_D^-(u_k)} - {v_{k-1}}^e_{g_D^-(v_{k-1})} \right)$ que é o conjunto de arestas de H'. O próximo lema mostra uma correspondência entre caminhos em H e H'.

Lema 5.3. *Dados um grafo misto acíclico H, considere um outro grafo misto acíclico H' obtido através da construção descrita anteriormente. Existe um caminho $x_1, x_2, \ldots, x_k$ no grafo H se e somente se existe um caminho no grafo H' que passa exatamente pelos dispositivos $Disp(x_1), Disp(x_2), \ldots, Disp(x_k)$.*

Demonstração: Tome um caminho $x_1, \ldots, x_k$ em H. Então existem os arcos $x_1 \to x_2, x_2 \to x_3, \ldots, x_{k-1} \to x_k$. Para o arco $x_{l-1} \to x_l$ em H existem inteiros i_l e j_l, e um arco ${x_{l-1}}^s_{i_l} \to {x_l}^e_{j_l}$ em H' para $l = 2, \ldots, k$. O arco ${x_{l-1}}^s_{i_l} \to {x_l}^e_{j_l}$ em H' é um arco que sai de $Disp(x_{l-1})$ e entra em $Disp(x_l)$ para $l = 2, \ldots, k$. Um caminho em H' passando exatamente pelos dispositivos

$Disp(x_1), \ldots, Disp(x_k)$ é formado pela concatenação dos caminhos ${x_{l-1}}^s_{i_l}, {x_l}^e_{j_l}$ (arco ${x_{l-1}}^s_{i_l} \to {x_l}^e_{j_l}$) com o caminho ${x_l}^e_{j_l}, {x_l}^e_{j_l+1}, \ldots, {x_l}^e_{g^-_H(x_l)}$, ${x_l}^s_1, \ldots, {x_l}^s_{i_{l+1}}$ internos aos dispositivos $Disp(x_l)$ para $l = 2, \ldots, k$. Veja a Figura 5.6.

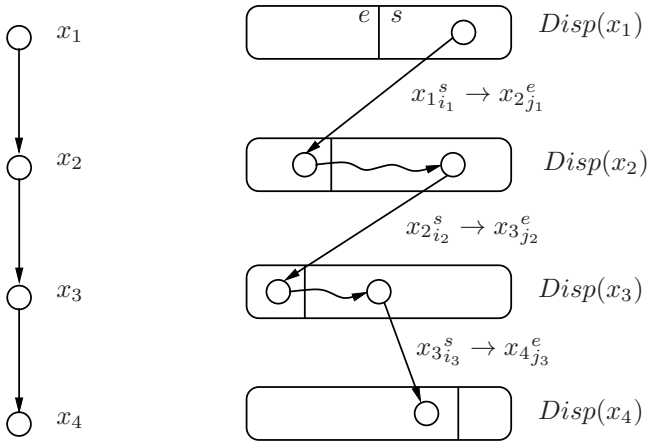


Figura 5.6: Um caminho em H e um caminho em H'.

Agora vamos mostrar o outro lado. Tome um caminho em H' passando exatamente pelos dispositivos $Disp(x_1), \ldots, Disp(x_k)$. Considere os arcos ${x_{l-1}}^s_{i_{l-1}} \to {x_l}^e_l$ de $Disp(x_{l-1})$ para $Disp(x_l)$ para $l = 2, \ldots, k$. Então existem arcos $x_{l-1} \to x_l$ em H para $l = 2, \ldots, k$. Portanto, existe um caminho $x_1, \ldots, x_k$ em H. □

Assim, para os grafos mistos H e H' como descritos anteriormente, existe um k-anel $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ em H constituído por caminhos internamente vértice-disjuntos de s_l para u_l e de s_l para v_l para $l = 1, \ldots, k$ se, e somente se, existem caminhos do dispositivos $Disp(s_l)$ para o dispositivo $Disp(u_l)$ e do dispositivo $Disp(s_l)$ para o dispositivo $Disp(v_l)$ passando por dispositivos quase disjuntos (com exceção do dispositivo $Disp(s_l)$) para $l = 1, \ldots, k$, isto é, se, e somente se, existe um k-anel $\mathcal{R}' = (\mathcal{E}', \mathcal{P}')$ em H'.

Por último, note que o número de arestas em $\mathcal{E}'$ é igual ao de $\mathcal{E}$. O número de vértices no digrafo H' é $2m$. O número de arcos em H' é $3m - n$ pois criamos $g^-_H(x) - 1$ arcos para todo vértice x com $g^-_H(x) \neq 0$, $g^+_H(x) - 1$ arcos para todo vértice x com $g^+_H(x) \neq 0$ e criamos adicionalmente 1 arco além dos $g^-_H(x) + g^+_H(x) - 2$ arcos para todo vértice v com $g^-_H(x) \neq 0$ e $g^+_H(x) \neq 0$. Se o digrafo original H é acíclico, então o novo digrafo H' continua acíclico e cada um de seus vértices tem grau de entrada no máximo 1 e grau de saída no máximo 2 ou grau de entrada no máximo 2 e grau de saída no máximo 1. Isso conclui a redução.

5.1.2 Complexidade Parametrizada

Se fixarmos um valor para k e se o digrafo D induzido pelos arcos do grafo misto dado é acíclico, então o problema *um-k-anel* pode ser decidido em tempo polinomial, a saber, $O(n^{3k+1})$ como veremos na Seção 5.3. Quando o problema *um-k-anel* é visto como um problema parametrizado, ele é $\mathcal{W}[1]$-difícil. Isso porque a redução polinomial feita no Lema 5.1 é também uma p-redução e o problema *k-caminhos-vértice-disjuntos* é $\mathcal{W}[1]$-difícil [Sli03].[5] Com isso, parece improvável que o problema *um-k-anel* esteja em $\mathcal{FPT}$, ou seja, parece improvável existir um algoritmo para o problema com tempo $O(f(k)n^c)$, onde f é qualquer função computável e c é uma constante.

Quando é permitido qualquer digrafo induzido pelos arcos do grafo misto, o problema *um-2-anel* não pode estar em $\mathcal{FPT}$ a menos que $\mathcal{P} = \mathcal{NP}$. Isso vem do fato de que o problema *um-2-anel* é $\mathcal{NP}$-completo. Assim, mostrar que ele está em $\mathcal{FPT}$, implica que $\mathcal{P} = \mathcal{NP}$.

[5] Slivkins [Sli03] mostrou que o problema *k-caminhos-arco-disjuntos* é $\mathcal{W}[1]$-difícil para digrafos acíclicos. Isso implica que o problema *k-caminhos-vértice-disjuntos* é $\mathcal{W}[1]$-difícil para digrafos acíclicos.

Na Seção 5.1, vimos que alguns problemas da área de Antropologia consistem essencialmente em encontrar (se existir) caminhos vértice-disjuntos em um digrafo acíclico. Vimos que provavelmente não existam algoritmos rápidos para resolvê-los. Outros trabalhos na literatura tratam de problemas relacionados. Na Seção 5.2, veremos a definição e alguns resultados para o problema *subdigrafo-homeomorfo*.

5.2 Relação entre os Problemas um-k-anel e subdigrafo-homeomorfo

Nesta seção vamos descrever o problema do subdigrafo homeomorfo que tem uma forte relação com o problema *um-k-anel*. Ele é declarado, na sua forma mais geral, como segue.

> O problema *subdigrafo-homeomorfo*: Dados os digrafos H e D, decidir se existem mapeamentos (um-para-um) m_V de vértices em H para vértices em D e m_A de arcos em H para caminhos em D (um arco $u \to v$ em H deve ser mapeado para um caminho de $m_V(u)$ para $m_V(v)$ em D). Os caminhos em D correspondentes a arcos em H devem ser vértice-disjuntos a menos de suas pontas.

Os digrafos H e D são chamados, respectivamente, de digrafo padrão e de digrafo de entrada. No caso mais geral, são permitidos laços e arcos paralelos em ambos os digrafos. Como observado em [LR80] e [FHW80], o problema *subdigrafo-homeomorfo* é $\mathcal{NP}$-completo [LR80] caso m_V seja ou não dado na entrada. Se m_V não é dado, então o problema de decidir se existe um ciclo hamiltoniano em um digrafo é redutível ao problema *subdigrafo-homeomorfo*. Basta escolher um H como um ciclo com a mesma quantidade de vértices do digrafo D. Se m_V é dado, então o problema *k-caminhos-vértice-disjuntos* é redutível ao problema *subdigrafo-homeomorfo*. Com isso, passaram a estudá-lo com H fixo. Sobre a hipótese de que $\mathcal{P} \neq \mathcal{NP}$, Fortune, Hopcroft e Wyllie [FHW80] mostraram que o problema *subdigrafo-homeomorfo*, com H fixo e m_V dado na entrada, é resolvido em tempo polinomial se e somente se o digrafo padrão está em uma determinada classe de digrafos. Essa classe é constituída pelos digrafos com um vértice especial, chamado de raiz, e todos os arcos ou entram na raiz ou saem da raiz (são permitidos laços na raiz). Outro resultado importante que aparece em [FHW80] é o seguinte: *se o digrafo D é acíclico, H é fixo e m_V é dado na entrada, então o problema subdigrafo-homeomorfo é resolvido em tempo polinomial para todo digrafo padrão H.*

No entanto, mesmo com o digrafo D sendo acíclico e m_V dado na entrada, se o digrafo padrão H não é fixo, então o problema *subdigrafo-homeomorfo* é $\mathcal{NP}$-completo [EIS75], [FHW80], [LR80] e [Vyg95].

Vamos redefinir um k-anel não-degenerado em termos dos mapeamentos m_V e m_A de um digrafo padrão H sobre um digrafo de entrada D. Dados um digrafo de entrada D e um conjunto ordenado de pares de vértices $\mathcal{E} = ((u_1, v_k), (u_2, v_1), \ldots, (u_k, v_{k-1}))$, um k-anel $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ em D é um subdigrafo de D obtido pelo mapeamento de H sobre D, onde H é o digrafo padrão formado por k componentes, cada uma com 3 vértices, s'_i, u'_i e v'_i, dois arcos $s'_i \to u'_i$ e $s'_i \to v'_i$, e o mapeamento m_V é dado parcialmente: $m_V(u'_i) = u_i$ e $m_V(v'_i) = v_i$ para $i = 1, \ldots, k$. (veja Figura 5.7).

Note que o vértice s'_i da componente i do digrafo padrão H não foi mapeado para $i = 1, \ldots, k$, ou seja, $m_V(s'_i)$ pode ser qualquer vértice no digrafo de entrada D. Em seguida definimos um novo problema que é uma combinação dos problemas *subdigrafo-homeomorfo* e *um-k-anel*.

> O problema *subdigrafo-homeomorfo-k-anel*: Dados um conjunto de pares de vértices $\mathcal{E} = ((u_1, v_k), (u_2, v_1), \ldots, (u_k, v_{k-1}))$, um digrafo de entrada D, um digrafo padrão H (com k componentes, cada uma com 3 vértices, s'_i, u'_i e v'_i, dois arcos $s'_i \to u'_i$ e $s'_i \to v'_i$) e um m_V dado parcialmente através de $\mathcal{E}$ ($m_V(u'_i) = u_i$ e $m_V(v'_i) = v_i$ para $i = 1, \ldots, k$), decidir se existem mapeamentos (um-para-um) m_V de vértices de H para vértices de D e m_A de arcos de H para caminhos em D (um arco $u \to v$ de H deve ser mapeado para um caminho de $m_V(u)$ para $m_V(v)$ em D). Os caminhos em D correspondentes a arcos em H devem ser vértice-disjuntos.

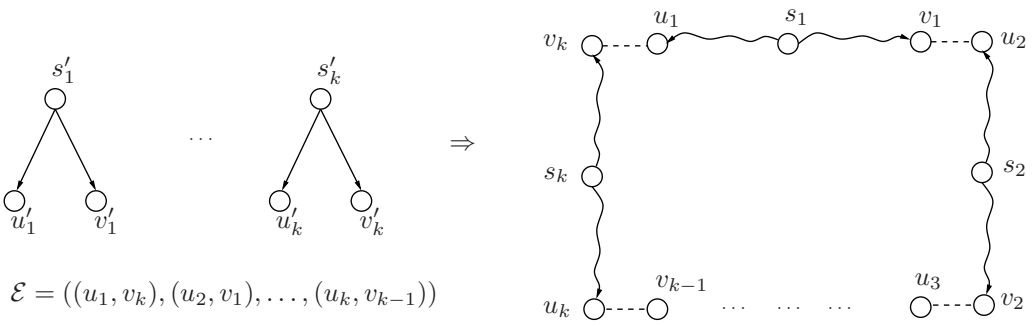


Figura 5.7: Um k-anel $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ em função dos mapeamentos m_V e m_A do digrafo padrão H sobre o digrafo de entrada D.

Ou seja, no problema *subdigrafo-homeomorfo-k-anel* desejamos saber se existe um k-anel $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ em D não-degenerado.

Por um instante, considere que m_V é dado totalmente. Assim, se são permitidos ciclos em D então, pelo Teorema 2 em [FHW80], o problema *subdigrafo-homeomorfo-k-anel* é $\mathcal{NP}$-completo já para $k = 2$. Além disso, se D é acíclico e o número de componentes de H é conhecido (k é fixo), então pelo Teorema 3 de [FHW80], o problema *subdigrafo-homeomorfo-k-anel* é resolvido em tempo polinomial. Agora voltamos ao problema como ele foi declarado (m_V dado parcialmente e H estruturado). Se D é acíclico e o número de componentes de H não é conhecido (k é livre), então o problema *subdigrafo-homeomorfo-k-anel* é $\mathcal{NP}$-completo, como veremos a seguir.

Lema 5.4. *O problema subdigrafo-homeomorfo-k-anel é $\mathcal{NP}$-completo se D é acíclico e o número de componentes de H não é conhecido (k é livre).*

Demonstração: Essa demonstração é similar à demonstração do Lema 5.1. É fácil ver que o problema está em $\mathcal{NP}$. Considere novamente o problema *k-caminhos-vértice-disjuntos* como declarado no Lema 5.1. Dados um digrafo acíclico D e k pares de vértices $(s_1, t_1), \ldots, (s_k, t_k)$ construímos um conjunto de pares de vértices $\mathcal{E}$, um digrafo acíclico de entrada D', um digrafo padrão H' com k componentes e um mapeamento parcial m_V da seguinte forma. O novo digrafo D' é uma cópia de D mais k novos vértices $t'_1, \ldots, t'_k$ e k novos arcos $s_1 \to t'_1, \ldots, s_k \to t'_k$. O conjunto de pares de vértices é $\mathcal{E} = ((t_1, t'_k), (t_2, t'_1), \ldots, (t_k, t'_{k-1}))$. Cada componente i de H' possui 3 vértices p_i, u_i e v_i, e 2 arcos $p_i \to u_i$ e $p_i \to v_i$ para $i = 1, \ldots, k$. E o mapeamento m_V é dado parcialmente através de $\mathcal{E}$, ou seja, $m_V(u_i) = t_i$ e $m_V(v_i) = t'_i$ para $i = 1, \ldots, k$. Note que o único arco que entra em t'_i é o arco $s_i \to t'_i$. Portanto, existem caminhos vértice-disjuntos $P_1, \ldots, P_k$ se e somente se existe um k-anel $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ não-degenerado. Assim, o problema *subdigrafo-homeomorfo-k-anel* é $\mathcal{NP}$-completo. □

5.3 O Problema um-k-anel com k Fixo

Considere o problema *k-caminhos-vértice-disjuntos*, como definido no Lema 5.1 com um digrafo acíclico D como entrada. Considere também o seguinte problema clássico em digrafos.

> O problema $(s - t)$-*caminho*: Dados um digrafo acíclico D e dois vértices s e t, decidir se existe um caminho dirigido de s para t em D.

Baseando-se na redução de Perl e Shiloach [PS78], Eppstein [Epp95] descreveu uma redução do problema *k-caminhos-vértice-disjuntos* para o problema $(s-t)$-*caminho*. Vamos rapidamente descrevê-la. Dados um digrafo acíclico D e k pares de vértices distintos $(s_1, t_1), \ldots, (s_k, t_k)$, primeiro são adicionados dois novos vértices s e t em D, e $2k$ novos arcos $s \to s_i$ e $t_i \to t$, para

$i = 1, \ldots, k$. Depois disso, o digrafo resultante é ordenado topologicamente. Considere f uma função que fornece uma ordenação topológica. Por último, é criado um novo digrafo acíclico D', onde um vértice $(v_1, v_2, \ldots, v_k)$ é uma k-upla ordenada de vértices de D e existe um arco de $(v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_k)$ para $(v_1, \ldots, v_{i-1}, w_i, v_{i+1}, \ldots, v_k)$ em D' para cada $i = 1, \ldots, k$, se as três seguintes condições são verdadeiras:

1. Existe um arco $v_i \to w_i$ em D.
2. Se $v_i = s$, então $w_i = s_i$ e se $w_i = t$ então $v_i = t_i$.
3. Ou $w_i = t$, ou $f(w_i) > \max(f(v_j))$, para cada $j \neq i$.

Essa redução é baseada na redução de Perl e Shiloach [PS78] para o problema 2-*caminhos-vértice-disjuntos* com um digrafo acíclico na entrada. O número de vértices e arcos do digrafo construído é, respectivamente, $O(n^k)$ e $O(mn^{k-1})$, onde n e m são os números de vértices e de arcos do digrafo D.

Lema 5.5 (Eppstein [Epp95]). *Fixado um k, e dados um digrafo acíclico D e k pares de vértices $(s_1, t_1), \ldots, (s_k, t_k)$, podemos construir um novo digrafo acíclico D' de tal forma que k caminhos vértice-disjuntos de s_i para t_i em D para $i = 1, \ldots, k$ correspondem a um caminho de $(s, s, \ldots, s)$ para $(t, t, \ldots, t)$ em D'.* □

Dado um caminho dirigido P' de $(s, s, \ldots, s)$ para $(t, t, \ldots, t)$ em D' podemos obter os caminhos vértice-disjuntos P_i de s_i para t_i olhando para cada arco de P' em ordem começando do vértice $(s, s, \ldots, s)$ e terminando no vértice $(t, t, \ldots, t)$. Se o arco $(v_1, v_2, \ldots, v_i, \ldots, v_{k-1}, v_k) \to (v_1, v_2, \ldots, w_i, \ldots, v_{k-1}, v_k)$ existe em P', então sabemos que o último vértice no caminho P_i é v_i e neste momento o vértice w_i é adicionado no final deste caminho.

Pelo Lema 5.5 sabemos que uma busca ordinária no digrafo de Eppstein para encontrar um caminho de $(s, \ldots, s)$ para $(t, \ldots, t)$, decide o problema *k-caminhos-vértice-disjuntos* em D. Assim, se k é fixo, temos um algoritmo polinomial para o problema *k-caminhos-vértice-disjuntos* com um digrafo acíclico dado na entrada.

Com o intuito de usar o digrafo de Eppstein para resolver o problema *um-k-anel*, vamos relacionar o problema *um-k-anel* com o problema *k-caminhos-vértice-disjuntos*. Estamos supondo que k é fixo. Considere que um grafo misto H e um subconjunto ordenado de arestas $\mathcal{E} = (u_1 - v_k, u_2 - v_1, \ldots, u_k - v_{k-1})$ de H é uma entrada do problema *um-k-anel*. Denote por D o digrafo induzido pelos arcos de H e suponha que ele seja acíclico. Vamos transformar essa entrada em várias entradas do problema 2*k-caminhos-vértice-disjuntos*. Se para alguma das entradas do último problema conclui-se que existem $2k$ caminhos vértice-disjuntos, então existe um k-anel $\mathcal{R} = (\mathcal{E}, \mathcal{P})$ em H. Uma entrada do problema 2*k-caminhos-vértice-disjuntos* é o próprio digrafo acíclico D mais $2k$ pares de vértices distintos $(s_1, u_1), (s'_1, v_1), \ldots, (s_k, u_k), (s'_k, v_k)$, onde $s_1, \ldots, s_k$ são vértices distintos de D e $s'_1, \ldots, s'_k$ são novos vértices criados em D. O grau de entrada de s'_i é zero e os seus filhos são os mesmos filhos de s_i para $i = 1, \ldots, k$. Veja um exemplo na Figura 5.8.

Para uma entrada do problema *um-k-anel*, o número de entradas criadas do problema 2*k-caminhos-vértice-disjuntos* é, no pior caso, o número de permutações possível de k vértices distintos $s_1, \ldots, s_k$ de D. Verificar, no pior caso, todas as entradas, garante a decisão do problema *um-k-anel*. O tempo gasto para decidir se existe um caminho desejado em uma entrada do problema 2*k-caminhos-vértice-disjuntos* é $O(mn^{2k-1})$, isto é, o número de arcos do digrafo de Eppstein construído para esta entrada. Temos que construir $O(n^k)$ entradas do problema, no pior caso. Portanto, o tempo gasto para decidir o problema *um-k-anel* com k fixo é $O(mn^{3k-1})$, no pior caso. Como a complexidade de tempo deste método é grande até mesmo quando k é pequeno, propomos alguns melhoramentos que experimentalmente deixaram mais baixo o tempo necessário para resolver o problema *um-k-anel* para as entradas consideradas.

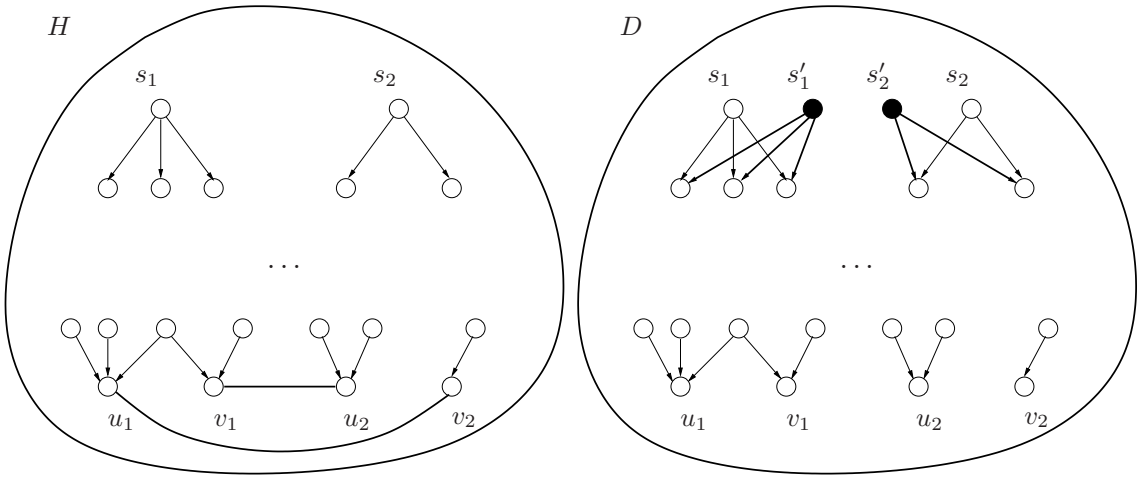


Figura 5.8: Na primeira figura temos uma entrada do problema *um-2-anel*: o grafo misto acíclico H e $\mathcal{E} = (u_1 - v_2, u_2 - v_1)$. Na segunda, fixados dois vértices s_1 e s_2 de H, temos uma entrada para o problema *4-caminhos-vértice-disjuntos*: os pares ordenados $(s_1, u_1), (s'_1, v_1), (s_2, u_2), (s'_2, v_2)$, e o digrafo acíclico D induzido pelos arcos de H.

5.3.1 Melhoramentos

Considere até o final desta seção uma entrada do problema *um-k-anel* como sendo um grafo misto acíclico H e um subconjunto ordenado de aresta $(u_1 - v_k, u_2 - v_1, \ldots, u_k - v_{k-1})$ de H. Vimos um método anteriormente que resolve o problema *um-k-anel* usando o digrafo de Eppstein. Sua complexidade de tempo é $O(mn^{3k-1})$. Vimos que este método constrói várias entradas do problema *2k-caminhos-vértice-disjuntos*. Para cada uma delas, o método constrói o digrafo de Eppstein. As entradas do problema *2k-caminhos-vértice-disjuntos* são formadas pelo digrafo acíclico D, induzido pelos arcos de H, e por $2k$ pares ordenados de vértices $(s_1, u_1), (s'_1, v_1), \ldots, (s_k, u_k), (s'_k, v_k)$, onde $s_1, \ldots, s_k$ são vértices distintos de D e s'_i são novos vértices criados em D com arcos para os filhos de s_i para $i = 1, \ldots, k$. Fixados $s_1, \ldots, s_k$, o método continua construindo o digrafo de Eppstein, denotado por $D_{s_1,\ldots,s_k}$, e assim decide se existem caminhos vértices disjuntos de s_i para u_i e de s_i para v_i para $i = 1, \ldots, k$ através de uma busca ordinária por um caminho de $(s, \ldots, s)$ para $(t, \ldots, t)$ em $D_{s_1,\ldots,s_k}$. Podíamos pensar primeiro em construir todo o digrafo $D_{s_1,\ldots,s_k}$ e depois procurar pelo caminho. No entanto, este digrafo é muito grande e demandaria muito tempo para construí-lo completamente. Portanto, nas nossas implementações, o digrafo é considerado implicitamente, sendo construído na medida em que a busca explora um novo vértice.

Os melhoramentos que apresentamos têm como objetivo, evitar a construção de uma entrada para o problema *2k-caminhos-vértice-disjuntos* e, uma vez criada tal entrada, evitar a exploração de vértices do digrafo de Eppstein que certamente não estão em nenhum caminho de $(s, \ldots, s)$ para $(t, \ldots, t)$.

O primeiro melhoramento usado pelo método é dado pela seguinte propriedade.

Propriedade 5.1. *Em um k-anel $(\mathcal{E}, \mathcal{P})$ de H, $\mathcal{P} = (P_1^{\mathcal{R}} Q_1, \ldots, P_k^{\mathcal{R}} Q_k)$, os caminhos internamente vértice-disjuntos P_i de s_i para u_i e Q_i de s_i para v_i para $i = 1, \ldots, k$ certificam que o vértice s_i é uma junção dos vértices u_i e v_i para $i = 1, \ldots, k$.*

Com isso, construímos previamente os conjuntos $\mathcal{J}_{u_i,v_i}$ para $i = 1, \ldots, k$, e os vértices $s_1, \ldots, s_k$, que antes eram escolhidos dentre quaisquer vértices distintos do digrafo D, agora são escolhidos dentre os conjuntos $\mathcal{J}_{u_i,v_i}$ para $i = 1, \ldots, k$.

O segundo melhoramento que propomos supõe uma entrada do problema *2k-caminhos-vértice-disjuntos*, ou seja, um digrafo acíclico D e $2k$ pares de vértices $(s_1, u_1), (s'_1, v_1), \ldots,$

$(s_k, u_k), (s'_k, v_k)$, onde s_i é uma junção dos vértices u_i e v_i para $i = 1, \ldots, k$. Assim, uma busca (em largura ou em profundidade) é realizada no digrafo $D_{s_1,\ldots,s_k}$ começando no vértice $(s, \ldots, s)$. Suponha que a busca encontra-se em um vértice $(x_1, y_1, \ldots, x_k, y_k)$. Suponha que ao processar o tal vértice, a existência do arco $(x_1, y_1, \ldots, x_j, y_j, \ldots, x_k, y_k) \to (x_1, y_1, \ldots, x_j, y'_j, \ldots, x_k, y_k)$ em $D_{s_1,\ldots,s_k}$ para algum j em $\{1, \ldots, k\}$ é constatada. No entanto, se não existe caminho de y'_j para v_j então não existe caminho de $(s, \ldots, s)$ para $(t, \ldots, t)$ passando pelo vértice $(x_1, y_1, \ldots, x_j, y'_j, \ldots, x_k, y_k)$. Para este caso, o segundo melhoramento não permite que a busca continue por este vértice. Note que o fecho transitivo de D deve ser conhecido. Veja um exemplo na Figura 5.9.

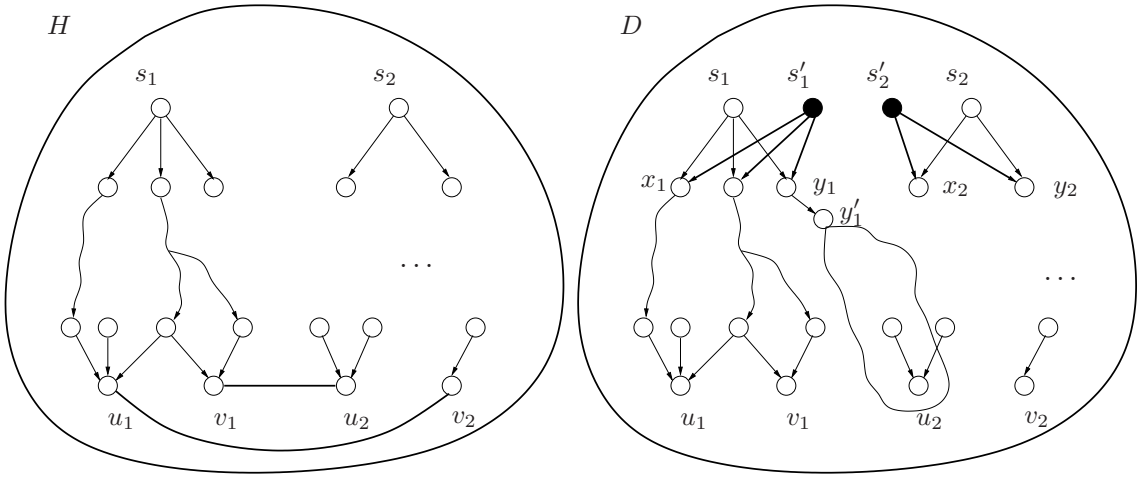


Figura 5.9: O vértice s_1 está em $\mathcal{J}_{u_1,v_1}$ e o vértice s_2 está em $\mathcal{J}_{u_2,v_2}$. Os descendentes de y_1 estão no triângulo ilustrado. Como não existe caminho de y'_1 para v_1, o segundo melhoramento não permite que a busca continue pelo vértice (x_1, y'_1, x_2, y_2) de D_{s_1,s_2}.

O terceiro melhoramento também supõe uma entrada do problema $2k$-*caminhos-vértice-disjuntos*, um digrafo acíclico D e os pares ordenados de vértices como anteriormente. Este melhoramento evita explorar alguns vértices do digrafo de Eppstein que o segundo melhoramento deixa explorar. Considere o grafo fluxo redutível $D[S^p]$ com raiz em p, para cada vértice p de D, onde S^p é o conjunto dos descendentes de p em D. Para cada vértice q descendente de p, considere o conjunto $\mathcal{S}_{p \rightsquigarrow q}$ formado por todos os dominadores não triviais de q e diferentes de q. Em outras palavras, se p é diferente de q, não existem caminhos vértice-disjuntos de p para q e p não é pai de q, então $\mathcal{S}_{p \rightsquigarrow q}$ é diferente de vazio e todos os caminhos de p para q passam por qualquer vértice em $\mathcal{S}_{p \rightsquigarrow q}$. Agora suponha um cenário parecido com o descrito anteriormente. Suponha que uma busca foi iniciada no digrafo $D_{s_1,\ldots,s_k}$ e que ela encontra-se em um vértice $(x_1, y_1, \ldots, x_j, y_j, \ldots, x_k, y_k)$. Então um caminho de $(s, \ldots, s)$ para $(x_1, y_1, x_2, y_2, \ldots, x_k, y_k)$ certifica de que existem caminhos internamente vértice-disjuntos de s_i para x_i e de s'_i para y_i, para $i = 1, \ldots, k$. Suponha que ao processar tal vértice é constatado a existência do arco $(x_1, y_1, \ldots, x_j, y_j, \ldots, x_k, y_k) \to (x_1, y_1, \ldots, x'_j, y_j, \ldots, x_k, y_k)$ neste digrafo para algum j em $\{1, \ldots, k\}$. Desta vez, suponha que existe um caminho de x'_j para u_j, porém, ou $\mathcal{S}_{x'_j \rightsquigarrow u_j} \cap \mathcal{S}_{x_i \rightsquigarrow u_i} \neq \emptyset$, ou $\mathcal{S}_{x'_j \rightsquigarrow u_j} \cap \mathcal{S}_{y_i \rightsquigarrow v_i} \neq \emptyset$ para algum i em $\{1, \ldots, k\}$, $i \neq j$. Portanto, o segundo melhoramento permite que a busca continue pelo vértice $(x_1, y_1, \ldots, x'_j, y_j, \ldots, x_k, y_k)$, no entanto, sabemos que não existem caminhos vértice-disjuntos de x'_j para u_j e, ou de x_i para u_i, ou de y_i para v_i. Assim, ao constatar a interseção não vazia do conjunto $\mathcal{S}_{x'_j \rightsquigarrow u_j}$ com algum conjunto $\mathcal{S}_{x_i \rightsquigarrow u_i}$ ou $\mathcal{S}_{y_i \rightsquigarrow v_i}$ para $i = 1, \ldots, k$, o terceiro melhoramento não permite que a busca continue por este vértice. Veja um exemplo na Figura 5.10.

Agora vamos falar como construir os conjuntos $\mathcal{S}_{p \rightsquigarrow q}$ para todo par de vértices p e q onde p é um ancestral de q. Dado um digrafo acíclico D, considere o grafo fluxo redutível $D[S^p]$ para

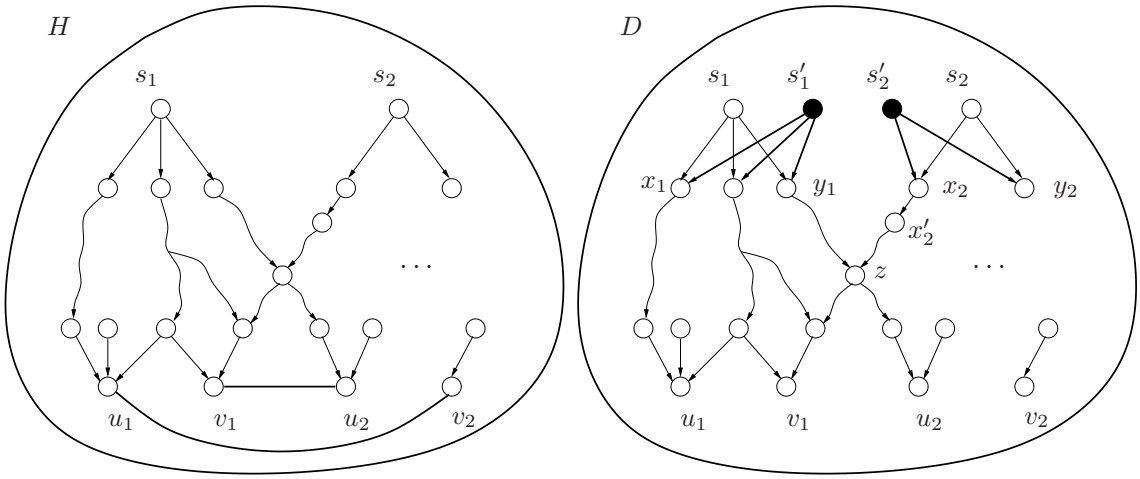


Figura 5.10: O vértice s_1 está em $\mathcal{J}_{u_1,v_1}$ e o vértice s_2 está em $\mathcal{J}_{u_2,v_2}$. Como $\mathcal{S}_{y_1 \rightsquigarrow v_1} \cap \mathcal{S}_{x_2 \rightsquigarrow u_2} \neq \emptyset$ (o vértice z aparece em ambos os conjuntos), o terceiro melhoramento não permite que a busca continue pelo vértice (x_1, y_1, x'_2, y_2) de D_{s_1,s_2}.

cada vértice p em D, onde S^p é o conjunto de descendentes de p em D. Construa uma partição em estrelas, r^p, do digrafo $D[S^p]$. Vimos no Lema 4.10 que $r^p(q)$ é o dominador mais alto não trivial de q em $D[S^p]$. Com isso, para todo par de vértices onde p é ancestral de q, temos

$$\mathcal{S}_{p \rightsquigarrow q} \leftarrow \begin{cases} \emptyset, & \text{se } p = q, \text{ ou } p \text{ é pai de } q \text{ em } D, \text{ ou } p \in \mathcal{J}_{q,q}; \text{ ou} \\ \{r^p(q)\} \cup \mathcal{S}_{r^p(q) \rightsquigarrow q}, & \text{caso contrário.} \end{cases}$$

A construção do conjunto citado considera uma ordenação topológica dos vértices de D. Para que o par (p, q) seja processado, todos os pares (p', q) onde p' vem depois que p na ordenação topológica já devem ter sido processados.

5.4 Experimentos

O principal objetivo desta seção é descrever os resultados experimentais obtidos na solução do problema *um-k-anel* aplicado a algumas sociedades. Os dados de cada sociedade, compostos por uma rede de parentesco e seus casamentos, foram fornecidos pelos professores Dr. Marcio Ferreira da Silva e Dr. João dal Poz Neto. As sociedades são:

- *Enawenê-Nawês*, do Noroeste do Mato Grosso, com 789 indivíduos (vértices), 1.368 relações de parentesco (arcos) e 170 relações de afinidade (arestas);
- *Arara*, do Sul do Pará, atualmente com 105 vértices, 197 arcos e 48 arestas;
- *Deni*, do Sul do Amazonas, atualmente com 875 vértices, 1.589 arcos e 333 arestas;
- *Xavante*, do Mato Grosso, atualmente com 459 vértices, 713 arcos e 254 arestas;
- *Irantxe-Myky*, Noroeste de Mato Grosso, com 618 vértices, 1.003 arcos e 177 arestas;
- *Arapium*, do Pará, com 1.214 vértices, 1.792 arcos e 291 arestas; e
- *Zoró*, do Sul do Amazonas, com 755 vértices, 1.224 arcos e 201 arestas;.

5.4.1 Experimentos - Primeira Parte

Na primeira parte dos experimentos, é feito um levantamento estatístico para cada sociedade. Para algumas estatísticas consideramos um par formado por dois indivíduos casados, mas não entre si, isto é, um par (u, v) onde u está em um casamento e v está em outro casamento. Chamamos um par desse tipo de *colegas em casamentos*. Note que os pares colegas em casamentos aparecem na definição de um k-anel para $k \geq 2$. Veja na Figura 5.11 os possíveis colegas em casamentos para os casamentos $u - x$ e $v - y$.

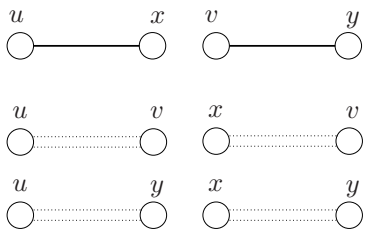


Figura 5.11: Os casamentos são $u - x$ e $v - y$. Os colegas em casamento estão representados nesta figura pelos indivíduos ligados por pontilhados duplos.

As estatísticas que queremos saber para cada sociedade são:

1º. Quantos casamentos têm exatamente i junções, para inteiros não-negativos $i = 0, 1, \ldots, n$;

2º. Quantos casamentos têm u como junção, para cada vértice u;

3º. Quantos colegas em casamentos têm exatamente i junções, para inteiros não-negativos $i = 0, 1, \ldots, n$; e

4º. Quantos colegas em casamentos têm u como junção, para cada vértice u.

A primeira estatística é obtida resolvendo, para cada sociedade, o problema *todas-junções-k-pares*. Os dados de entrada são o digrafo acíclico da sociedade e seus casamentos. A segunda estatística é obtida resolvendo, para cada sociedade, o problema *s-junção-k-pares* para cada indivíduo s da sociedade. Os dados de entrada são o digrafo acíclico da sociedade, seus casamentos e um vértice s. A terceira e quarta estatísticas são obtidas da mesma forma, porém, no lugar dos casamentos, são considerados todos os possíveis colegas em casamentos. Acreditamos que estas informações nos dão uma ideia de quão difícil é resolver o problema *um-k-anel* para cada sociedade.

A Figura 5.12 apresenta os resultados da primeira estatística para cada sociedade. Um número i no eixo x representa uma quantidade de vértices que são junções de algum casamento ($0 \leq i \leq n$, ou seja, de nenhum vértice até todos os vértices) e um número j no eixo y representa uma quantidade de casamentos ($0 \leq j \leq k$, ou seja, de nenhum casamento até todos os casamentos). Em outras palavras, com essa estatística sabemos a quantidade de casamentos $u - v$ com $|\mathcal{J}_{u,v}| = i$ para $i = 0, 1, \ldots, n$. Com exceção da sociedade Arara, note o rápido decaimento nos gráficos. Por exemplo, no gráfico dos Enawenê-Nawês 162 casamentos não têm junção e 8 casamentos têm 2 junções. Note também uma certa oscilação nos gráficos. Por exemplo, na sociedade Zoró, 175 casamentos não têm junção, 6 casamentos têm 1 junção, 15 casamentos têm 2 junções, 1 casamentos tem 3 junções, 3 casamentos têm 4 junções e 1 casamento tem 5 junções. Excluindo os casamentos cujos conjuntos de junções são vazios, o tamanho dos conjuntos que mais ocorre é 2.

A Figura 5.13 apresenta os resultados da segunda estatística para cada sociedade. Um número i no eixo x representa um indivíduo na sociedade ($0 \leq i \leq n - 1$) e um número j no eixo y representa uma quantidade de casamentos ($0 \leq j \leq k$). Essa estatística nos diz

quantas vezes um indivíduo aparece como uma junção de um casamento. Ordenamos o eixo x em ordem decrescente no número de casamentos que participa cada indivíduo, isto é, um valor pequeno para i representa um indivíduo que é uma junção de muitos casamentos. A figura não mostra os indivíduos que não são junções de casamentos. Por exemplo, no gráfico da sociedade Enawenê-Nawês somente quatro indivíduos são junções. Os indivíduos 0 e 1 são junções de 6 casamentos, e os indivíduos 2 e 3 são junções de 2 casamentos. Na sociedade Deni, existem 48 indivíduos que são junções de casamentos e existem dois indivíduos que são junções de mais que 90 casamentos. Observe que a sociedade Enawenê-Nawês apresenta poucos indivíduos que são junções de casamentos.

A Figura 5.14 apresenta os resultados da terceira estatística para cada sociedade. De maneira similar à primeira estatística, um número i no eixo x representa uma quantidade de vértices que são junções de algum colega em casamento ($0 \leq i \leq n$) e um número j no eixo y representa uma quantidade de colegas em casamentos ($0 \leq j \leq 2(k^2 - k)$, ou seja, dados k casamentos, de nenhum colega em casamento até todos os colegas em casamentos). Essa estatística nos diz a quantidade de colegas em casamentos $u - v$ com $|\mathcal{J}_{u,v}| = i$ para $i = 0, 1, \ldots, n$. Mais uma vez temos um rápido decaimento nos gráficos. Na sociedade Arara, o decaimento ocorre a partir do 2 no eixo x. Veja na figura que em todas as sociedades, com exceção da sociedade Arara, a grande maioria dos colegas em casamentos não têm junção. Em geral, o tamanho do conjunto das junções para quaisquer colegas em casamentos não é grande. Na sociedade Deni, existem 9 colegas em casamentos cujos conjuntos de junções têm tamanho igual a 12. Da mesma forma que anteriomente, excluindo os colegas em casamentos cujos conjuntos de junções são vazios, o tamanho dos conjuntos que mais ocorre é 2.

A Figura 5.15 apresenta os resultados da quarta estatística para cada sociedade. Um número i no eixo x representa um indivíduo na sociedade ($0 \leq i \leq n - 1$) e um número j no eixo y representa uma quantidade de colegas em casamentos ($0 \leq j \leq 2(k^2 - k)$). Essa estatística nos diz quantas vezes um indivíduo aparece como uma junção de um colega em casamento. Mais uma vez, ordenamos o eixo x em ordem decrescente no número de colegas em casamentos, isto é, um valor pequeno para i representa um indivíduo que é uma junção de muitos colegas em casamentos. Para a sociedade Deni, mostramos somente os indivíduos que são junções de pelo menos 800 colegas em casamentos. Para as demais sociedades, pelo menos 100. Existem 2 indivíduos na sociedade Xavante que são junções de mais que 6800 colegas em casamentos. Na sociedade Deni, 2 indivíduos são junções de mais que 44.000. Note a diferença entre os gráficos da sociedade Enawenê-Nawês nas Figuras 5.13 e 5.15. Na Figura 5.15, a sociedade Enawenê-Nawês apresenta muitos indivíduos que são junções de colegas em casamentos.

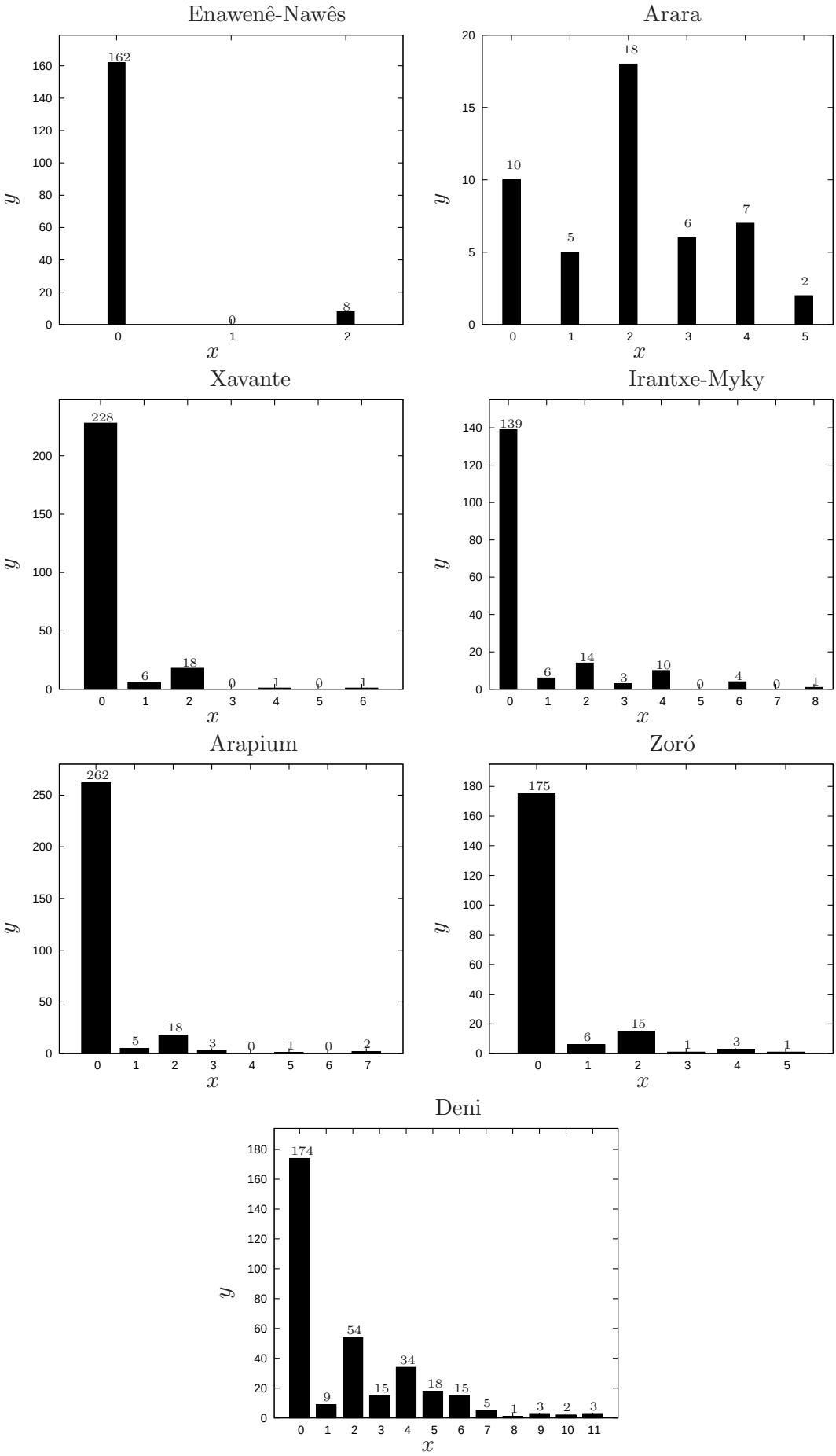


Figura 5.12: 1º estatística - Para inteiros não-negativos $i = 0, 1, \ldots, n$, quantos casamentos têm exatamente i junções?

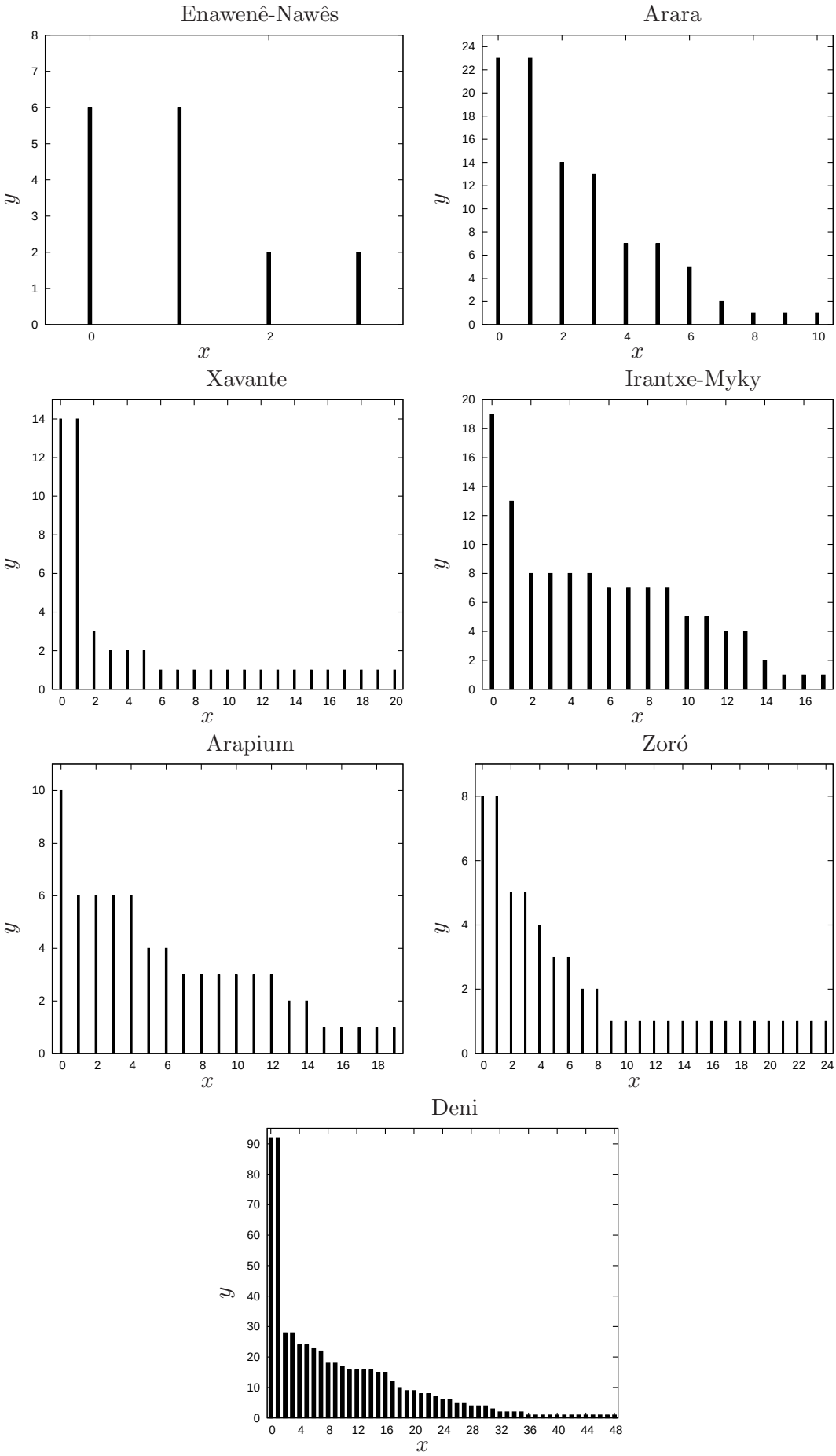


Figura 5.13: 2º estatística - Para cada vértice u, quantos casamentos têm u como junção?

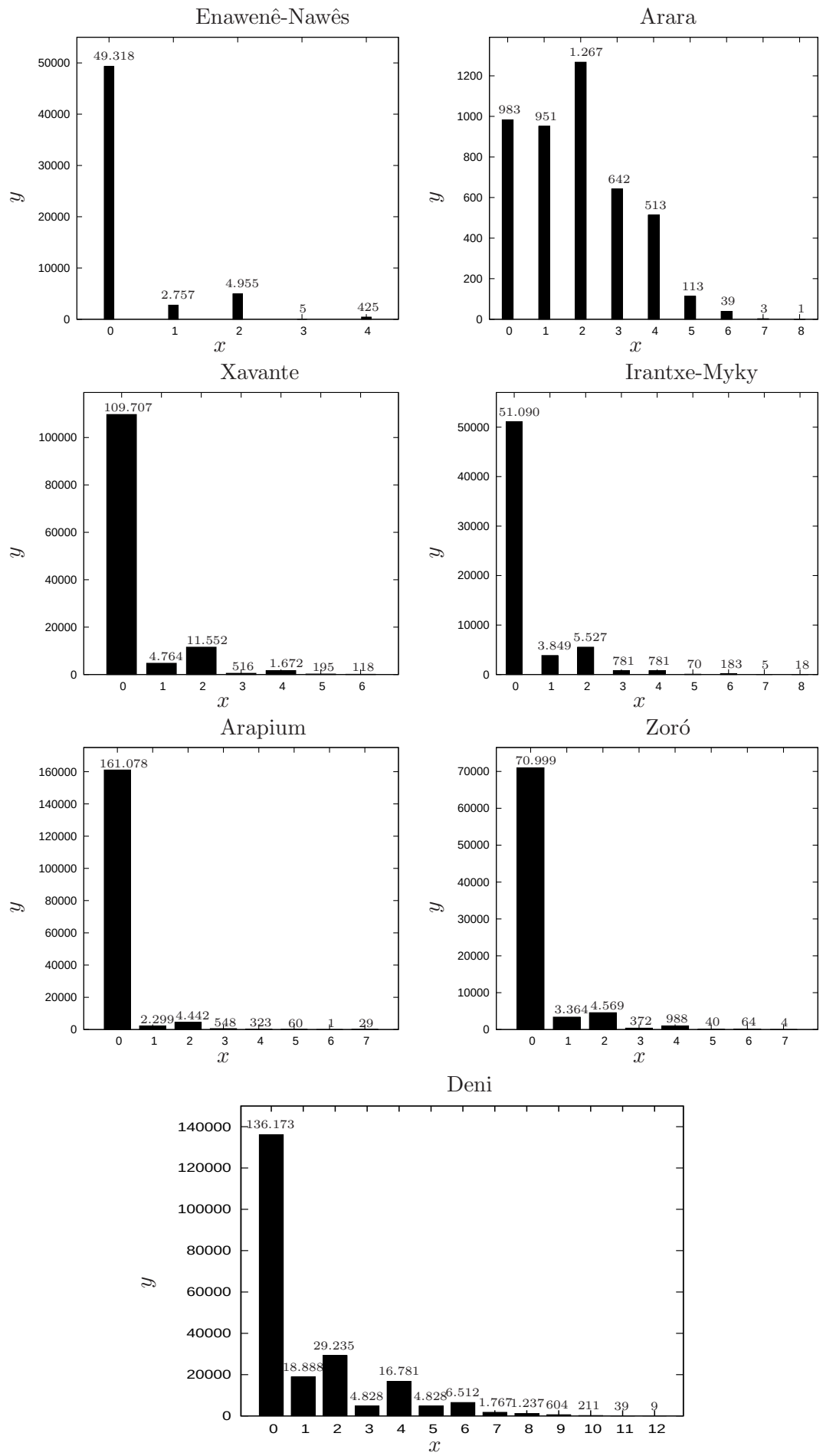


Figura 5.14: 3º estatística - Para inteiros não-negativos $i = 0, 1, \ldots, n$, quantos colegas em casamentos têm exatamente i junções?

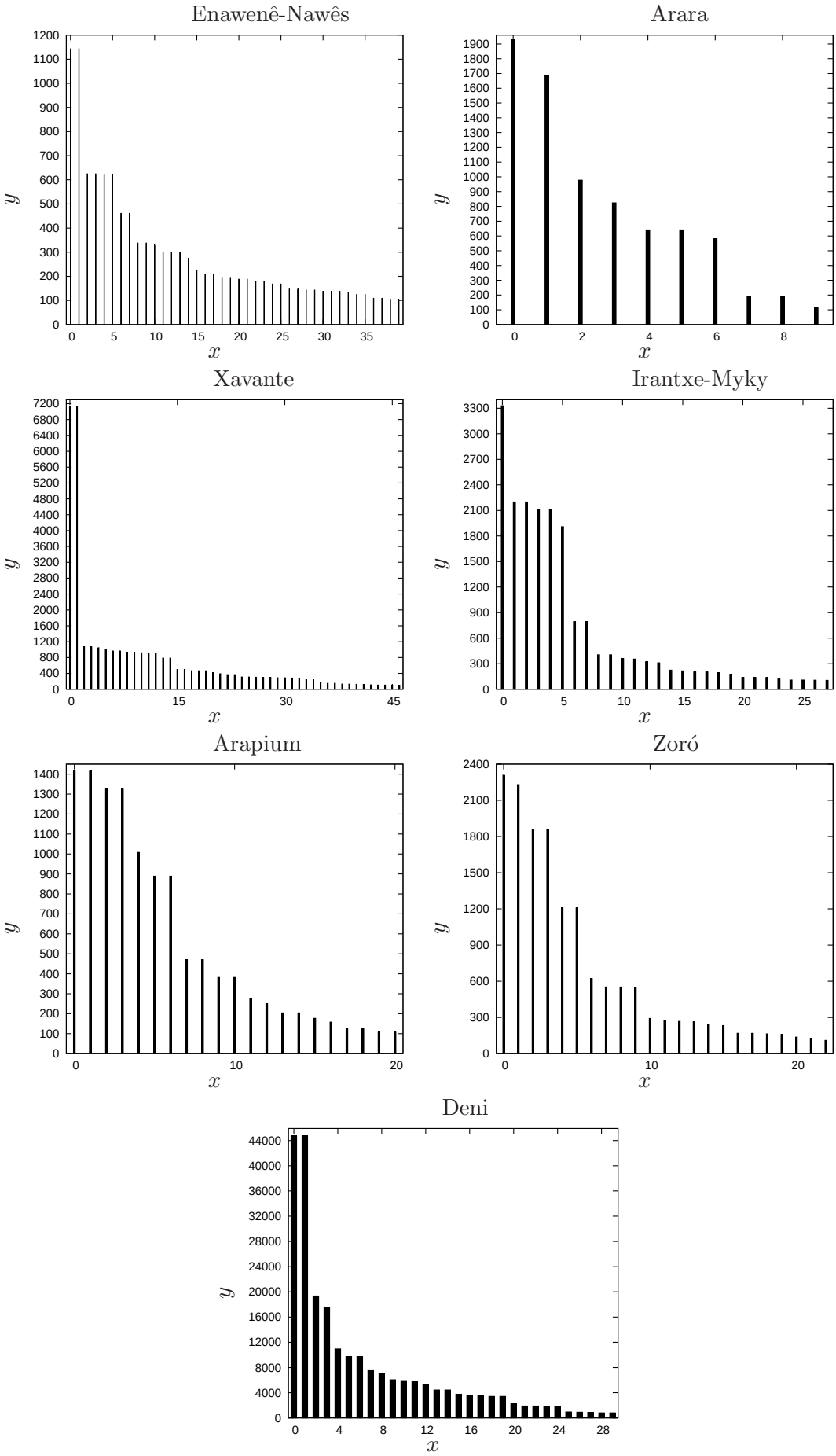


Figura 5.15: 4º estatística - Para cada vértice u, quantos colegas em casamentos tem u como junção?

5.4.2 Experimentos - Segunda Parte

Agora começamos a segunda parte dos experimentos onde descrevemos os resultados obtidos na solução do problema *um-k-anel* ($k = 2, 3$). Lembre que neste problema, dado um grafo misto H e um subconjunto ordenado $\mathcal{E}$ com k arestas de H, queremos saber se existe ou não um k-anel $(\mathcal{E}, \mathcal{P})$ em H. O anel que queremos encontrar aqui é não-degenerado. Os experimentos foram realizados sobre as 7 sociedades citadas anteriormente: Enawenê-Nawês, Arara, Deni, Xavante, Irantxe-Myky, Arapium e Zoró. Para cada sociedade, criamos 2.000 entradas do problema *um-k-anel* ($k = 2, 3$). Uma entrada $\mathcal{I} = (H, \mathcal{E})$ possui o grafo misto H de uma sociedade, e um subconjunto ordenado $\mathcal{E}$ do conjunto de arestas de H de tamanho k ($k = 2, 3$). Fixada uma sociedade com grafo misto H, para completar uma entrada do problema *um-k-anel*, um subconjunto ordenado de arestas de H foi escolhido ao acaso, uniformemente e sem repetição. Implementamos o nosso programa na linguagem C++ e o código fonte está disponível sobre a licença GPL no sítio do autor deste texto. Executamos os experimentos em uma máquina 64-bits com Linux Gentoo, 64GB de memória RAM e dois processadores Intel Xeon E5620 2.40 GHz. O nosso programa não foi paralelizado.

Os primeiros resultados experimentais foram obtidos sobre o problema *um-2-anel* aplicando somente o primeiro melhoramento (M1), isto é, em um 2-anel $(\mathcal{E}, \mathcal{P})$ de H, onde $\mathcal{E} = (u_1 - v_2, u_2 - v_1)$ e $\mathcal{P} = (P_1^{\mathcal{R}} Q_1, P_2^{\mathcal{R}} Q_2 k)$, os caminhos internamente vértice-disjuntos P_i de s_i para v_i e Q_i de s_i para v_i para $i = 1, 2$ certificam que o vértice s_i é uma junção dos vértices u_i e v_i para $i = 1, 2$. Os resultados estão na Tabela 5.1. Para cada entrada do problema, uma vez construído o digrafo de Eppstein, aplicamos sobre ele uma busca em profundidade com um limite no tempo de 130 segundos. A coluna TLE da tabela mostra o número de instâncias em que esse limite ocorreu. As soluções das 2.000 instâncias para cada sociedade devem terminar em 1 hora, caso contrário, interrompemos a execução do programa. Por isso, para algumas sociedades, o número de instâncias testadas é menor que 2.000. O tempo médio e o desvio padrão estão nas colunas Tempo e DP. A coluna M1 representa o número de vezes que o digrafo de Eppstein não foi construído pois, ou $\mathcal{J}_{u_1,v_1}$, ou $\mathcal{J}_{u_2,v_2}$ é igual ao conjunto vazio. Observe a grande diferença nos valores das respostas positivas (sim existe um 2-anel) e negativas (não existe um 2-anel) para todas as sociedades com exceção da sociedade Arara. Note os altos valores na coluna M1. Perceba que não é tão fácil dizer que não existe um 2-anel para a sociedade Arara, como pode ser visto na coluna do tempo médio gasto e do desvio padrão.

Tabela 5.1: Eppstein com DFS e M1 - Problema *um-2-anel*

		Positiva			Negativa				
Sociedade	Instâncias	Qtd	Tempo	DP	Qtd	M1	Tempo	DP	TLE
Enawenê-Nawês	2.000	37	27,19	28,24	1.958	1.957	0,07	1,52	5
Arara	295	141	13,14	17,65	146	131	4,18	17,63	8
Deni	138	5	54,28	30,61	108	108	0,05	0,00	25
Xavante	2.000	33	3,68	5,04	1.967	1.962	0,09	2,02	0
Irantxe-Myky	756	11	18,62	20,18	722	716	0,43	5,60	23
Arapium	2.000	6	21,08	24,76	1.990	1.988	0,17	2,57	4
Zoró	1.866	25	46,82	42,12	1.824	1.819	0,05	0,57	17

Os próximos resultados experimentais foram obtidos sobre os problemas *um-2-anel* e *um-3-anel* aplicando todos os melhoramentos (M1, M2 e M3) como definidos na seção anterior. Os melhoramentos M1 e M3 podem evitar a construção do digrafo de Eppstein. Por isso, nas Tabelas 5.2, 5.3, 5.4 e 5.5 aparece uma coluna M1,3 que significa quantas vezes os melhoramentos M1 ou M3 evitaram a construção deste digrafo. Verificamos experimentalmente que os melhoramentos em conjunto trabalharam muito bem. Todas as 2.000 entradas para cada sociedade

foram resolvidas rapidamente. Para estes casos, uma vez criado o digrafo de Eppstein, aplicamos as buscas em largura (Tabelas 5.2 e 5.4) e em profundidade (Tabelas 5.3 e 5.5). Os tempos médios gastos para as buscas em profundidade e em largura foram similares. Note como estão bem distribuídos os 2-anéis na amostra da sociedade Arara, e um pouco menos distribuídos, na sociedade Deni (colunas Qtd - Positiva/Negativa). Existem poucos 2-anéis na amostra da sociedade Arapium. Também existem poucos 3-anéis nas amostras de todas as sociedades, com exceção das sociedades Arara e Deni.

Tabela 5.2: Eppstein com BFS, M1, M2 e M3 - Problema *um-2-anel*

	Positiva			Negativa			
Sociedade	Qtd	Tempo	DP	Qtd	M1,3	Tempo	DP
Enawenê-Nawês	42	0,122	0,005	1.958	1.958	0,040	0,005
Arara	966	0,003	0,001	1.034	908	0,002	0,001
Deni	343	0,151	0,007	1.657	1.644	0,053	0,018
Xavante	33	0,043	0,003	1.967	1.964	0,014	0,003
Irantxe-Myky	81	0,076	0,004	1.919	1.912	0,026	0,007
Arapium	7	0,282	0,011	1.993	1.993	0,092	0,012
Zoró	38	0,112	0,007	1.962	1.959	0,038	0,007

Tabela 5.3: Eppstein com DFS, M1, M2 e M3 - Problema *um-2-anel*

	Positiva			Negativa			
Sociedade	Qtd	Tempo	DP	Qtd	M1,3	Tempo	DP
Enawenê-Nawês	42	0,120	0,005	1.958	1.958	0,040	0,005
Arara	966	0,003	0,001	1.034	908	0,001	0,001
Deni	343	0,149	0,006	1.657	1.644	0,053	0,018
Xavante	33	0,042	0,003	1.967	1.964	0,014	0,003
Irantxe-Myky	81	0,076	0,004	1.919	1.912	0,026	0,007
Arapium	7	0,280	0,007	1.993	1.993	0,092	0,012
Zoró	38	0,112	0,005	1.962	1.959	0,037	0,007

Tabela 5.4: Eppstein com BFS, M1, M2 e M3 - Problema *um-3-anel*

	Positiva			Negativa			
Sociedade	Qtd	Tempo	DP	Qtd	M1,3	Tempo	DP
Enawenê-Nawês	6	0,153	0,003	1.994	1.993	0,040	0,005
Arara	517	0,088	0,093	1.483	1.224	0,047	0,249
Deni	102	0,281	0,146	1.898	1.886	0,075	0,713
Xavante	5	0,094	0,026	1.995	1.995	0,014	0,002
Irantxe-Myky	6	0,125	0,013	1.994	1.992	0,026	0,007
Arapium	1	0,355	0,000	1.999	1.999	0,092	0,005
Zoró	4	0,191	0,044	1.996	1.996	0,037	0,003

Tabela 5.5: Eppstein com DFS, M1, M2 e M3 - Problema *um-3-anel*

	Positiva			Negativa			
Sociedade	Qtd	Tempo	DP	Qtd	M1,3	Tempo	DP
Enawenê-Nawês	6	0,130	0,003	1.994	1.993	0,040	0,005
Arara	517	0,038	0,047	1.483	1.224	0,038	0,187
Deni	102	0,204	0,089	1.898	1.886	0,069	0,489
Xavante	5	0,061	0,010	1.995	1.995	0,014	0,002
Irantxe-Myky	6	0,099	0,008	1.994	1.992	0,026	0,007
Arapium	1	0,300	0,000	1.999	1.999	0,091	0,005
Zoró	4	0,145	0,022	1.996	1.996	0,037	0,003

Uma vez criado o digrafo de Eppstein, os melhoramentos M2 e M3 podem evitar computações. Vemos na Tabela 5.6, para os experimentos realizados no problema *um-3-anel* com busca em profundidade, o número médio de arcos do digrafo de Eppstein que efetivamente foram explorados (que causam uma tentativa de inserção de um vértice do digrafo na pilha da busca), juntamente com o seu desvio padrão. Vemos também a quantidade de entradas, o número médio de computações evitadas (o número de vértices do digrafo de Eppstein não explorados), e o seu desvio padrão para os melhoramentos M2 e M3. Por exemplo, para a sociedade Arara, foram evitadas em média 1.449.260 computações por M2 em 776 instâncias e foram evitadas em média 1.247 computações por M3 em 75 instâncias. Como esperado, o número de computações evitadas por M2 é maior que por M3. Os valores da média e do desvio padrão que aparecem na Tabela 5.6 foram truncados na parte inteira. Observe o alto desvio padrão para as computações evitadas por M2 e M3 nas sociedades Arara, Deni.

Tabela 5.6: Arcos, M2 e M3 no digrafo de Eppstein - DFS *um-3-anel*

	Arcos		Melhoramento M2			Melhoramento M3		
Sociedade	Média	DP	Qtd	Média	DP	Qtd	Média	DP
Enawenê-Nawês	17.829	3.623	7	369.969	260.158	0	0	0
Arara	31.675	28.977	776	1.449.260	3.686.688	75	1.247	4.096
Deni	51.954	53.910	114	5.835.800	32.399.209	30	9.181	26.175
Xavante	27.563	12.240	5	257.125	103.016	0	0	0
Irantxe-Myky	28.389	11.310	8	673.830	701.551	0	0	0
Arapium	32.846	0	1	412.858	0	0	0	0
Zoró	27.498	10.664	4	442.966	323.007	0	0	0

Por último, desenvolvemos uma ferramenta, hoje em uso pelos colegas antropólogos, que resolve o problema *todos-k-anéis*. Dados k colegas em casamentos $(u_1, v_1), \ldots, (u_k, v_k)$ e os seus respectivos conjuntos de junções $\mathcal{J}_{u_1,v_1}, \ldots, \mathcal{J}_{u_k,v_k}$, o método da aplicação primeiro encontra todos os caminhos de s_i^j para u_i, e de s_i^j para v_i para $i = 1, \ldots, k$ e $j = 1, \ldots, |\mathcal{J}_{u_i,v_i}|$, e depois lista aqueles caminhos que são internamente vértice-disjuntos entre si. Aplicamos esse método para decidir o problema *um-k-anel* ($k = 2, 3$) parando o método quando um tal anel é encontrado. Os resultados foram satisfatórios e estão descritos nas Tabelas 5.7 e 5.8.

Tabela 5.7: Todos os caminhos - Problema *um*-2-*anel*

	Positiva			Negativa		
Sociedade	Qtd	Tempo	DP	Qtd	Tempo	DP
Enawenê-Nawês	42	0,0440	0,0046	1.958	0,0436	0,0043
Arara	966	0,0013	0,0008	1.034	0,0012	0,0007
Deni	343	0,0511	0,0046	1.657	0,0496	0,0045
Xavante	33	0,0153	0,0023	1.967	0,0158	0,0022
Irantxe-Myky	81	0,0280	0,0031	1.919	0,0280	0,0031
Arapium	7	0,0932	0,0074	1.993	0,0978	0,0079
Zoró	38	0,0400	0,0038	1.962	0,0398	0,0041

Tabela 5.8: Todos os caminhos - Problema *um*-3-*anel*

	Positiva			Negativa		
Sociedade	Qtd	Tempo	DP	Qtd	Tempo	DP
Enawenê-Nawês	6	0,0428	0,0024	1.994	0,0404	0,0033
Arara	517	0,0012	0,0007	1.483	0,0012	0,0007
Deni	102	0,0517	0,0038	1.898	0,0497	0,0038
Xavante	5	0,0156	0,0024	1.995	0,0141	0,0020
Irantxe-Myky	6	0,0261	0,0020	1.994	0,0252	0,0027
Arapium	1	0,0980	0,0000	1.999	0,0918	0,0054
Zoró	4	0,0357	0,0038	1.996	0,0369	0,0032

6

Conclusão

Neste trabalho, descrevemos alguns algoritmos para os seguintes problemas sobre junções: *representante-acp-todos-pares*, *todos-acp-todos-pares* e *representante-junção-todos-pares*. Uma junção dos vértices u e v é um vértice s com caminhos vértice-disjuntos de s para u e de s para v. Um ancestral comum mais próximo (acp) dos vértices u e v é uma junção s tal que não existe um caminho de s para s', para qualquer vértice comum s' dos vértices u e v. A entrada de cada um desses problemas é um digrafo acíclico D, e os algoritmos pré-processam D construindo uma estrutura de dados capaz de responder rapidamente uma junção (ou todas as junções) para qualquer par de vértices. Os algoritmos mais rápidos para cada um dos problemas citados anteriormente usam o algoritmo para multiplicar matrizes. O problema *todas-junções-todos-pares* foi definido nesta tese. Ele possui uma relação forte com a aplicação da Antropologia e a sua solução é usada como ferramenta na aplicação. A Tabela 6.1 destaca os melhores tempos de pré-processamento para cada um desses problemas. Um dos resultados obtidos nesta tese aparece em negrito. Para a leitura da Tabela 6.1, considere que o digrafo da entrada possui n vértices.

Tabela 6.1: Algoritmos rápidos para problemas sobre junções - todos os pares

Problema	Tempo
representante-acp-todos-pares	$O(n^{2,5719})$
representante-junção-todos-pares	$O(n^{\omega})$
todos-acps-todos-pares	$O(n^{3,2567})$
todas-junções-todos-pares	$\mathbf{O(n^3)}$

A principal contribuição deste trabalho é um algoritmo simples, elegante, eficiente e de fácil implementação para o problema *s-junção-todos-pares*. São dados um digrafo acíclico D e um vértice s de D. O digrafo D é pré-processado em tempo linear, construindo uma estrutura de dados capaz de responder em tempo constante, se s é ou não uma junção dos vértices u e v para todo par de vértices u e v.

Se considerarmos que os k pares para os quais queremos saber uma junção são também dados na entrada, então surgem naturalmente os seguintes problemas: *representante-acp-k-pares*, *todos-acps-k-pares*, *representante-junção-k-pares* e *todas-junções-k-pares*. Bons resultados foram obtidos ao usarmos o algoritmo para o problema *s-junção-todos-pares* em cada um desses problemas. A Tabela 6.2 mostra o tempo de pré-processamento para cada problema ao adicionar restrições no tamanho do digrafo. Mais uma vez, alguns resultados obtidos nesta tese aparecem em negrito. Considere que o digrafo da entrada possui n vértices e m arcos.

A importância do algoritmo para o problema *s-junção-todos-pares* nesta tese é óbvia. Foi através dele que derivamos um algoritmo para o problema *s-arco-junção-k-pares*, descrevemos de maneira diferente o método linha-coluna que multiplica matrizes booleanas, encontramos uma relação min-max em grafos fluxos e construímos uma árvore de dominadores para grafos fluxos

Tabela 6.2: Algoritmos rápidos para problemas sobre junções - k pares

Problema	Tempo	Restrição
representante-acp-k-pares	$\mathbf{o(n^{2,5719})}$	$m + k = o(n^{1,5719})$
representante-junção-k-pares	$\mathbf{o(n^{\omega})}$	$m + k = o(n^{1,373})$
todos-acps-k-pares	$\mathbf{o(n^{3,2567})}$	$k = o(n^{1,2567})$
todas-junções-k-pares	$\mathbf{O(n(m+k))}$	-

redutíveis. O algoritmo para o problema *s-junção-todos-pares* foi publicado no artigo com título *Algorithms for Junctions in Acyclic Digraphs*, no livro *Facets of Combinatorial Optimization*, páginas 175-194, 2013 [FF13].

O problema da Antropologia *um-k-anel* é $\mathcal{NP}$-completo e, quando visto como um problema parametrizado, ele é $\mathcal{W}[1]$-difícil. Mesmo com a dificuldade deste problema, propomos soluções usando o método de Eppstein [Epp95] com alguns melhoramentos. Experimentos mostraram o bom funcionamento da nossa solução. Desenvolvemos uma ferramenta para o problema *todos-k-anéis* para $k = 2, 3$ que vem sendo usada atualmente pelos nossos parceiros da antropologia.

Durante o desenvolvimento desta tese, deparamos com problemas que não conseguimos resolver e que continuam em aberto. Em seguida citamos alguns deles. É possível resolver o problema *todos-acps-todos-pares* em tempo $O(n^3)$? É possível resolver o problema *todas-junções-todos-pares* em tempo $O(m + K)$, onde K é o número total de junções para todo par de vértices? É possível adaptar o algoritmo que resolve o problema *s-junção-todos-pares* considerando um digrafo geral na entrada? É possível adaptar o algoritmo que constrói uma árvore de dominadores de um grafo fluxo redutível de tal forma que seu tempo seja linear?

Outro problema que não consideramos nesta tese, mas nossos parceiros antropólogos têm interesse é o de encontrar somente os *k-anéis que reencadeiam casamentos* em sociedades onde os indivíduos são particionados em grupos $\mathcal{C}_1, \ldots, \mathcal{C}_l$. Um k-anel que reencadeia casamentos é um k-anel $(\mathcal{E}, \mathcal{P})$, $\mathcal{P} = (P_1^{\mathcal{R}} Q_1, \ldots, P_k^{\mathcal{R}} Q_k,)$ tal que todo vértice u em $P_i \cup Q_i$, para todo $i = 1, \ldots, k$, pertença a um grupo $\mathcal{C}_j$ para algum j em $\{1, \ldots, l\}$. Exemplos de k-anéis que reencadeiam casamentos podem ser vistos na Figuras 5.2 e 6.1.

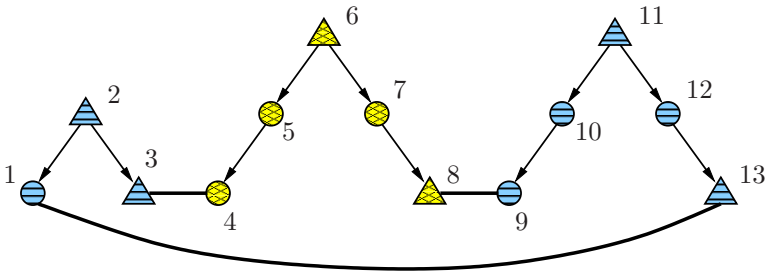


Figura 6.1: Um k-anel que reencadeia casamentos.

Referências Bibliográficas

[AC72] F.E. Allen e J. Cocke. Graph theoretic constructs for program control flow analysis. IBM Research Report RC 3923, Thomas J. Watson Research Center, Yorktown Heights, NY, 1972.

[AHLT99] Stephen Alstrup, Dov Harel, Peter W. Lauridsen e Mikkel Thorup. Dominators in linear time. *SIAM J. Comput.*, 28(6):2117–2132, 1999.

[AHU76] Alfred V. Aho, John E. Hopcroft e Jeffrey D. Ullman. On finding lowest common ancestors in trees. *SIAM J. Comput.*, 5(1):115–132, 1976.

[ALSU07] Alfred V. Aho, Monica S. Lam, Ravi Sethi e Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, segunda edição, 2007.

[AN96] Noga Alon e Moni Naor. Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16(4-5):434–449, 1996.

[AU72] Alfred V. Aho e Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling. Vol. I: Parsing.* Prentice-Hall Inc., Englewood Cliffs, N. J., 1972. Prentice-Hall Series in Automatic Computation.

[BEG+07] Matthias Baumgart, Stefan Eckhardt, Jan Griebsch, Sven Kosub e Johannes Nowak. All-pairs ancestor problems in weighted dags. In Bo Chen, Mike Paterson e Guochuan Zhang, editores, *ESCAPE*, volume 4614 de *Lecture Notes in Computer Science*, páginas 282–293. Springer, 2007.

[BFCP+05] Michael A. Bender, Martín Farach-Colton, Giridhar Pemmasani, Steven Skiena e Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *J. of Algor.*, 57(2):75–94, 2005.

[BJG08] Jørgen Bang-Jensen e Gregory Z. Gutin. *Digraphs: Theory, Algorithms and Applications.* Springer Publishing Company, Incorporated, segunda edição, 2008.

[BV93] Omer Berkman e Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM J. Comput.*, 22(2):221–242, 1993.

[BV94] Omer Berkman e Uzi Vishkin. Finding level-ancestors in trees. *J. of Comp. and Syst. Sci.*, 48(2):214–230, 1994.

[CGJ+07] Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein e Petra Mutzel. *Handbook of Graph Drawing and Visualization*, capítulo The Open Graph Drawing Framework (OGDF). Chapman & Hall/CRC, 2007.

[CH05] Richard Cole e Ramesh Hariharan. Dynamic LCA queries on trees. *SIAM J. Comput.*, 34(4):894–923, 2005.

[CKL07] Artur Czumaj, Mirosław Kowaluk e Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theor. Comput. Sci.*, 380(1-2):37–46, 2007.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest e Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, terceira edição, 2009.

[Dem12] Erik Demaine. Advanced data structures - aula 15. Notas: Jelle van den Hooff, Yuri Lin, Andrew Winslow, 2012.

[DF95a] Rod G Downey e Michael R Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

[DF95b] Rod G Downey e Michael R Fellows. Fixed-parameter tractability and completeness II: On completeness for $\mathcal{W}[1]$. *Theor. Comput. Sci.*, 141(1):109–131, 1995.

[dPdS09] João dal Poz e Marcio F. da Silva. Maqpar: a homemade tool for the study of kinship networks. *Vibrant*, 6(2):29–51, 2009.

[dPFdS13] João dal Poz, Carlos Eduardo Ferreira e Marcio Ferreira da Silva. Antropologia e computação: diálogos na selva, 2013. Seminário disponível em http://200.144.182.130/cesta/index.php/en/videos/216-cesta-tripla-joao-dal-poz-carlos-ferreira-marcio-silva.

[dS13] Marcio Ferreira da Silva. Comunicado pessoal, 2013.

[Dum75] Louis Dumont. *Introducción a dos Teorías de la Antropología Social*. Editorial Anagrama, 1975.

[EIS75] Shimon Even, Alon Itai e Adi Shamir. On the complexity of time table and multicommodity flow problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, SFCS '75, páginas 184–193, Washington, DC, USA, 1975. IEEE Computer Society.

[EMN07] Stefan Eckhardt, Andreas Michael Mühling e Johannes Nowak. Fast lowest common ancestor computations in dags. In Lars Arge, Michael Hoffmann e Emo Welzl, editores, *ESA*, volume 4698 de *Lecture Notes in Computer Science*, páginas 705–716. Springer, 2007.

[Epp95] David Eppstein. Finding common ancestors and disjoint paths in DAGs. Relatório Técnico 95-52, Univ. of California, Irvine, Dept. of Information and Computer Science, Irvine, CA, 92697-3425, USA, 1995.

[FF13] Carlos Eduardo Ferreira e Álvaro Junio Pereira Franco. Algorithms for junctions in acyclic digraphs. In *Facets of Combinatorial Optimization*, páginas 175–194. Springer, 2013.

[FG76] A. Frank e A. Gyárfás. Directed graphs and computer programs. *Problémes Combinatoires et Théorie des Graphes*, páginas 157–158, 1976.

[FG06] Jörg Flum e Martin Grohe. *Parameterized Complexity Theory*. Texts in theoretical computer science. Springer-Verlag Berlin and Heidelberg GmbH & Company KG, 2006.

[FHW80] Steven Fortune, John Hopcroft e James Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10(2):111–121, 1980.

[Fur70] M. E. Furman. Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph. *Sov. Math. Dokl*, 11(5):1252, 1970.

[FW94] Michael L. Fredman e Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. of Comp. and Syst. Sci.*, 48(3):533–551, 1994.

[FY87] Paulo Feofiloff e Daniel H. Younger. Directed cut transversal packing for source-sink connected graphs. *Combinatorica*, 7(3):255–263, 1987.

[Gab90] Harold N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In David S. Johnson, editor, *SODA*, páginas 434–443. SIAM, 1990.

[GT86] Andrew V. Goldberg e Robert E. Tarjan. A new approach to the maximum flow problem. In *Proceedings of the 8th annual ACM symposium on Theory of computing*, STOC '86, páginas 136–146, New York, NY, USA, 1986. ACM.

[GT04] Loukas Georgiadis e Robert E. Tarjan. Finding dominators revisited: extended abstract. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '04, páginas 869–878, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[GTW06] Loukas Georgiadis, Robert Endre Tarjan e Renato Fonseca F. Werneck. Finding dominators in practice. *J. of Graph Algor. and Applic.*, 10(1):69–94, 2006.

[HHD+04] Klaus Hamberger, Michael Houseman, Isabelle Daillant, Laurent Barry, Douglas R White et al. Matrimonial ring structures. *Mathematiques et sciences humaines*, (168):83–119, 2004.

[HT84] Dov Harel e Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[HU73] John E. Hopcroft e Jeffrey D. Ullman. Set merging algorithms. *SIAM J. Comput.*, 2(4):294–303, 1973.

[HU74] Matthew S. Hecht e Jeffrey D. Ullman. Characterizations of reducible flow graphs. *J. of ACM*, 21(3):367–375, Julho 1974.

[HW96] Michael Houseman e Douglas R White. Structures réticulaires de la pratique matrimoniale. *L'Homme*, 36(139):59–85, 1996.

[Knu71] Donald E. Knuth. An empirical study of fortran programs. *Soft.: Pract. and Exper.*, 1(2):105–133, 1971.

[Kőn31] Dénes Kőnig. Graphok és matrixok. *Mat. Fiz. Lapok*, 38:116–119, 1931. Em inglês: Graphs and Matrices.

[LG12] Francois Le Gall. Faster algorithms for rectangular matrix multiplication. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, páginas 514–523. IEEE, 2012.

[Lin09] Andrzej Lingas. A fast output-sensitive algorithm for boolean matrix multiplication. In *Algorithms-ESA 2009*, volume 5757 de *Lecture Notes in Computer Science*, páginas 408–419. Springer, 2009.

[LR80] Andrea S LaPaugh e Ronald L Rivest. The subgraph homeomorphism problem. *J. of Comp. and Syst. Sci.*, 20(2):133–149, 1980.

[LR83] Grazia Lotti e Francesco Romani. On the asymptotic complexity of rectangular matrix multiplication. *Theor. Comput. Sci.*, 23(2):171–185, 1983.

[LS69] Claude Lévi-Strauss. *Elementary Structures of Kinship*, volume 340. Beacon Press, 1969.

[LT79] Thomas Lengauer e Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. on Prog. Lang. and Syst.*, 1(1):121–141, Janeiro 1979.

[LY78] C.L. Lucchesi e D.H. Younger. A minimax theorem for directed graphs. *J. London Math. Soc.(2)*, 17(3):369–374, 1978.

[Mun71] Ian Munro. Efficient determination of the transitive closure of a directed graph. *Inf. Process. Lett.*, 1(2):56 – 58, 1971.

[NU94] Matti Nykänen e Esko Ukkonen. Finding lowest common ancestors in arbitrarily directed trees. *Inf. Process. Lett.*, 50(6):307–310, 1994.

[PS78] Yehoshua Perl e Yossi Shiloach. Finding two disjoint paths between two pairs of vertices in a graph. *J. ACM*, 25(1):1–9, 1978.

[Rad33] R Rado. Bemerkungen zur kombinatorik im anschluss an untersuchungen von herrn d. *Kőnig. Sitzungsber. Berliner Math. Ges*, 32:60–75, 1933.

[Ram88] Vijaya Ramachandran. Finding a minimum feedback arc set in reducible flow graphs. *J. of Algor.*, 9(3):299–313, 1988.

[Ram90] Vijaya Ramachandran. A minimax arc theorem for reducible flow graphs. *SIAM J. on Disc. Math.*, 3(4):554–560, 1990.

[RR94] G. Ramalingam e Thomas Reps. An incremental algorithm for maintaining the dominator tree of a reducible flowgraph. In *Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '94, páginas 287–296, New York, NY, USA, 1994. ACM.

[Sch82] Alexander Schrijver. Min-max relations for directed graphs. *Annals of Discrete Mathematics*, 16:261–280, 1982.

[Sch03] Alexander Schrijver. *Combinatorial Optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[Sed02] Robert Sedgewick. *Algorithms in C - part 5: graph algorithms.* Addison-Wesley-Longman, terceira edição, 2002.

[Sip96] Michael Sipser. *Introduction to the Theory of Computation.* International Thomson Publishing, primeira edição, 1996.

[Sli03] Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. In *Algorithms-ESA 2003*, páginas 482–493. Springer, 2003.

[SS98] Claus-Peter Schnorr e CR Subramanian. Almost optimal (on the average) combinatorial algorithms for boolean matrix product witnesses, computing the diameter. In *Randomization and Approximation Techniques in Computer Science*, páginas 218–231. Springer, 1998.

[ST84] J. W. Suurballe e Robert E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984.

[SW12] James Stanier e Des Watson. A study of irreducibility in c programs. *Soft.: Pract. and Exper.*, 42(1):117–130, 2012.

[Tho05] Torsten Tholey. Finding disjoint paths on directed acyclic graphs. In Dieter Kratsch, editor, *WG*, volume 3787 de *Lecture Notes in Computer Science*, páginas 319–330. Springer, 2005.

[Tho12] Torsten Tholey. Linear time algorithms for two disjoint paths problems on directed acyclic graphs. *Theor. Comput. Sci.*, 2012.

[VJZ70] Yvonne Verdier, Tina Jolas e Françoise Zonabend. Parler famille. *L'homme*, 10(3):5–26, 1970.

[Vyg95] Jens Vygen. $\mathcal{NP}$-completeness of some edge-disjoint paths problems. *Disc. Applied Math.*, 61(1):83–90, 1995.

[Wen94] Zhaofang Wen. New algorithms for the LCA problem and the binary tree reconstruction problem. *Inf. Process. Lett.*, 51(1):11–16, 1994.

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, páginas 887–898, New York, NY, USA, 2012. ACM.

[Woo78] D.R. Woodall. Menger and Kőnig systems. In *Theory and Applications of Graphs*, volume 642 de *Lecture Notes in Mathematics*, páginas 620–635. Springer, 1978.

[Yus08] Raphael Yuster. All-pairs disjoint paths from a common ancestor in $\tilde{O}(n^{\omega})$ time. *Theor. Comput. Sci.*, 396(1-3):145–150, 2008.

[Zwi98] Uri Zwick. All pairs shortest paths in weighted directed graphs-exact and almost exact algorithms. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, páginas 310–319. IEEE, 1998.

Anexos

A

Pseudocódigos

Neste capítulo descrevemos os pseudocódigos dos principais algoritmos desta tese. Os vetores globais usados em todos os pseudocódigos estão listados em seguida.

r: Armazena vértices; Indexado por vértices. $r(v)$ é o representante da partição em estrelas que pertence v.

r': Armazena valores booleanos; Indexado por vértices. $r'(v) = 1$ se e somente se existem caminhos internamente vértice-disjuntos de s para v.

p: Armazena vértices; Indexado por vértices. $p(v)$ é o pai de v em uma árvore geradora de busca em profundidade.

pós: Armazena valores inteiros em $\{0, 1, \ldots, n-1\}$; Indexado por vértices. $pós(v) = i$ se i é o valor de v em uma varredura pós-ordem.

sóp: Armazena vértices; Indexado por valores em $\{0, 1, \ldots, n-1\}$. $sóp(i) = v$ se v é o vértice correspondente ao valor i em uma varredura pós-ordem.

minpós: Armazena vértices; Indexado por vértices. $minpós(v) = u$ se u é o vértice com menor valor pós-ordem na árvore T^v.

A.1 Pseudocódigo para o Problema s-junção-k-pares

Entrada: um digrafo acíclico D com n vértices e m arcos, e um vértice s de D.
Saída: uma partição em estrelas de D e s (veja o Capítulo 4).

SJKP(D, s)

```
T^s ← AGBP(D, s)
para cada vértice u ∈ T^s
      faça r'(u) ← 0
            r(u) ← p(u)
r'(s) ← 1
r(s) ← s
para cada vértice s_i ∈ δ^+_{T^s}(s) em pós-ordem decrescente
      faça se ∃ t → s_i ∉ T^s
                então r'(s_i) ← 1
            r(s_i) ← s_i
            SJKP-Rec(s_i)
```

SJKP-Rec(z)

```
1  se pós(z) ≠ minpós(z)    ▷ Se |T^z| > 1
2      então w ← sóp(pós(z) − 1)
3            enquanto pós(w) ≥ minpós(z)
4                faça se ∃ t → w ∉ T^s e r(z) ≠ r(t)
5                        então r′(w) ← 1, r(w) ← w
6                              SJKP-Rec(w), w ← sóp(minpós(w) − 1)
7                        senão r(w) ← z, w ← sóp(pós(w) − 1)
```

A.2 Pseudocódigo para o Problema s-arco-junção-k-pares

Entrada: um digrafo acíclico D com n vértices e m arcos, e um vértice s de D.
Saída: estruturas de dados r e r' (veja o Capítulo 4).

SAJKP(D, s)

```
1  T^s ← AGBP(D, s)
2  para cada vértice u ∈ T^s
3       faça r′(u) ← 0, r(u) ← p(u)
4  r′(s) ← 1, r(s) ← s
5  SAJKP-Rec(s)
```

SAJKP-Rec(z)

```
1  para cada vértice z_i ∈ δ^+_{T^s}(z) em pós-ordem decrescente
2       faça r(z_i) ← z_i
3            se ∃ t → z_i ∉ T^s
4               então r′(z_i) ← 1
5                     SAJKP-Rec(z_i)
6               senão SAJKP-Aux(z_i)
```

SAJKP-Aux(z)

```
1  se pós(z) ≠ minpós(z) ▷ Se |T^z| > 1
2      então w ← sóp(pós(z) − 1)
3            enquanto pós(w) ≥ minpós(z)
4                faça se ∃ t → w ∉ T^s e (r′(t) = 1 ou r(z) ≠ r(t))
5                        então r(w) ← w, r′(w) ← 1
6                              SAJKP-Rec(w)
7                              w ← sóp(minpós(w) − 1)
8                        senão r(w) ← z, w ← sóp(pós(w) − 1)
```

A.3 Pseudocódigo para o Problema árvore-de-dominadores

$\mathcal{B}^s$ é a partição em estrelas de D e s, e $w_1, \ldots, w_{\beta^\star}$ são os representantes em $\mathcal{B}^s$.
Entrada: um grafo fluxo redutível D com n vértices e m arcos, e a sua raiz s.
Saída: uma árvore de dominadores T de D.

ÁrvoreDominadores(D, s)

```
1  se D tem mais que 1 vértice
2      então B^s ← SJKP(D, s) ▷ Recebe as estruturas de dados r e r′.
3            para i = 1, ..., β^⋆
4                faça ÁrvoreDominadores(D[B^s_i], w_i)
5                     p_T(w_i) ← s ▷ p_T(w_i) é o pai de w_i em T
```

B

Desenho de algumas sociedades

Na Figura B.1, apresentamos um bom desenho da sociedade Arara. Ele foi feito com o *Open Graph Drawing Framework* (OGDF) com otimização (CPLEX 12.1.0) [CGJ+07].

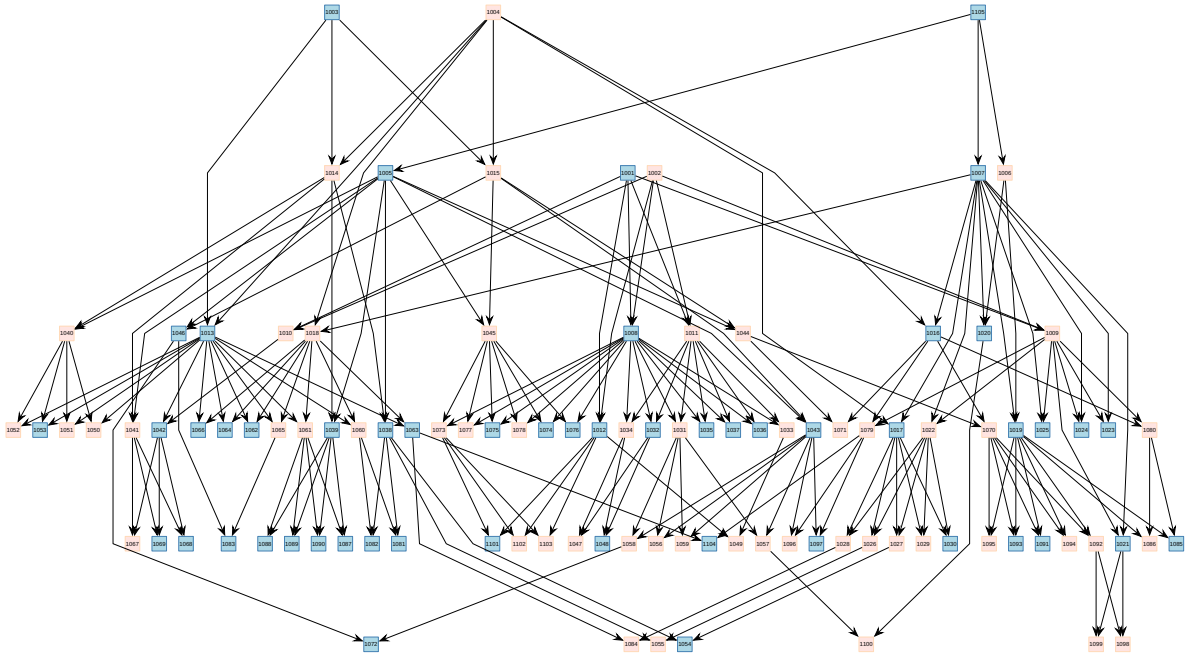


Figura B.1: Sociedade Arara. Note uma junção 1005 (segundo vértice no segundo nível) para o casamento envolvendo os indivíduos 1046 (segundo vértice no terceiro nível) e 1058 (no meio do quinto nível).

Na Figura B.2, apresentamos o melhor desenho da sociedade Xavante feito pelo *Open Graph Drawing Framework* (OGDF) com otimização (CPLEX 12.1.0) [CGJ+07]. Não conseguimos desenhos apresentáveis para as sociedades Enawenê-Nawês, Deni, Arapium, Irantxe-Myky e Zoró devido ao grande número de vértices e de arcos.

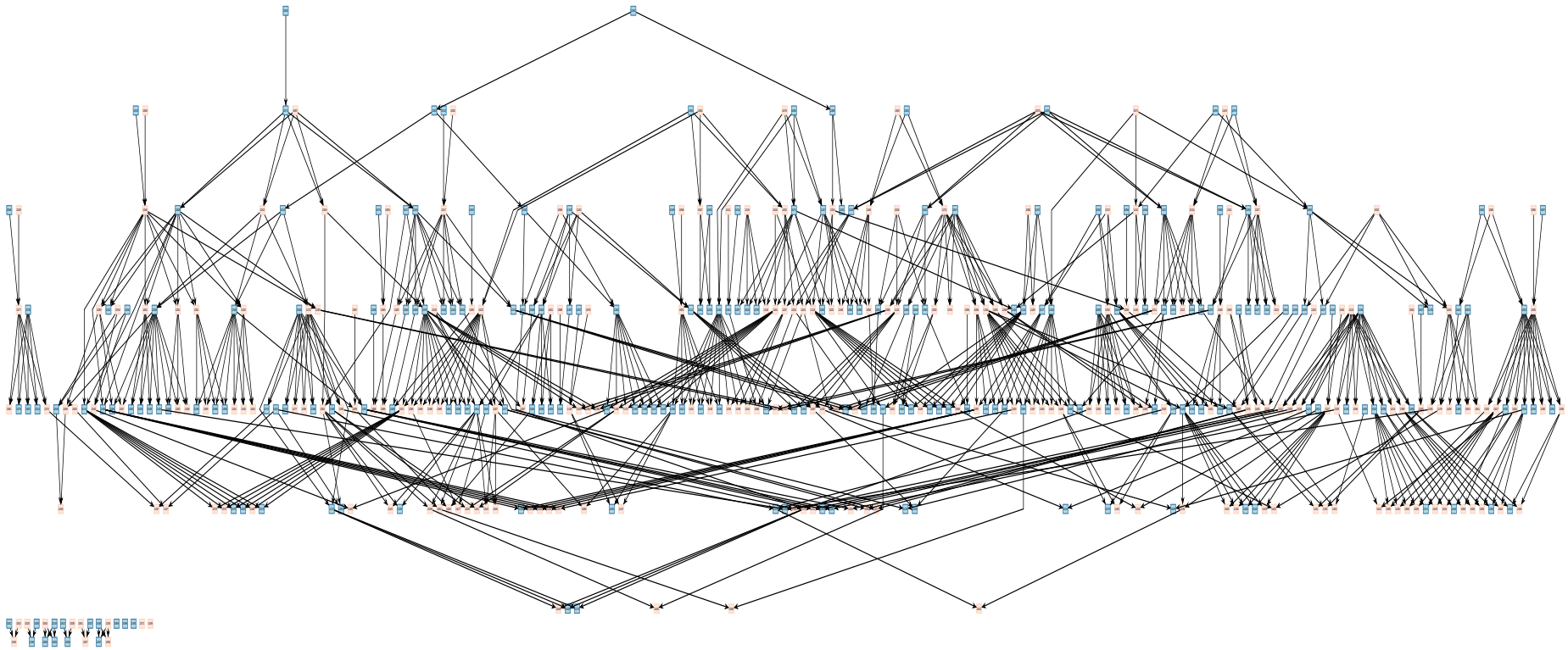


Figura B.2: Sociedade Xavante.

Índice Remissivo

árvore, 5
- de dominadores, 8
- enraizada, 6
- geradora de busca em profundidade, 6

loops, 5

abreviações
- acp, 9
- agbp, 6, 8
- gfr, 8

altura, 6
ancestral comum mais próximo, 9
anel, 10
- k-anel, 10
  - reencadeia casamentos, 82
- degenerado, 11

arco, 5
- cruzado, 7
- da árvore, 6
- de avanço, 6
- de retorno, 7, 8
- invertido, 5
- paralelo, 5

arco-junção, 9
aresta, 5

busca em profundidade, 6

caminho, 5
- arco-disjunto, 9
- concatenação, 5
- dirigido, 5
- subcaminho, 5
- vértice-disjunto, 9, 13

ciclo, 5
circuito booleano, 14
- k-satisfatível, 14
- vértice de entrada, 14
- vértice de saída, 14
- vértice irrestrito e, 14
- vértice e, 14
- vértice *não*, 14
- vértice *ou*, 14

classes, 12
- $\mathcal{FPT}$, 13
- $\mathcal{NP}$, 12
- $\mathcal{P}$, 12
- $\mathcal{W}[1]$, 13
- $\mathcal{W}[2]$, 14
- $\mathcal{W}[3]$, 14
- $\mathcal{W}[P]$, 14

cobertura por arestas, 53
cobertura por vértices, 53
colegas em casamentos, 71
complexidade
- computacional
  - clássica, 12
- parametrizada, 12

conjunto
- de arcos
  - dirigido, 53

consulta, 12
- s-acp(u, v), 12
- s-arco-junção(u, v), 12
- s-junção(u, v), 12
- acp(u, v), 12
- representante-acp(u, v), 12
- representante-junção(u, v), 12
- todas-junções(u, v), 12
- todos-acps(u, v), 12

corte dirigido, 53

digrafo, 5
- acíclico, 5
- fonte-sorvedouro-conexo, 53
- linha, 6

digrafo acíclico
- profundidade em, 23

dijunção, 53
distância, 23
- indireta, 24

dominador
- mais alto, não trivial55
  - não trivial, 70

trivial, 54

emparelhamento, 52
entrada válida, 13
estrutura de dados, 11
expoente da multiplicação de matrizes retangulares, 9

fecho transitivo, 9
filho, 6
folha, 6

grafo, 5
bipartido, 52
conexo, 5
correspondente, 5
fluxo, 7
misto, 6
acíclico, 6, 63, 68
grafo fluxo
redutível, 8

incide, 5
independente, 53
induzido, 6

junção, 9
degenerada, 9, 11

máquina
de Turing
determinística, 12
não-determinística, 12
método
linha-coluna, 51
matriz
booleana, 9
de adjacência, 9
de testemunha, 9

ordem topológica
crescente, 6, 23, 35, 49
decrescente, 6, 49
ordenação topológica, 6
otimização do código, 34

p-redução, 13, 14, 64
pai, 6
partição em estrelas, 47, 70
pré-processamento, 11
problema
$(s-t)$-caminho, 66
k-caminhos-vértice-disjuntos, 13, 15, 62, 66
circuito-k-satisfatível, 14
subdigrafo-homeomorfo-k-anel, 65
subdigrafo-homeomorfo, 65
todos-k-anéis, 59
um-k-anel, 61
s-acp-todos-pares, 27
s-arco-junção-k-pares, 50
s-junção-k-pares, 37
s-junção-todos-pares, 31
árvore-de-dominadores, 34
acp-todos-pares, 17
consulta-janela, 12
máxima-testemunha-todos-pares, 24
menor-distância-todos-pares, 23
representante-acp-k-pares, 26
representante-acp-todos-pares, 22
representante-junção-k-pares, 31
representante-junção-todos-pares, 28
rmq-todos-pares, 20
todas-junções-k-pares, 33
todas-junções-todos-pares, 31
todos-acps-k-pares, 28
todos-acps-todos-pares, 26
parametrizado, 13
profundidade de um circuito, 14

raiz, 6, 7
redobra, 60
reencadeia, 60
regra de troca, 60
relação min-max, 54
relação de dominância, 8

símbolos
$D[S]$, 6
D_l, 6
D_{ft}, 9
$M(i,j)$, 9
α, 54
$\alpha^\star$, 54
β, 54
$\beta^\star$, 54
$\delta_D^+(u)$, 6
$\delta_D^-(u)$, 6
$\mathcal{ACP}_{u,v}$, 10
$\mathcal{AJ}_{u,v}$, 9
$\mathcal{B}^s$, 54
$\mathcal{B}_i^s$, 54
$\mathcal{J}_{u,v}$, 9
ω, 9
$\omega(1,1,k)$, 9
$g_D^+(u)$, 6
$g_D^-(u)$, 6

$x - y$, 5
$x \to y$, 5
sociedade
Arapium, 70
Arara, 70
Deni, 70
Enawenê-Nawês, 70
Irantxe-Myky, 70
Xavante, 70
Zoró, 70
subgrafo acíclico maximal, 8

teia de um circuito, 14

vértice, 5
adjacente, 5
ancestral, 9
próprio, 9
descendente, 9
próprio, 9
domina, 7
dominador imediato, 7
fonte, 6, 53
origem comum, 9
ponta, 5
ponta final, 5
ponta inicial, 5
sorvedouro, 6, 53