GEOMETRIA COMPUTACIONAL DE PONTOS EM MOVIMENTO

Carlos Ramon Pantaleón Dionisio

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO GRAU DE MESTRE
EM
CIÊNCIA DA COMPUTAÇÃO

Orientador: **Prof. José Coelho de Pina**

Durante a elaboração deste trabalho o autor recebeu auxílio financeiro do CNPq.

São Paulo, 28 de fevereiro de 2000

Geometria Computacional de Pontos em Movimento

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e defendida por Carlos Ramon Pantaleón Dionisio
e aprovada pela comissão julgadora.

São Paulo, 28 de fevereiro de 2000.

Banca examinadora:

- Prof. Dr. José Coelho de Pina (orientador) - IME-USP
- Prof. Dr. Carlos Eduardo Ferreira - IME-USP
- Prof. Dr. Jorge Stolfi- IC-UNICAMP

A minha familia,
pelo apoio durante todos estes anos.

Agradecimentos

Ao Prof. José Coelho, pela paciência, amizade e excelente orientação demostrados durante a realização deste trabalho.

Aos professores do Departamento de Matemática da Universidade Nacional de Ingenieria - Perú, por terem contribuído para a minha formação acadêmica.

Aos meus amigos, pelo apoio durante a realização deste trabalho.

Aos professores e funcionários do IME-USP, pela formação e serviços prestados.

Finalmente, a todos que de alguma forma contribuíram a realização deste trabalho.

Resumo

Nesta dissertação apresentamos uma visão geral de técnicas, algoritmos e estruturas de dados para a solução de problemas geométricos envolvendo pontos que estão se movendo continuamente no plano. Tais problemas geométricos podem ser vistos como abstração de problemas em áreas como controle de tráfego aéreo, robótica, telefonia celular, computação gráfica, etc.

Descreveremos os três modelos para problemas de pontos em movimento que encontramos na literatura, a saber, o modelo off-line de Atallah [9, 10] e Ottman e Wood [34], o modelo de tempo-real de Kahan [28, 29], e modelo cinético de Basch, Guibas e Hershberger [13, 14].

***

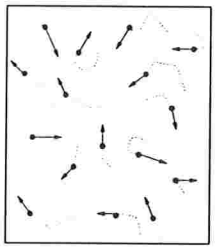
In this monograph we survey known techniques, algorithms and data structures for geometric problems concerning points moving continuously on the plane. These problems can be seen as an abstraction of problems in air traffic control, collision detection in robotics and animation, switching cellular phone transceiver stations amongst moving automobiles, visibility determination in computer computer graphics, etc.

We describe the three models for data in motion problems we have found in the literature, namely, the off-line model due to Atallah [9, 10] and Ottman, and Wood [34], the real-time model due to Kahan [28, 29], and the kinetic model due to Basch, Guibas e Hershberger [13, 14].

Índice

CAPÍTULO 1

Introdução



Problemas em geometria computacional têm como instância um conjunto de objetos geométricos que comumente representam entidades do mundo físico. Por exemplo, pontos podem representar agências do correio ou aviões. Quando este conjunto de objetos e seus atributos são conhecidos de antemão temos um problema que é dito *off-line* (veja, por exemplo, a página 118 de [36]). Problemas *dinâmicos* ou *incrementais* requerem que uma solução seja rapidamente atualizada enquanto a instância do problema é modificada através de algumas operações — tipicamente remoções e inserções de objetos. Problemas dinâmicos são também chamados de *on-line*.

Uma instância, para um problema on-line, consiste em uma seqüência de operações que não são conhecidas de antemão e que devem ser realizadas uma após a outra. Entre as operações encontramos *consultas* (*queries*) que são simplesmente pedidos de informação sobre o valor de alguma função definida sobre o conjunto de objetos considerado. Como exemplo, consideremos o problema de se manter o valor da menor distância entre os pares de pontos de um conjunto que está sendo alterado ao longo do tempo através de remoções e inserções de pontos. Neste exemplo consultas seriam perguntas do tipo: "Qual é o valor da menor distância entre dois pontos do conjunto?". Uma visão geral sobre algoritmos dinâmicos em geometria computacional pode ser encontrada em [19].

Nesta dissertação apresentamos uma visão geral de técnicas, algoritmos e estruturas de dados para a solução de problemas geométricos envolvendo pontos que estão se movendo continuamente no plano. Tais problemas geométricos podem ser vistos como abstração de problemas em áreas como controle de tráfego aéreo, telefonia celular e computação gráfica. Assim, a instância para os problemas que veremos são pontos em movimento e para cada um destes o único atributo no qual estaremos interessados é a sua posição. Neste contexto imaginaremos os pontos como sendo representações de aviões e freqüentemente nos referiremos ao movimento de cada ponto como sendo o seu *plano de vôo*; da mesma

maneira que é feito em [13, 14, 16, 17].

Descreveremos três modelos para problemas envolvendo pontos em movimento, a saber: modelo *off-line* de Atallah [9, 10] e Ottman e Wood [34]; modelo *de tempo-real* de Kahan [28, 29]; e modelo *cinético* de Basch, Guibas e Hershberger [13, 14]. Pretendemos apresentar uma visão geral sobre as técnicas, algoritmos e estruturas de dados usadas para abordar estes três modelos. Estes modelos foram os únicos que encontramos na literatura.

No início da década de oitenta, Atallah [9, 10] e Ottman e Wood [34] consideraram problemas de pontos em movimento onde cada ponto se move de uma maneira predeterminada. Logo, os problemas tratados por Atallah, Ottman e Wood são off-line — Atallah chamou estes problemas de dinâmicos e reservou o adjetivo *estático* para denotar problemas onde os pontos mantêm posições fixas; usaremos o adjetivo estático neste mesmo sentido.

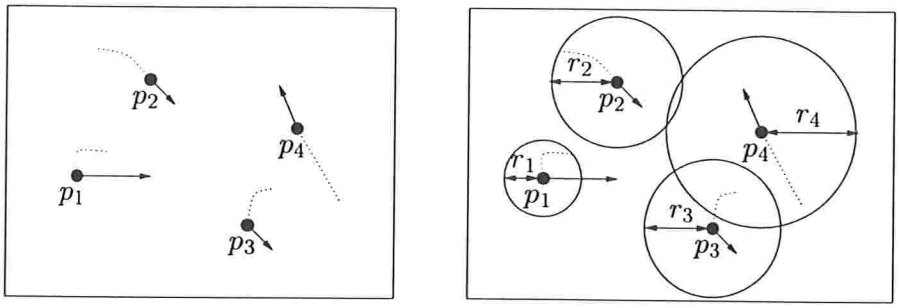
Na situação estudada por Atallah em [9, 10] temos um conjunto de n pontos no espaço euclidiano d-dimensional onde a coordenada de cada ponto é determinada por um polinômio na variável 'tempo' t; ou seja, a posição de cada ponto é dada através de uma equação da forma $p(t) = c_0 + c_1 t + \cdots + c_k t^k$, onde $c_i \in \mathbb{R}^d$. Logo, c_0 indica a posição do ponto no instante $t = 0$. Chamamos a equação de movimento de um ponto (i.e., o plano de vôo do ponto) de um *k-movimento* se o grau do polinômio que descreve o seu movimento é menor ou igual a k.

Atallah obteve limites superiores para o número de mudanças nas 'descrições combinatórias' do par de pontos mais-próximo e do fecho convexo de um conjunto de pontos no plano. Por uma *descrição combinatória* para um objeto geométrico entendemos uma estrutura de dados que o representa; por exemplo, um par de pontos mais-próximo pode ser representado simplesmente por um par, enquanto o fecho convexo de um conjunto de pontos no plano pode ser representado pela lista circular do polígono determinado pelo fecho.

Limites superiores, bem como limites inferiores, para o número de mudanças nas descrições combinatórias de outros objetos geométricos determinados por um conjunto de pontos (e.g., árvore geradora mínima, diagrama de Voronoi, par mais-distante) que realizam um 1-movimento têm sido estabelecidos (cf. [2, 25, 30]).

Monitorar objetos em movimento é um componente importante de sistemas de tempo-real, como, por exemplo, em sistemas de controle de tráfego aéreo (cf. [18]). Diremos que um problema é de *tempo-real* se este é um problema on-line onde a resposta a cada consulta deve ser dada em tempo que não exceda um certo prazo limite (veja, por exemplo, a página 118 de Preparata e Shamos [36]). Se a instância do problema tem um número tão grande de objetos então é impraticável que o algoritmo dê uma resposta correta a cada consulta no tempo estipulado.

Kahan [28, 29] tratou de problemas de tempo-real envolvendo pontos em movimento. Nos problemas considerados um grande número de pontos se movem continuamente e a cada instante a posição de um dado ponto pode ser obtida através de algum dispositivo ou sensor, como, por exemplo, um radar. A única informação conhecida de antemão sobre o movimento de cada ponto p_i é um limite superior r_i para a sua velocidade ($i = 1, \ldots, n$). Assim, se no instante t^* o ponto p_i está na posição $p_i(t^*) \in \mathbb{R}^2$, então no instante $t^* + 1$ o ponto p_i tem a sua posição dentro do círculo de centro $p_i(t^*)$ e raio r_i ($i = 1, \ldots, n$) (veja Figura 1.1). Neste tipo de problema deseja-se obter 'rapidamente' as respostas a uma seqüência de consultas que não são conhecidas de antemão, ou seja, as consultas são recebidas on-line.



instante t^* instante $t^* + 1$

Figura 1.1: A informação conhecida sobre o movimento de cada ponto p_i é apenas um limite r_i para a sua velocidade ($i = 1, 2, 3, 4$).

Mais recentemente, Basch, Guibas e Hershberger [13, 14] estudaram problemas de se manter continuamente a descrição combinatória de um certo objeto geométrico de interesse (e.g., fecho convexo) determinado por um conjunto de pontos em movimento contínuo. Nos problemas tratados é assumido que cada ponto tem um plano de vôo, que pode vir a ser alterado. Desta forma, diferentemente do modelo de problema tratado por Atallah [9, 10], o plano de vôo completo de cada ponto não precisa ser conhecido de antemão, cada ponto pode ter o seu movimento alterado a qualquer momento devido a fatores externos, como colisão entre pontos. O plano de vôo pode ser uma equação descrevendo o movimento do ponto.

Apesar dos pontos se moverem continuamente e, portanto, o objeto geométrico considerado também estar se movendo continuamente, a descrição combinatória deste objeto só muda em certos momentos críticos discretos. A técnica utilizada por Basch, Guibas e Hershberger [13, 14] para se manter a descrição combinatória de um objeto geométrico em movimento contínuo se concentra exatamente nestes momentos críticos. Os três autores

propuseram uma estrutura de dados, denominada *cinética*[1], que além da descrição combinatória do objeto também mantém *certificados* (i.e., predicados geométricos) que formam uma prova da corretude da descrição combinatória que está sendo mantida.

Estes predicados devem envolver um número constante de parâmetros. Por exemplo, uma prova para a corretude da descrição combinatória da Triangularização de Delaunay pode ser dada através de certificados do tipo `InCircle`(p_1, p_2, p_3, p_4), que dados quatro pontos p_1, p_2, p_3, p_4 é verdadeiro se "o ponto p_4 está no interior do círculo determinado pelos pontos p_1, p_2 e p_3" (cf. [27], veja também a Propriedade do Círculo Circunscrito em Resende e Stolfi [21] página 185). Muitas vezes os certificados são obtidos de algoritmos que resolvem o correspondente problema estático, isto é, problemas onde os pontos estão fixos. Como exemplo, consideremos o problema estático de construir o fecho convexo de um conjunto de pontos no plano, que tem sido extensivamente estudado em geometria computacional (veja, por exemplo, o Capítulo 3 de O'Rourke [33] e o Capítulo 3 de Preparata e Shamos [36]). Um certificado muito utilizado pelos algoritmos estáticos para construir o fecho convexo é `Left`(p_1, p_2, p_3) que é verdadeiro se "o ponto p_3 está à esquerda do segmento orientado p_1p_2". Em [13, 14] certificados do tipo `Left`(p_1, p_2, p_3) fazem parte da estrutura de dados cinética desenvolvida para manter o fecho convexo de um conjunto de pontos em movimento. A Figura 1.2 ilustra os dois tipos de certificados discutidos neste parágrafo. Observemos que, na figura, quando um certificado passa de verdadeiro para falso ou vice-versa, temos uma alteração na descrição combinatória do objeto geométrico considerado.

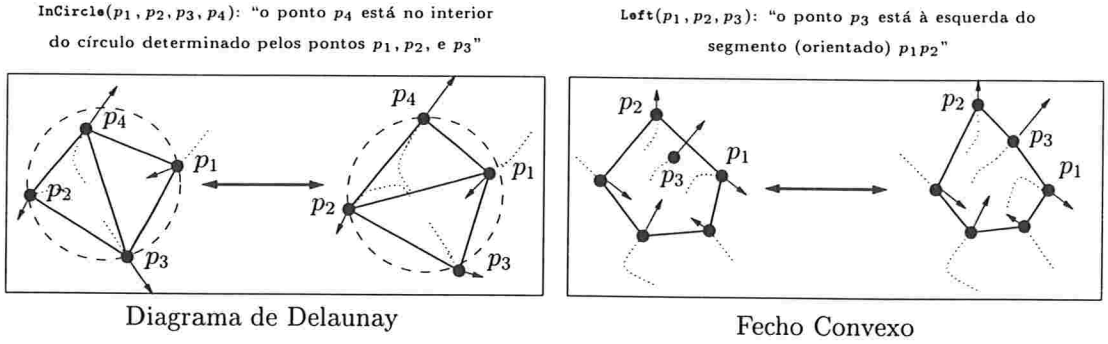


Figura 1.2: Ilustração dos certificados `InCircle` e `Left`.

Têm sido propostas estruturas de dados cinéticas para manter a descrição combinatória de vários objetos geométricos: fecho convexo e par mais-próximo no plano [13, 14], *binary space partition* [1, 3]; par mais-próximo e árvore geradora mínima em dimensão arbitrária [17]; par mais-distante (diâmetro) [2]; detecção de colisão entre polígonos [12, 23].

Os três modelos discutidos brevemente nesta introdução têm as suas semelhanças, por

[1] Veja a Tese de Ph.D. de Basch [11]

exemplo, em todos temos alguma informação sobre o movimento de cada ponto. Por outro lado, a principal diferença entre eles é o *tipo* de informação disponível sobre este movimento:

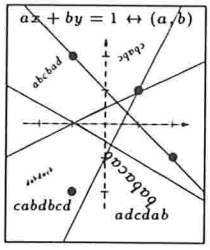
- nos problemas considerados por Atallah o plano de vôo de cada ponto é um polinômio totalmente conhecido de antemão;
- Kahan considera que a única informação conhecida sobre o movimento de cada ponto é um limite para sua velocidade; e
- para Basch, Guibas e Hershberger, cada ponto tem o seu plano de vôo, que pode ser alterado on-line.

Ao longo desta dissertação chamaremos de *estáticos* os problemas onde os objetos mantêm posições fixas ao longo do tempo. Os adjetivos *off-line*, *tempo-real* e *cinético* serão reservados para os tipos de problemas tratados por Atallah [9, 10], Kahan [28, 29], e Basch, Guibas e Hershberger [13, 14], respectivamente. Ademais, chamaremos um algoritmo de estático, off-line, de tempo-real ou cinético se o problema que ele resolve é estático, off-line, de tempo-real ou cinético, respectivamente.

Esta dissertação está organizado como segue. No Capítulo 2 apresentamos conceitos básicos que serão necessários em algumas dos capítulos subseqüentes. O Capítulo 3 descreve os três modelos para problemas de pontos em movimentos discutidos nesta introdução: modelo off-line; modelo de tempo-real; e modelo cinético. Neste mesmo capítulo também descrevemos as medidas de complexidade que estaremos considerando em cada um dos diferentes modelos. Algoritmos para o problema de se determinar o ponto de maior ordenada entre um conjunto de pontos em movimento contínuo ao longo do eixo das ordenadas são apresentados no Capítulo 4, o chamado problema do máximo que, apesar de básico, serve para ilustrarmos os algoritmos e técnicas nos três modelos. Acreditamos que este Capítulo 4 é capaz de dar uma idéia razoável de algumas técnicas que são usadas para se solucionar problemas de pontos em movimento nos diferentes modelos considerados. No Capítulo 5 tratamos do problema de se determinar um par de pontos mais-próximo. Finalmente, os problemas off-line e cinético de se computar o fecho convexo de um conjuntos de pontos em movimento são tratados no Capítulo 6.

CAPÍTULO 2

Conceitos básicos



Descreveremos neste capítulo alguns dos conceitos básicos que serão necessários ao longo desta dissertação. Mais especificamente, discutiremos um pouco sobre seqüências de Davenport-Schinzel [20], envelopes e dualidade. As seqüências de Davenport-Schinzel, introduzidas por H. Davenport e A. Schinzel no ano 1960, são estruturas combinatórias sobre palavras que se apresentam na análise da complexidade combinatória de envelopes e, portanto, na análise da eficiência de algoritmos para construí-los.

O dual de um conjunto de pontos no plano é um conjunto de retas e vice-versa. Dualidade nos permite transformar um problema dado em um outro problema onde, algumas vezes, as propriedades do problema original podem ser mais facilmente percebidas, o que pode facilitar a sua solução. Ao resolver um problema dualizado estaremos também resolvendo o problema original, com a mesma complexidade de tempo, já que existe uma correspondência bijetiva entre ambas soluções.

Alguns problemas geométricos no plano podem ser transformados no problema de construir-se o envelope superior ou inferior de um conjunto de retas. Um exemplo de tal tipo de problema é o *Problema do Fecho Convexo no Plano*: *dado*: um conjunto S de pontos no plano; *encontrar*: o fecho convexo de S. As cadeias inferior e superior do fecho convexo de S podem ser obtidas, através de dualidade, a partir do envelope superior e inferior do conjunto de retas duais de S.

Ao longo desta dissertação procuramos utilizar os termos de geometria computacional como são apresentados em livros conhecidos tais como: Figueiredo e Carvalho [24]; Mulmuley [32]; O'Rourke [33]; Preparata e Shamos [36]; e Resende e Stolfi [21]. Em particular, os tópicos tratados neste capítulo podem ser encontrados em Agarwal e Sharir [5], Atallah [9, 10], Davenport e Schinzel [20], Guibas e Shamir [26] e Capítulo 6 de O'Rourke [33].

2.1 Seqüências de Davenport-Schinzel

Seja Σ um alfabeto com n símbolos. Uma seqüência $\sigma = \sigma_1\sigma_2...\sigma_m$ sobre o alfabeto Σ é uma *seqüência de Davenport-Schinzel* de ordem s (*$DS(n,s)$-seqüência* ou simplesmente *(n,s)-seqüência*) se satisfaz as seguintes condições:

- Quaisquer dois símbolos consecutivos de σ devem ser distintos, ou seja, $\sigma_i \neq \sigma_{i+1}$ para $i = 1, \ldots, m-1$.
- Para cada par de símbolos distintos $a, b \in \Sigma$, qualquer subseqüência alternante de a's e b's em σ tem comprimento máximo $s+1$.

Desta forma, uma $(n,1)$-seqüência não contém *aba* como subseqüência, uma $(n,2)$-seqüência não contém *abab* como subseqüência, e assim por diante. Por exemplo, *abbcda* não é uma seqüência de Davenport-Schinzel para nenhum valor de s. A seqüência *acbcda* é uma $(4,2)$-seqüência, mas não é uma $(4,1)$-seqüência. A seqüência *adbcdacb* é uma $(4,3)$-seqüência, mas não é uma $(4,2)$-seqüência.

Para está dissertação será particularmente de interesse o valor $\lambda_s(n)$ do maior comprimento de uma (n,s)-seqüência, ou seja,

$$\lambda_s(n) = \max\{|\sigma| : \sigma \text{ é uma } (n,s)\text{-seqüência}\}.$$

Davenport e Schinzel [20] mostraram que $\lambda_1(n) = n$ e $\lambda_2(n) = 2n-1$. Encontrar o valor exato de $\lambda_s(n)$ para $s > 2$ é um problema matematicamente difícil. Ainda pode-se provar facilmente (por indução em n) que $\lambda_3(n) = O(n \log n)$. Szemerédi [42] provou que para s fixo $\lambda_s(n) = O(n \log^* n)$.

Teorema 2.1 *Seja Σ um alfabeto com n símbolos, então:*

(*a*) $\lambda_1(n) = n$

(*b*) $\lambda_2(n) = 2n-1$

(*c*) $\lambda_3(n) = O(n \log n)$

Prova. Por definição, qualquer $(n,1)$-seqüência não contém *aba* como subseqüência, ou seja todos os símbolos de uma $(n,1)$-seqüência são distintos. Isto prova (*a*).

A seqüência $\sigma_1\sigma_2 \ldots \sigma_{n-1}\sigma_n\sigma_{n-1} \ldots \sigma_2\sigma_1$, com $\sigma_i \neq \sigma_j$ para $i \neq j$ é uma $(n,2)$-seqüência de comprimento $2n-1$. Portanto $\lambda_2(n) \geq 2n-1$. Mostraremos, por indução no tamanho n do alfabeto, que $\lambda_2(n) \leq 2n-1$. Com isto provamos (*b*), i.e. $\lambda_2(n) = 2n-1$.

Para $n = 1$ temos trivialmente que o comprimento da maior $(1,2)$-seqüência é 1, ou seja, $\lambda_2(1) = 1 = 2n - 1$. Suponhamos agora que $n > 1$ e que $\lambda_2(n-1) \leq 2(n-1) - 1 = 2n - 3$. Seja σ uma $(n,2)$-seqüência sobre o alfabeto Σ. Mostraremos que existe um símbolo de Σ que ocorre uma única vez em σ. Se todo símbolo em σ ocorresse pelo menos duas vezes então existiria um símbolo a de σ e existiriam seqüências α, β, γ tais que:

- β é não-vazia;
- todo símbolo de Σ ocorre no máximo uma vez em β; e
- $\sigma = \alpha a \beta a \gamma$.

Seja b um símbolo qualquer de β. Como b ocorre pelo menos duas vezes em σ então b ocorre em α ou b ocorre em γ. No primeiro caso temos que $baba$ é subseqüência de σ e no segundo caso temos que $abab$ é subseqüência de σ. Em ambas as possibilidades chegamos a um fato que contradiz a hipótese de σ ser uma $(n,2)$-subseqüência. Logo, existe um símbolo (digamos) a de Σ que ocorre apenas uma vez em σ.

Removendo a de σ e se os símbolos adjacentes a a forem iguais então também removendo um deles obtemos uma seqüência σ' que é uma $(n-1,2)$-seqüência e, portanto, pela hipótese de indução, $|\sigma'| \leq \lambda_2(n-1) = 2n - 3$. Portanto

$$|\sigma| \leq |\sigma'| + 2 \leq 2n - 3 + 2 = 2n - 1.$$

Com isto completamos a demonstração de (b).

Provaremos agora (c). Seja σ uma $(n,3)$-seqüência sobre o alfabeto Σ. Para todo símbolo x em Σ seja $f(x)$ o número de ocorrências de x em σ. Seja a um símbolo de Σ de menor ocorrência em σ, isto é $m := f(a) \leq f(x)$ para todo símbolo x em σ. Temos que

$$m \leq \frac{\lambda_3(n)}{n}.$$

Sejam $\alpha_0, \alpha_1, \ldots, \alpha_m$ seqüências sobre Σ tais que

$$\sigma = \alpha_0 a_1 \alpha_1 a_2 \ldots \alpha_{m-1} a_m \alpha_m,$$

com $a_i = a$ para $i = 1 \ldots m$. Como σ é uma $(n,3)$-seqüência então não existe símbolo de Σ que seja ao mesmo tempo antecessor e sucessor de algum a, a menos que este a seja a_1 ou a_m. De fato, para $m = 1$ e $m = 2$ esta afirmação é trivial. Para $m \geq 3$ a afirmação segue do fato que σ é uma $(n,3)$-seqüência.

Removendo todas as ocorrências de a em σ e se os dois símbolos adjacentes a a_1 ou a a_m são iguais então removemos um deles, obtendo uma $(n-1,3)$-seqüência, digamos σ', tal que

$$|\sigma| \leq |\sigma'| + \frac{\lambda_3(n)}{n} + 2.$$

Assim

$$\lambda_3(n) \leq \lambda_3(n-1) + \frac{\lambda_3(n)}{n} + 2.$$

Reescrevendo a desigualdade acima obtemos a recorrência

$$\frac{\lambda_3(n)}{n} \leq \frac{\lambda_3(n-1)}{n-1} + \frac{2}{n-1}.$$

Finalmente, resolvendo a recorrência obtemos a cota superior

$$\lambda_3(n) \leq 2n(1 + \log n).$$

∎

Limites melhores para $\lambda_s(n)$ podem ser encontrados em [5]:

$$\lambda_3(n) = \Theta(n\alpha(n))$$

$$\lambda_4(n) = \Theta(n2^{\alpha(n)})$$

$$\lambda_{2s+2}(n) = n2^{\Theta(\alpha^s(n))} \qquad \text{para } s \geq 2$$

$$\lambda_{2s+3}(n) = n2^{O(\alpha^s(n)\log\alpha(n))} \qquad \text{para } s \geq 1$$

onde α é a inversa da função de Ackermann.

2.2 Envelope superior

Seja $\mathcal{C}^*$ o conjunto das funções $y = f(t)$ de valor real. Seja $\mathcal{C} = \{f_1, f_2, \ldots, f_n\}$ um conjunto de n funções em $\mathcal{C}^*$. Definimos o *envelope superior* $\mathcal{E}_{\mathcal{C}}^{sup}$ de $\mathcal{C}$ como:

$$\mathcal{E}_{\mathcal{C}}^{sup}(t) = \max_{1 \leq i \leq n} \{f_i(t)\}$$

onde o máximo é calculado sobre as funções que estão definidas em t. Sejam os intervalos $I_1, I_2, \ldots, I_m$ uma partição da união dos intervalos de definição das funções, ordenados da esquerda para a direita tal que se k é um inteiro entre 1 e m então a mesma função f_{u_k} aparece sobre $\mathcal{E}_{\mathcal{C}}^{sup}$ para todo ponto em I_k, ou seja:

$$\mathcal{E}_{\mathcal{C}}^{sup}(t) = f_{u_k}(t) \qquad \text{para cada } t \in I_k.$$

Denotaremos por $\mathcal{S}^{sup}(\mathcal{C})$ a *seqüência do envelope superior* $u_1u_2\ldots u_m$ de $\mathcal{C}$, onde m é o número de partes das $f_i^{'}s$ que formam o gráfico de $\mathcal{E}_{\mathcal{C}}^{sup}$.

O *envelope inferior* de $\mathcal{C}$, denotado por $\mathcal{E}_{\mathcal{C}}^{inf}$, é definido de forma análoga, ou seja

$$\mathcal{E}_{\mathcal{C}}^{inf}(t) = \min_{1\leq i\leq n}\{f_i(t)\}.$$

Também é definida, analogamente, a *seqüência do envelope inferior* de $\mathcal{C}$. Todos os resultados sobre envelope superior também se aplicam ao envelope inferior.

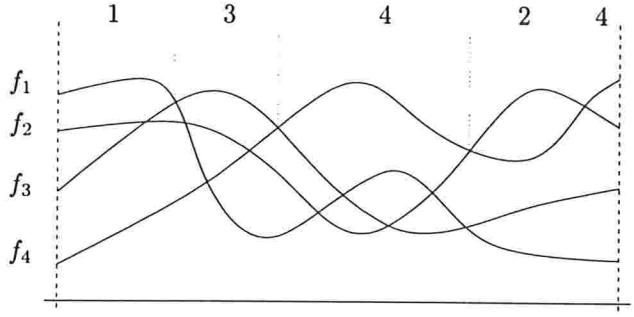


Figura 2.1: Envelope superior e seqüência do envelope superior.

O envelope $\mathcal{E}_{\mathcal{C}}^{sup}$ é formado por partes de funções f_{u_k} em $\mathcal{C}$, onde o índice u_k faz parte da seqüência $\mathcal{S}^{sup}(\mathcal{C})$ e ocorre nas mesmas posições em que f_{u_k} determina $\mathcal{E}_{\mathcal{C}}^{sup}$. A Figura 2.1 mostra o envelope superior de $\mathcal{C} = \{f_1, f_2, f_3, f_4\}$, temos que $\mathcal{S}^{sup}(\mathcal{C}) = (1, 3, 4, 2, 4)$ e que $\mathcal{S}^{inf}(\mathcal{C}) = (4, 1, 2, 3, 1)$.

Se f_1 e f_2 são duas funções em $\mathcal{C}^*$ que se intersectam em quatro pontos, então dependendo do tipo das funções obteremos seqüências do envelope superior de comprimentos diferentes. Observemos que a seqüências resultante pode ser uma $(2, 4)$-seqüência, como ilustrada pela Figura 2.2(a), ou uma $(2, 8)$-seqüência, como ilustrada pela Figura 2.2(b). Estes resultados são devidos aos Teoremas 2.2 e 2.4 os quais mostramos a continuação.

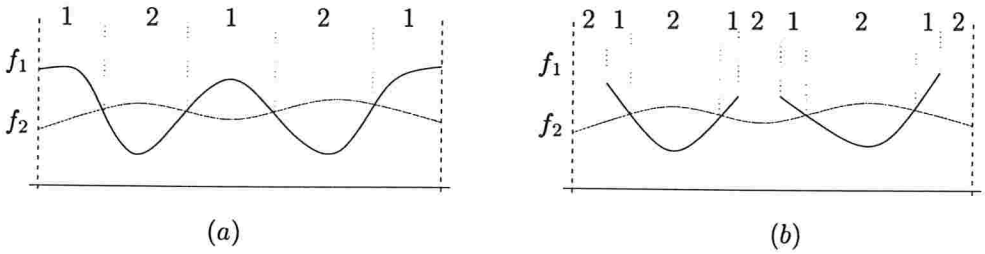


Figura 2.2: O tipo de função determina a descrição combinatória do envelope.

A seguir mostraremos alguns resultados que são necessários na análise da complexidade combinatória de envelopes, ou seja, na análise do comprimento máximo de uma seqüência

do envelope superior de um conjunto de funções. Dependendo do tipo das funções se obtêm diferentes limitantes superiores para o comprimento máximo da correspondente seqüência do envelope superior.

Seja I um intervalo da reta real. Denotaremos por $\mathcal{F}_I^*$ o subconjunto das funções em $\mathcal{C}^*$ definidas sobre I. Seja $\mathcal{F}_s^*$ o subconjunto das funções em $\mathcal{F}_I^*$ tal que quaisquer duas funções distintas em $\mathcal{F}_s^*$ se intersectam no máximo s vezes.

Teorema 2.2 (Agarwal e Sharir [5, 6]) *Se $\mathcal{F}_s^n = \{f_1, f_2, \ldots, f_n\}$ é um conjunto de n funções contínuas em $\mathcal{F}_s^*$ então $\mathcal{S}^{sup}(\mathcal{F}_s^n)$ é uma DS(n, s)-seqüência. Ademais, se σ é uma DS(n, s)-seqüência então existe um conjunto $\mathcal{F}_s^n$ de n funções contínuas em $\mathcal{F}_s^*$ tal que $\mathcal{S}^{sup}(\mathcal{F}_s^n) = \sigma$.*

Prova. Por definição não existe um par de símbolos iguais em $\mathcal{S}^{sup}(\mathcal{F}_s^n)$. Suponha que $\mathcal{S}^{sup}(\mathcal{F}_s^n) = u_1u_2\ldots u_m$ contém $s+2$ índices $i_1 < i_2 < \ldots < i_{s+2}$ tal que $u_{i1} = u_{i3} = \ldots = a$, $u_{i2} = u_{i4} = \ldots = b$ e $a \neq b$, ou seja $\mathcal{S}^{sup}(\mathcal{F}_s^n)$ contém uma subseqüência alternante da forma $abab\ldots$ de comprimento $s+2$. Pela definição de seqüência do envelope superior temos que $f_a(t) > f_b(t)$ para todo t pertencente ao interior dos intervalos $I_{i_1}, I_{i_3}, \ldots$. Analogamente, $f_a(t) < f_b(t)$ para todo t pertencente ao interior dos intervalos $I_{i_2}, I_{i_4}, \ldots$, como f_a e f_b são contínuas então elas se intersectam em $s+1$ pontos, que é uma contradição. Logo $\mathcal{S}^{sup}(\mathcal{F}_s^n)$ é uma DS(n, s)-seqüência.

A prova da segunda parte do teorema encontra-se em Atallah [8]. ∎

Corolário 2.3 *Para qualquer conjunto $\mathcal{F}_s^n = \{f_1, f_2, \ldots, f_n\}$ de n funções contínuas em $\mathcal{F}_s^*$, o comprimento da seqüência do envelope superior $\mathcal{S}^{sup}(\mathcal{F}_s^n)$ não é maior que $\lambda_s(n)$ e existem exemplos onde este limite é alcançado.* ∎

Uma função $g \in \mathcal{C}^*$ tem uma *transição* em $t = t_0$ se é indefinida antes de t_0 e definida depois de t_0 ou se é definida antes de t_0 e indefinida depois de t_0.

Seja $\mathcal{C}_s^*$ o conjunto das funções em $\mathcal{C}^*$ tal que quaisquer duas funções distintas em $\mathcal{C}_s^*$ se intersectam no máximo s vezes.

Teorema 2.4 (Atallah [9, 10]) *Se $\mathcal{C}_s^n = \{f_1, f_2, \ldots, f_n\}$ é um conjunto de n funções em $\mathcal{C}_s^*$, tal que f_i é contínua exceto em não mais de p pontos de descontinuidade e q transições, então a $\mathcal{S}^{sup}(\mathcal{C}_s^n)$ é uma DS$(n, s+2p+2q)$-seqüência.*

Prova. Seja $\mathcal{E}_{\mathcal{C}_s^n}^{sup}(t) = \max_{1\leq i\leq n}\{f_i(t)\}$ o envelope superior de $\mathcal{C}_s^n$ e seja $\mathcal{S}^{sup}(\mathcal{C}_s^n) = u_1u_2\ldots u_m$ a seqüência do envelope superior de $\mathcal{C}_s^n$.

Por definição não existe um par de símbolos iguais em $\mathcal{S}^{sup}(\mathcal{C}_s^n)$. Seja l_{ij} o número de vezes que acontece cada um dos três itens seguintes:

- uma intersecção entre f_i e f_j.
- uma transição ou uma descontinuidade de f_i.
- uma transição ou uma descontinuidade de f_j.

Então pelas hipóteses $l_{ij} \leq s+q+p+q+p = s+2p+2q$. Se $t_1 < t_2$ e $\mathcal{E}_{\mathcal{C}_s^n}^{sup}(t_1) = f_i(t_1)$, $\mathcal{E}_{\mathcal{C}_s^n}^{sup}(t_2) = f_j(t_2)$, então no intervalo $[t_1, t_2]$ acontece ao menos um dos três ítens anteriores. Isto implica que para qualquer subseqüência alternante da forma $\sigma = u_i u_j u_i u_j \ldots$ com comprimento $w+1$ temos que $w \leq l_{ij}$. Logo, como $l_{ij} \leq s+2p+2q$, σ tem comprimento máximo $s+2p+2q+1$. Portanto, $\mathcal{S}^{sup}(\mathcal{C}_s^n)$ é uma DS$(n, s+2p+2q)$-seqüência. ∎

Corolário 2.5 *Para qualquer conjunto $\mathcal{C}_s^n = \{f_1, f_2, \ldots, f_n\}$ de n funções em $\mathcal{C}_s^*$, tal que f_i é continua exceto em não mais de p descontinuidades e q transições, o comprimento da seqüência do envelope superior $\mathcal{S}^{sup}(\mathcal{C}_s^n)$ não é mais que $\lambda_{s+2p+2q}(n)$.* ∎

2.3 Dualidade

Através do conceito de dualidade podemos converter um conjunto de pontos em um arranjo de retas e vice-versa. Uma razão para este tipo de transformação ser útil é que muitas vezes as relações entre pontos são reveladas mais explicitamente através do arranjo de retas correspondente. (Esta observação foi feita por Edelsbrunner [22], página 4.) Existem diversas transformações *ponto-reta* possíveis, dependendo da representação da reta e o contexto do problema tratado. Como cada reta pode ser especificada por dois números, então cada reta pode ser associado a um ponto cujas coordenadas são estes números. Por exemplo, a reta $L : ax + by = 1$ pode ser associado ao ponto $p : (a, b)$; esta transformação é conhecida como *dualidade polar*. No Capitulo 5 desta dissertação usaremos a transformação $p : (m, b) \Leftrightarrow L : y = mx + b$, para relacionar a cadeia superior do fecho convexo de um conjunto de pontos ao envelope superior do arranjo dual destes pontos. Usaremos o símbolo $\mathbb{D}(S)$ para indicar a transformação dual do conjunto de pontos S.

CAPÍTULO 3

Modelos e medidas de complexidade

```
inic LOAD    2
     ADD     =1
     STORE   2
     LOAD    *2
     ADD     =3
     STORE   1
     LOAD    3
     SUB     *1
     STORE   3
     LOAD    2
     ADD     =1
     STORE   2
     JUMP    inic
```

Neste capítulo, descreveremos três modelos que encontramos na literatura para problemas de pontos em movimento: modelo off-line de Atallah [9, 10]; modelo de tempo-real de Kahan [28, 29]; e modelo cinético de Basch, Guibas e Hershberger [13, 14]. Ainda neste capítulo apresentaremos as medidas de complexidade que foram propostas para cada um destes modelos. O modelo de computação para a análise dos algoritmos apresentados é a Máquina de Acesso Aleatório real (real-RAM — *real Random Access Machine*) com custos uniformes (cf. Seção 1.4 de [36] e Capítulo 1 de [7]). Os registradores da máquina real-RAM podem armazenar números reais (que podem ter tamanho arbitrários) e operações como $+$, $-$, $\times$, $\div$, $\sqrt{}$ são exemplos de operações primitivas, isto é, estão disponíveis a custo unitário.

No modelo off-line o movimento de cada ponto, ou seja o seu plano de vôo, é totalmente conhecido de antemão. O plano de vôo tipicamente é representado por um polinômio. No modelo de tempo-real só se conhece de antemão uma informação parcial sobre o movimento dos pontos; um limite superior para a velocidade de cada ponto. Já no modelo cinético, cada ponto tem o seu plano de vôo que pode ser alterado on-line.

Ao analisarmos a complexidade dos algoritmos apresentados estaremos principalmente interessados em estimar a sua complexidade de tempo (i.e., número de operações primitivas feitas por uma máquina real-RAM) e sua complexidade de espaço. No que se refere a complexidade de tempo, dependendo do modelo sob consideração, serão estimados, por exemplo, o tempo gasto para calcular uma seqüência de descrições combinatórias, o tempo gasto para responder a uma consulta, o tempo gasto em pré-processamento e o tempo gasto em processar um evento. Como é usual, as complexidades de tempo e de espaço serão

expressas como funções do tamanho da entrada.

3.1 Modelo off-line

No modelo off-line, o movimento total dos pontos é conhecido de antemão. Nesta situação, estudada por Atallah [9, 10] temos um conjunto de pontos no espaço euclidiano d-dimensional onde as coordenadas de cada ponto são polinômios na variável 'tempo' t; ou seja, a posição de cada ponto é dada através de uma equação da forma $p(t) = c_0 + c_1 t + \cdots + c_k t^k$, onde $c_i \in \mathbb{R}^d$, logo, c_0 indica a posição do ponto no instante $t = 0$. Chamaremos a equação do movimento de um ponto (i.e., o plano de vôo do ponto) de um *k-movimento* se o grau do polinômio que descreve o seu movimento é menor ou igual a k. Assim, os dados de entrada para um problema no modelo off-line são:

- (pontos) n pontos $p_1, \ldots, p_n$ em movimento contínuo;
- (informação sobre o movimento) um polinômio $p_i(t)$ representando o k-movimento do ponto p_i $(i = 1, \ldots, n)$.

Conhecido o movimento total dos pontos, o objetivo dos problemas neste modelo é determinar uma seqüência de descrições combinatórias para o problema sob consideração. Por exemplo, para o Problema do par mais-próximo, deseja-se uma seqüência dos pares de pontos mais-próximos. Atallah apresentou algoritmos que obtêm uma tal seqüência para o problema do máximo, do par mais-próximo e do fecho convexo.

Medidas de complexidade

Na análise da complexidade dos algoritmos, no modelo off-line, supomos que as seguintes operações podem ser realizadas em tempo constante sobre pontos que realizam, para k limitado, um k-movimento:

- calcular a posição $p(t)$ de um ponto p em qualquer instante t;
- resolver equações do tipo $Q(t) = 0$, onde $Q(t)$ é um polinômio de grau limitado.

Ou seja, Atallah assume que obter todas as raízes de um polinômio de grau limitado é uma operação primitiva. Também é suposto que o espaço gasto para representar o k-movimento de cada ponto é constante.

Da mesma maneira que ocorre com algoritmos estáticos, na análise de algoritmos off-line, as complexidades de tempo e de espaço serão expressas como funções do número n de pontos.

Para a análise dos algoritmos é de interesse o número máximo de mudanças na descrição combinatória para o problema tratado. Por exemplo, para o Problema do máximo, que será considerado no próximo capítulo, o número de mudanças da descrição combinatória para pontos que realizam um k-movimento é $O(\lambda_k(n))$, onde $\lambda_k(n)$ é o comprimento da maior seqüência de Davenport-Schinzel de ordem k sobre um alfabeto de n símbolos (veja o Capítulo 2). Atallah obteve limites superiores para o número de mudanças nas descrições combinatórias do par de pontos mais-próximo e do fecho convexo de um conjunto de pontos no plano.

3.2 Modelo de tempo-real

No modelo de tempo-real a única informação conhecida de antemão sobre o movimento de cada objeto é um limite superior para a sua velocidade. Neste modelo, proposto por Kahan em [28, 29], temos um conjunto de n pontos $\mathcal{P} = \{p_1, \ldots, p_n\}$ em $\mathbb{R}^d$ que estão em movimento contínuo, onde para cada ponto p_i temos um limite superior r_i para a sua velocidade ($i = 1, \ldots, n$).

O modelo proposto por Kahan [28] para este tipo de problema está ilustrado na Figura 3.1. No modelo a operação que atualiza a posição de um dado ponto é encarada como sendo a chamada a um oráculo, que pode ser encarado como sendo uma função $U : \mathbb{R} \times \mathcal{P} \to \mathbb{R}^d$ que é chamada de *função de atualização*. Assim, $U(t, p)$ atualiza a posição do ponto p para o instante t. Neste modelo, um usuário ou cliente fornece uma seqüência de consultas on-line a um programa o qual deve respondê-las uma após outra. Para responder às consultas o programa pode requerer ao oráculo (dispositivo físico) a posição atual de alguns dos pontos. Logo, os dados de entrada para este tipo de problema são:

- (pontos) n pontos $p_1, \ldots, p_n$ em movimento contínuo;
- (informação sobre o movimento) um limite r_i para a velocidade do ponto p_i ($i = 1, \ldots, n$);
- (atualização) um oráculo que dado um ponto e um instante fornece a posição do ponto nesse instante;
- (consultas) uma seqüência de consultas $(q_1, q_2, q_3, \ldots)$ que são fornecidas on-line e que devem ser respondidas uma após outra.

Em termos algorítmicos o que se deseja é que as consultas sejam respondidas o 'mais rápido' possível.

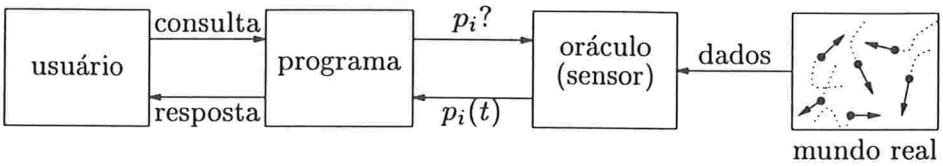


Figura 3.1: O modelo de Kahan.

Devido ao grande número de pontos que estão em movimento é impraticável ao programa, por questão de tempo, atualizar a posição de todos os pontos antes de responder a cada consulta. Desta forma, qualquer algoritmo incumbido de responder às consultas fornecidas pelo usuário deve escolher alguns pontos (não muitos) que terão as suas posições atualizadas, antes que uma resposta possa ser dada. Para fixarmos as idéias consideremos o exemplo apresentado por Kahan [28]. Em animação de um simulador de vôo uma consulta pode ser: "Quais aviões estão no campo de visão?". Para responder a esta pergunta é certamente desnecessário atualizarmos as posições de aviões que no instante anterior estivessem longe do campo de visão e cujos limites de velocidade não permitem que estes já tenham alcançado o campo de visão. Assim, para um algoritmo que responda a este tipo de consulta basta atualizar somente as posições daqueles aviões que têm alguma chance de estarem no campo de visão. Kahan propôs, para alguns problemas, estratégias que, levando em conta a velocidade máxima de cada ponto, escolhem os pontos que devem ter as suas posições atualizadas.

Na realidade o modelo de problema estudado por Kahan em [28, 29] não se enquadra perfeitamente no que chamamos de problema de tempo-real, tendo em vista que a decisão para atualizar a posição de um ponto deixou de fazer parte da entrada e passou a fazer parte do algoritmo. Kahan [28] chamou este tipo de problemas de *Data Motion Problems.*

O modelo está ilustrado na Figura 3.1: o usuário faz uma consulta on-line ao programa; o dispositivo de atualização captura uma "foto instantânea" do mundo real, envolvendo os objetos em movimento; o programa ao processar a consulta pode requerer posições atuais de alguns pontos o qual é fornecido através do oráculo considerando a foto instantânea; o programa processa as posições atuais dos objetos, talvez faça novos requerimentos de posições atuais considerando a mesma foto instantânea; finalmente devolve o valor da consulta ao usuário. Assim, o valor da consulta corresponde ao instante no qual foi feita à consulta.

Outros exemplos de problemas em ciência da computação, do tipo Data Motion Problems são: determinar o fecho convexo de pontos em movimento; controle de tráfego aéreo; manter posições relativas de objetos em movimento em aplicações de visão computacional; e comutar estação de transferência para telefonia celular em movimento.

Medidas de complexidade

A eficiência dos algoritmos para o tipo de problema de tempo-real tratado por Kahan será medida através do número de operações primitivas feitas pelo algoritmo e também através do número de vezes que o algoritmo chama o oráculo para atualizar a posição de um determinado ponto. No modelo de tempo-real a chamada ao oráculo é considerada uma operação primitiva, ou seja, o custo para atualizar a posição de um ponto é constante.

Seja $Q := (q_1, q_2, q_3, \ldots)$ uma seqüência de consultas on-line. Denotaremos por $C_A^T(q)$ e $C_A^U(q)$, respectivamente, o tempo gasto e o número de atualizações feitas por um algoritmo A para responder à consulta q. O *tempo gasto no pior caso* por um algoritmo A é $\sup_Q \max_{q_i \in Q} C_A^T(q_i)$. Um algoritmo não-determinístico L é dito *sortudo* (*lucky*) se para responder corretamente a qualquer consulta o algoritmo executa um número mínimo de atualizações, ou seja chamadas ao oráculo. Diremos que um algoritmo A é *k-sortudo* (*within k of lucky*) se $C_A^U(q) \leq C_L^U(q) + k$ para toda consulta q e diremos que o algoritmo A é *k-fator-sortudo* (*k-lucky*) se para um número de pontos suficientemente grande $C_A^U(q) \leq kC_L^U(q) + O(1)$ para toda consulta q. Um algoritmo A é *k-sortudo ótimo* (*update optimal*) se A é k-sortudo e não existe um algoritmo k'-sortudo para $k' < k$ e diremos que A tem *k-fator-sortudo ótimo* (*optimal lucky ratio*) se A é k-fator-sortudo e não existe um algoritmo k'-fator-sortudo para $k' < k$.

Chamaremos de *região livre* (*free range*) de um ponto p à menor região do espaço $\mathbb{R}^d$ que sabemos com certeza conter o ponto p. Assim, no instante em que a posição de um ponto é atualizada a sua região livre se restringe a um único ponto. O conjunto de todas as regiões livres é o *estado do conhecimento* (*state of the knowledge*) do problema.

A medida de desempenho relativa proposta por Kahan é semelhante à análise de algoritmos on-line competitivos (*competitive on-line algorithms*) sugerida por Sleator e Tarjan [41], onde o desempenho de um algoritmo on-line é medido através da comparação com o desempenho de um algoritmo (determinístico) off-line ótimo.

3.3 Modelo cinético

No modelo cinético, o movimento dos pontos pode ser alterado on-line. A estrutura de dados cinética ($\mathbb{EDC}$) proposta por Basch, Guibas e Hershberger [13, 14] tem como objetivo manter continuamente a descrição combinatória de um objeto geométrico determinado por um conjunto de pontos em movimento. Os dados de entrada para um problema, no modelo cinético, são:

- (pontos) n pontos $p_1, \ldots, p_n$ em movimento contínuo;

- (informação sobre o movimento) um plano de vôo para cada ponto;
- (atualização) uma seqüência de alterações de planos de vôo que são fornecidas on-line.

O plano de vôo pode ser uma equação descrevendo o movimento do ponto, esta equação é tipicamente um polinômio. Não incluímos consultas na entrada já que a descrição combinatória do objeto geométrico de interesse está prontamente disponível ao usuário a todo instante; tudo se passa como se cada vez que o usuário desejasse, digamos, o fecho convexo dos pontos, o programa fornecesse prontamente a sua descrição combinatória.

Apesar dos pontos se moverem continuamente e, portanto, o objeto geométrico considerado também estar se movendo continuamente, a descrição combinatória deste objeto só muda em certos momentos críticos discretos. A técnica utilizada por Basch, Guibas e Hershberger [13, 14] para se manter a descrição combinatória de um objeto geométrico em movimento contínuo se concentra exatamente nestes momentos críticos.

Definimos um *certificado* como sendo simplesmente um predicado geométrico sobre um número constante de pontos em movimento. Por exemplo, consideremos dois pontos p e q no plano, um certificado pode ser algo do tipo "o ponto p está acima do ponto q". Usando os planos de vôo dos pontos podemos, para cada certificado, determinar o momento em que este deixa de ser válido, isto é, podemos computar o seu *prazo de validade*. Os certificados são armazenados em uma fila de prioridade que os mantêm em ordem de prazo de validade. A violação de um certificado é chamada de *evento*. Quando um evento causa uma mudança na descrição combinatória do objeto geométrico mantido ele é chamado de *evento externo* (ou *evento topológico*, cf. [25, 38, 39, 37]); quando ele não afeta a descrição, mas causa somente atualizações nos certificados, na fila e em outras eventuais estruturas de dados, por razões de consistência, ele é dito um *evento interno*. Observe que o número de eventos externos depende do tipo de objeto geométrico mantido e do tipo de movimento. No entanto, o número de eventos internos depende da estrutura de dados cinética para o problema considerado. Os eventos associados aos certificados $\texttt{InCircle}(p_1, p_2, p_3, p_4)$ e $\texttt{Left}(p_1, p_2, p_3)$, ilustrados na Figura 3.2 (que já foi vista na Introdução), são externos.

Uma estrutura de dados cinética é constituída por certificados, algoritmos, e estruturas de dados, que em conjunto mantêm a descrição combinatória do objeto geométrico de interesse. Mais especificamente, uma $\mathbb{EDC}$ consiste de:

- um algoritmo de preprocessamento;
- um conjunto de certificados;
- algoritmos para processar os diversos tipos de eventos e as mudanças de plano de vôo;

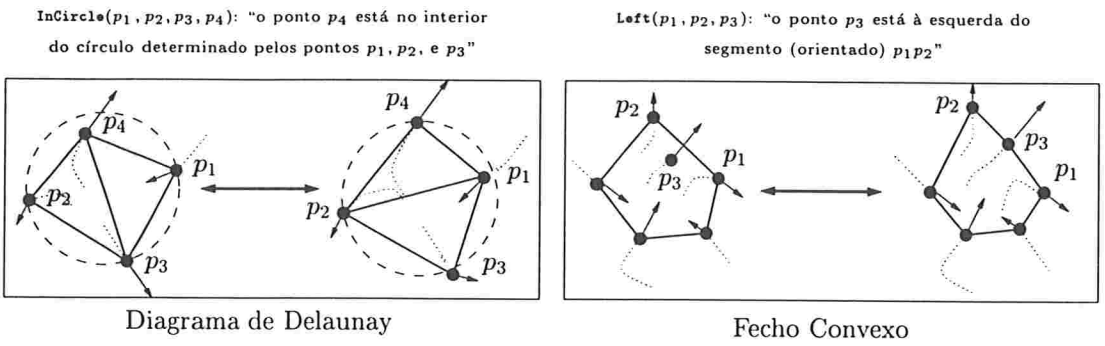


Diagrama de Delaunay

Fecho Convexo

Figura 3.2: Ilustração dos certificados `InCircle` e `Left`.

- uma fila de prioridade contendo os certificados em ordem de prazo de validade, também chamaremos esta fila de uma *fila de eventos*; e
- algumas estruturas de dados auxiliares.

O algoritmo de preprocessamento produz a descrição combinatória e determina os certificados iniciais. Estes certificados serão armazenados em uma fila de prioridade que os mantém em ordem de prazo de validade. Além disso, este algoritmo deve inicializar eventuais estruturas de dados auxiliares.

O conjunto de certificados mantidos pela estrutura de dados cinética formam uma prova da corretude da descrição combinatória atual.

Os algoritmos que processam os diversos tipos de eventos e as mudanças de plano de vôo têm como objetivo manter a descrição combinatória do objeto geométrico de interesse e assim continuar com a "simulação" do movimento dos pontos. Quando ocorre um evento o certificado que perdeu a sua validade deve ser removido da fila e os certificados para a nova descrição combinatória do objeto devem ser calculados. Talvez outros certificados também tenham que ser removidos da fila e alguns outros novos tenham que ser inseridos. Quando um ponto muda o seu plano de vôo, todos os certificados na fila de eventos que dependem dele devem ser recalculados e ter suas posições na fila atualizadas. O novo conjunto de certificados devem formar uma prova de corretude para a nova descrição combinatória.

Como determinar uma $\mathbb{EDC}$ para um dado problema no modelo cinético? Basch, Guibas e Hershberger [13, 14] descrevem algumas técnicas de como determinar tais estruturas. Eles aplicaram as suas técnicas ao Problema do máximo, par mais-próximo, fecho convexo e outros. Inicialmente, os autores consideram um algoritmo apropriado que resolve a versão estática do problema de interesse, posteriormente este algoritmo é usado para criar a estrutura de dados cinética. O processo de se obter uma estrutura de dados cinética a partir de um algoritmo estático é chamado de *cinetização*. Uma técnica geral de cinetização consiste

em tomar-se como certificados os cálculos intermediários feitos pelo algoritmo estático.

Medidas de complexidade

Na análise da complexidade dos algoritmos cinético suporemos que as seguintes operações podem ser realizadas em tempo constante, ou seja, estas operações são consideradas elementares:

- calcular o posição de um ponto em qualquer instante;
- resolver equações do tipo $Q(t) = 0$, onde $Q(t)$ é um polinômio de grau limitado.

Uma estrutura de dados cinética é avaliada de diversas maneiras diferentes. Denotaremos por n o número de pontos em movimento. Uma estrutura de dados cinética que seja boa deve manter uma estrutura de certificados que permita que o custo de processar qualquer evento seja pequeno, por 'pequeno' queremos dizer assintoticamente poli-logarítmico em n, i.e. $O(\text{polilog}\, n)$, ou $O(n^{\epsilon})$ para alguma constante positiva 'pequena' ϵ. Uma estrutura de dados que satisfaça esta condição será dita de *resposta rápida* (*responsive*). Existem outras propriedades que são desejáveis em uma estrutura de dados cinética. Intuitivamente é interessante que a razão entre o número total de eventos e o número de eventos externos seja o menor possível, já que o número de eventos externos é um limite inferior para o custo de qualquer algoritmo que mantenha a descrição combinatória. A estrutura de dados será chamada *eficiente* se a razão entre o número total de eventos processados no pior caso e o número de eventos externos no pior caso for pequena.

Agora voltemos nossa atenção para aspectos de espaço gasto por uma estrutura de dados cinética. Nós diremos que a estrutura é *compacta* se o número máximo de certificados na fila de prioridade for assintoticamente linear no número de pontos. Finalmente, a estrutura será *local* se para cada ponto dado o número máximo de certificados na fila de prioridade envolvendo o ponto é pequeno (i.e., $O(\text{polilog}\ n)$). Esta propriedade é crucial para o tratamento eficiente de mudanças em planos de vôo.

Considere, por exemplo, a estrutura de certificados considerada na introdução que forma uma prova da corretude da descrição combinatória da Triangularização de Delaunay. Nesta estrutura temos um certificado do tipo `InCircle` para cada triângulo. Como cada ponto pode pertencer a $\Omega(n)$ destes triângulos, então a mudança no plano de vôo de um ponto pode chegar a causar a atualização de $\Omega(n)$ certificados. Portanto, apesar da estrutura de dados cinética derivada desta estrutura de certificados ser compacta, ela não é local e, portanto, não é apropriada para alterações no plano de vôo.

Ao analisarmos uma estrutura de dados cinética estaremos interessados em determinar se ela é de resposta rápida, eficiente, compacta e local; se for assim ela será chamada de *ótima*.

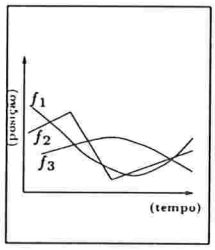
***

O modelo de Atallah é apropriado para realizar estudos teóricos sobre a descrição combinatória do objeto geométrico considerado. Os resultados dependeram do tipo de movimento dos pontos. Neste modelo, a seqüência formada pela descrições combinatórias do objeto geométrico considerado tem comprimento finito.

No modelo de Kahan, construir uma solução para um problema dado, compreende duas fases: identificar uma estratégia que não faz muitas atualizações para resolver o problema, e encontrar um algoritmo eficiente que implementa tal estratégia.

Também, no modelo cinético, construir uma solução para um problema dado compreende duas fases: resolver a versão estática do problema, e manter a estrutura de certificados quando começa o movimento dos pontos. Os cálculos prévios são mantidos com o objetivo de atualizar a estrutura de certificados mais eficientemente.

Problema do máximo



Formulamos o *problema do máximo* da seguinte maneira: *dados*: n pontos movendo-se de maneira contínua sobre o eixo das ordenadas; *determinar*: um ponto de maior ordenada. (Veja Figura 4.1.) Diremos que a ordenada de um ponto é o seu *valor*. O problema estático do máximo consiste simplesmente em encontrar o máximo de uma seqüência de números. Este é um problema básico que aparece como sendo o primeiro problema tratado nos artigos de Atallah [9, 10], Kahan [28] e Basch, Guibas e Hershberger [13, 14]. Dependendo do modelo os pontos serão dados de maneiras diferentes: através de planos de vôos ou de limites de velocidade e oráculos.

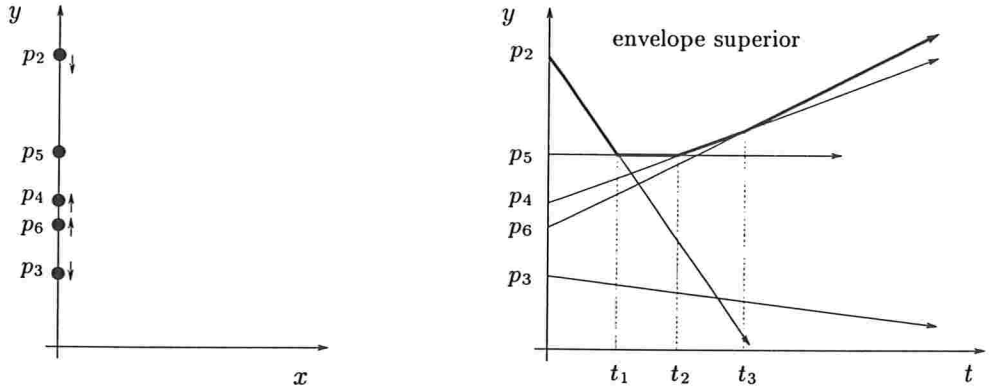


Figura 4.1: Pontos movendo-se no eixo das ordenadas a uma velocidade constante.

Na Seção 4.1 trataremos do Problema do máximo no modelo off-line. Como veremos, neste modelo, o problema se reduz ao cálculo do envelope superior das funções que descrevem o movimento dos pontos.

No modelo de tempo-real os pontos se movem sobre o eixo das ordenadas respeitando limites de velocidade dados. O único tipo de consulta que deve ser respondida em tempo-real

é "Qual é o ponto com maior valor?". Descreveremos, na Seção 4.2, o algoritmo apresentado por Kahan [28] para tratar do Problema do máximo no modelo de tempo-real. A estratégia utilizada por Kahan, para responder às consultas, consiste em somente atualizar a posição dos pontos que são 'candidatos a máximo'.

No modelo cinético, para cada um dos pontos em movimento sobre o eixo das ordenadas é dado o seu plano de vôo e uma seqüência de alterações de planos de vôo que são fornecidas on-line. Neste modelo, o primeiro problema que devemos encarar é o de encontrar uma estrutura de certificados conveniente. Consideraremos as três estruturas de certificados apresentadas por Basch, Guibas e Hershberger [13, 14] que ilustrarão exemplos simples de estruturas de dados cinéticas ($\mathbb{EDC}$). Na Seção 4.3 consideraremos três estruturas de certificados apresentadas por Basch, Guibas e Hershberger [13, 14] que ilustrarão exemplos simples de estruturas de dados cinéticas ($\mathbb{EDC}$). A primeira estrutura é denominada *Ordem Cinética*, cuja estrutura de certificados forma uma prova da ordem dos valores dos pontos. A segunda estrutura é chamada de *Heap Cinético* que é uma $\mathbb{EDC}$ onde os certificados são provenientes de um heap. Finalmente, a terceira estrutura é chamada de *Torneio Cinético* na qual a estrutura de certificados é baseada em um torneio.

4.1 Máximo off-line

O *problema do máximo*, no modelo off-line, consiste em *dado*: um conjunto S de n pontos em movimento sobre o eixo das ordenadas; e a equação do k-movimento de cada um desses pontos (os planos de vôo dos pontos); *determinar*: uma seqüência de pontos de maior ordenada de S ao longo do tempo.

A *seqüência de pontos de maior ordenada* de S consiste em uma seqüência formada por pontos de S. Nesta seqüência os pontos de S são listados na ordem cronológica em que eles aparecem como ponto de maior ordenada. Assim, o primeiro ponto na seqüência é um ponto de maior ordenada no instante inicial $t = t_0$ e o último ponto na seqüência é um ponto de maior ordenada no instante $t = +\infty$.

Sejam $y_1, \ldots, y_n$ as n funções reais que descrevem o movimento de cada ponto sobre o eixo das ordenadas. Deseja-se, para cada instante t, determinar qual é o ponto com a maior ordenada, isto é equivalente a obtermos $h(t) := \max_{1 \leq i \leq n}\{y_i(t)\}$. A função h é formada por partes das funções $y_1, \ldots, y_n$, este número de partes é chamado de *complexidade combinatória* de h. O gráfico da função h é o envelope superior das curvas formada pelos gráficos de $y_1, \ldots, y_n$ (a situação encontra-se ilustrada na Figura 4.2).

Atallah determinou a complexidade combinatória da função h relacionando envelope superiores ao comprimento $\lambda_s(n)$ da maior seqüência de Davenport-Schinzel de ordem s

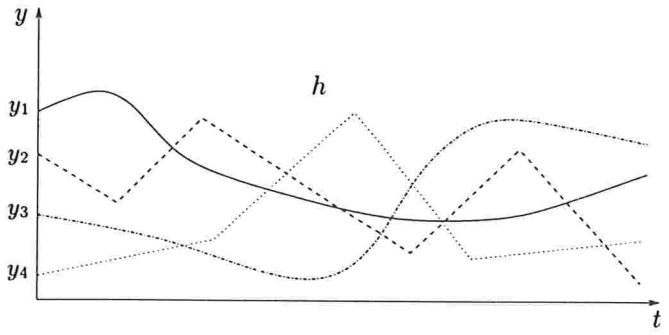


Figura 4.2: A função h.

sobre um alfabeto de n símbolos. Se $y_1, \ldots, y_n$ são funções reais contínuas tais que y_i e y_j coincidem em no máximo s pontos, $1 \leq i < j \leq n$, então a função h é formada por no máximo $\lambda_s(n)$ partes, e este limite é o melhor possível (veja Corolário 2.3). A complexidade combinatória de h é um limite inferior para a complexidade de qualquer algoritmo que resolve o problema do máximo off-line (No modelo cinético a alternância entre partes de h corresponderão a eventos externos). Outros resultados sobre a complexidade combinatória da função h, no caso em que as funções que descrevem o movimento dos pontos tem descontinuidades ou transições, pode-se encontrar no Capítulo 2.

Atallah mostrou como uma descrição combinatória de h pode ser construída através de um algoritmo do tipo divisão-e-conquista em tempo $O(\lambda_s(n) \log n)$, onde $\lambda_s(n)$ é um limitante superior para a complexidade combinatória de h. Lembremos que para s fixo $\lambda_s(n)$ é da ordem de $n \log^* n$ (veja Capítulo 2). Agora, dada a descrição combinatória de h podemos determinar o ponto de S de maior valor em tempo $O(\log \lambda_s(n))$. O seguinte algoritmo calcula a descrição combinatória de h.

Algoritmo EnvelopeRecursivo. Recebe um conjunto S de n funções reais que descrevem o movimento de cada ponto sobre o eixo das ordenadas e devolve, para cada instante t, o ponto com a maior ordenada.

Alternativa I $|S| = 1$

Devolva a única função de S e pare.

Alternativa II $|S| > 1$

Particione o conjunto S em dois subconjuntos, S_e e S_d, de tamanhos aproximadamente iguais.

Seja E_e e E_d o envelope superior de S_e e S_d, respectivamente, ou seja:

$E_e(t) = \max_{y_i \in S_e}\{y_i(t)\}$.

$E_d(t) = \max_{y_i \in S_d}\{y_i(t)\}$.

Devolva $E(t) = \max\{E_e(t), E_d(t)\}$ e pare.

4.2 Máximo de tempo-real

O *problema do máximo*, no modelo de tempo-real, consiste em *dados*: um conjunto S de n pontos em movimento contínuo sobre o eixo das ordenadas; para cada ponto de S um limite superior sobre sua velocidade; um oráculo que atualiza a posição de um ponto dado; e uma seqüência de consultas do tipo "Qual o ponto de maior valor?", que são fornecidas on-line; *responder em tempo-real*: cada consulta um após outra.

Neste modelo os pontos $y_1, \ldots, y_n$ estão em movimento sobre o eixo das ordenadas com limites de velocidade $r_1, \ldots, r_n$, respectivamente. O pontos correspondem a objetos externos ao computador, assim o valor de cada ponto deve ser obtido através de uma chamada ao oráculo (que pode ser um dispositivo físico como um radar). Denotaremos por $y_1(t), \ldots, y_n(t)$ os valores dos pontos no instante t.

A estratégia ingênua para responder a uma consulta é primeiro atualizar a posição de cada um dos pontos, através de n chamadas ao oráculo, e depois comparar os seus valores. Para melhorar a estratégia ingênua podemos apelar para o senso comum: podemos temporariamente ignorar os pontos que, quando da última atualização, tinham valores bem menores que o máximo e que devido aos limites de velocidades certamente não alcançaram o valor máximo atual.

Consideremos a seguinte estratégia n-fator-sortudo. Quando um ponto y_i é atualizado no instante t, ele obtém a prioridade $p_i = (\max(t) - y_i(t))/(r_i + r_{\max}) + t$, onde $\max(t)$ é o valor do ponto máximo no instante t e $r_{\max}$ é o limite de velocidade do ponto máximo. Esta prioridade representa o menor instante em que o ponto y_i poderia alcançar o ponto que era o máximo no instante t. Assim, se um consulta é feita em um instante T então o subconjunto de pontos com prioridade não maior que T deve ser atualizado para determinarmos o ponto máximo e respondermos à consulta.

Esta estratégia baseada em tempo-para-alcançar-o-máximo pode economizar atualizações, mas em geral não constitui uma boa estratégia. De fato, suponha que um ponto máximo em um instante t tenha o seu valor acrescido com velocidade máxima. Suponha que uma nova consulta é feita no instante T. A um algoritmo sortudo basta somente atualizar o valor de um ponto (o do máximo) para responder à consulta corretamente (lembremos que um algoritmo sortudo é um algoritmo não-determinístico que atualiza um número mínimo de pontos). Em contraste, um algoritmo baseado nessa estratégia que prioriza o tempo poderia chegar a fazer n atualizações para a mesma situação, se a prioridade de maior valor

não for maior que o instante T no qual é feito a consulta.

A seguir descrevemos uma estratégia mais eficiente, proposta por Kahan, a qual atualiza os pontos de acordo com uma outra medida de prioridade.

Para $i = 1, \ldots, n$ associaremos o ponto y_i à reta $l_i(t) = r_i(t - t') + y_i(t')$ que passa pelo ponto $(t', y_i(t'))$ com tangente igual a r_i, onde t' é o instante mais recente no qual o valor do ponto y_i foi atualizado. Chamaremos uma tal reta de *reta limite* do ponto y_i (em relação ao instante t'). A Figura 4.3 ilustra várias retas limites que são associadas a pontos. Diremos que o valor $l_i(T)$ é o *valor limite* do ponto y_i no instante T, tendo em vista que $l_i(T)$ é um limitante superior para o valor do ponto y_i no instante T $(i = 1, \ldots, n)$.

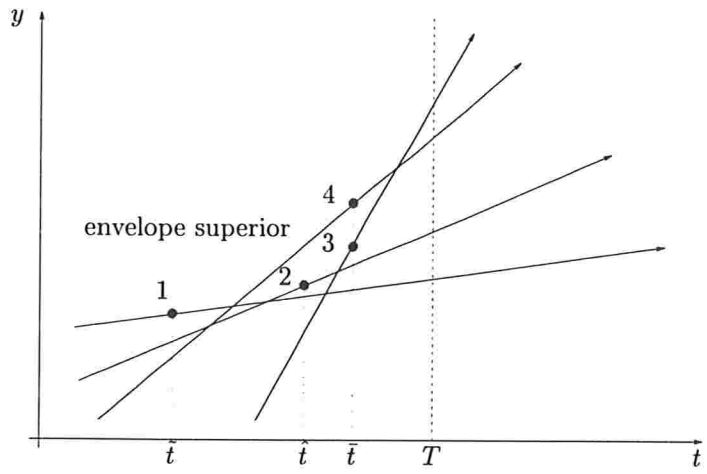


Figura 4.3: Retas limites para velocidades não uniforme.

Suponhamos que uma consulta é feita no instante T. As ordenadas dos pontos de intersecção da reta $x = T$ correspondem aos valores limites do respectivos pontos. A estratégia para atualização de pontos, proposta por Kahan, a fim de responder à consulta feita consiste em atualizar a posição dos pontos, em ordem decrescente dos seus valores limites, até que o valor atualizado de um ponto seja maior que os valores limites de todos os pontos que ainda não foram atualizados no instante T. Quando um ponto y_i é atualizado no instante T, a reta limite correspondente a ele é transladada de tal forma que passe pelo ponto $(T, y_i(T))$.

Em seu artigo Kahan [28] mostra que o algoritmo que implementa a estratégia acima leva tempo $O(u \log^2 n)$ para responder a uma consulta, onde u é o número de atualizações. A seguir descrevemos mais detalhadamente esta estratégia para o caso particular no qual todos os pontos têm o mesmo limite de velocidade. Mostraremos que o algoritmo baseado nesta estratégia é 1-sortudo e gasta tempo $O(u \log n)$ para responder a uma consulta. Antes de prosseguirmos lembremos o que significa um algoritmo ser k-sortudo. Seja $C_A^U(q)$ o número

de atualizações feitas por um algoritmo A para responder à consulta q. Um algoritmo não-determinístico L é dito *sortudo* se para responder corretamente a qualquer consulta o algoritmo executa um número mínimo de atualizações, ou seja chamadas ao oráculo. Finalmente, dizemos que um algoritmo A é k-sortudo se $C_A^U(q) \leq C_L^U(q) + k$ para toda consulta q.

Limite de velocidade uniforme

Consideremos a situação mais simples onde o limite de velocidade de todos os pontos é o mesmo, isto é, $r_1 = \cdots = r_n = r$. A Figura 4.4 ilustra as retas limites para esta situação. Para limite de velocidade uniforme a estratégia de atualização, proposta por Kahan, é mais eficiente que a estratégia baseada em tempo-para-alcançar-o-máximo.

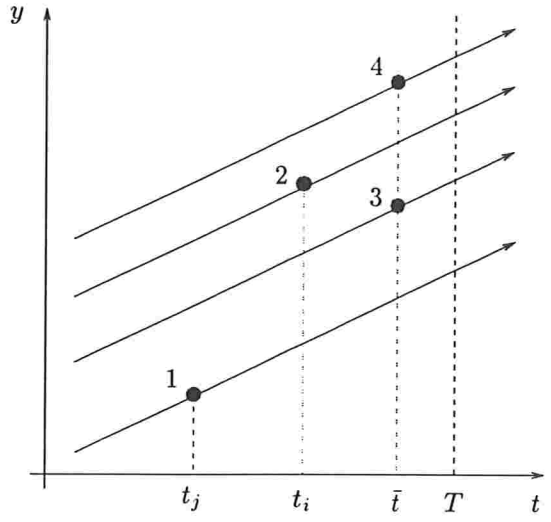


Figura 4.4: Limite de velocidade uniforme.

Suponhamos que uma consulta é feita no instante T. Denotemos por t_i o instante da última atualização do valor do ponto y_i $(i = 1, \ldots, n)$. Suponhamos que no instante T o valor limite do ponto y_i é maior ou igual ao valor limite do ponto y_j, ou seja,

$$l_i(T) = y_i(t_i) + r(T - t_i) \geq y_j(t_j) + r(T - t_j) = l_j(T),$$

que é equivalente a

$$y_i(t_i) - rt_i \geq y_j(t_j) - rt_j.$$

Definimos a prioridade p_i do ponto y_i como sendo $y_i(t_i) - rt_i$ $(i = 1 \ldots n)$. Notemos que a prioridade não depende do instante atual T em que a consulta foi feita. Portanto, para responder à consulta feita, primeiro consideramos um ponto com prioridade máxima e

atualizamos o seu valor que será o nosso máximo temporário `maxtemp`. A seguir, passamos a atualizar a posição dos pontos em ordem decrescente de prioridade, e fazemos `maxtemp` := $\max\{\texttt{maxtemp}, y_i(T)\}$, onde y_i é o ponto que acabou de ser atualizado. O processo pára quando encontramos um ponto y_i tal que a sua prioridade p_i é menor ou igual a `maxtemp` $- rT$. Finalmente, a consulta é respondida e as prioridades de todos os pontos atualizados é recalculada.

Chamaremos de *Algoritmo MaximoDeTempoReal* o algoritmo que implementa a estratégia que acabamos de descrever para responder a uma consulta. Kahan mostrou que o Algoritmo MaximoDeTempoReal faz no máximo uma atualização a mais que um algoritmo sortudo.

> **Algoritmo MaximoDeTempoReal.** Recebe um conjunto $S := \{y_1, \ldots, y_n\}$ de pontos em movimento contínuo sobre o eixo das ordenadas, um limite de velocidade r para todos os pontos, uma consulta feita no instante T, um oráculo que fornece a posição de qualquer ponto no instante T; e devolve um ponto com maior valor no instante T.

Cada iteração começa com um subconjunto P dos pontos de S que ainda precisam ser examinados pelo algoritmo, um real `maxtemp` correspondente ao valor do ponto máximo temporário, e um ponto y que será examinado na iteração. Denotaremos por Q o conjunto $S - P$. A primeira iteração começa com $P = S$, $Q = \emptyset$, `maxtemp` $= -\infty$, e y sendo o ponto de S com maior prioridade. Cada iteração consiste no seguinte:

Alternativa I $P \neq \emptyset$

Seja p a prioridade do ponto y.

Caso 1 $p >$ `maxtemp` $- rT$

Atualize o valor do ponto y.
Seja `maxtemp`$' = \max\{\texttt{maxtemp}, y(T)\}$ e Q' o conjunto $Q + y$.
Seja ainda P' o conjunto $P - y$ e seja y' o ponto de P' com maior prioridade.
Comece nova iteração com P', Q', `maxtemp`$'$ e y' nos papéis de P, Q, `maxtemp` e y.

Caso 2 $p \leq$ `maxtemp` $- rT$

Determine a nova prioridade dos pontos em Q para uma futura consulta.
Devolva o ponto de valor `maxtemp` e pare.

Alternativa II $P = \emptyset$

Determine a prioridade de todos os pontos em Q $(= S)$ para uma futura consulta.
Devolva o ponto de valor `maxtemp` e pare.

Corretude do Algoritmo MaximoDeTempoReal

O algoritmo atualiza o valor dos pontos em ordem decrescente de prioridade, ou, equivalentemente, em ordem decrescente de valor limite no instante T. No início o valor do ponto com maior prioridade (que é um candidato a máximo), será o máximo temporário. Enquanto o Caso 1 é executado, o valor do ponto y examinado na iteração tem chance de ser maior que `maxtemp`, então seu valor é atualizado e comparado com `maxtemp`. Suponha que para um ponto y é válido o Caso 2 e que t foi o último instante no qual este ponto foi atualizado, isto implica que:

$$p + rT = y(t) + r(T - t) \leq \texttt{maxtemp}$$

ou seja, o valor limite para o ponto y, no instante T, é menor que o valor do máximo temporário, o mesmo acontece para todos os demais pontos que tem prioridade menor que a prioridade do ponto y. Neste passo, `maxtemp` é o valor de um ponto máximo.

Análise do Algoritmo MaximoDeTempoReal

No pior caso o algoritmo faz n atualizações. Se implementamos o algoritmo fazendo uso de uma fila de prioridade para representar o conjunto P, que contém as prioridades dos pontos, a complexidade de tempo do Algoritmo MaximoDeTempoReal, para responder uma consulta q, é $C_A^T(q) = O(u \log n)$, onde u é o número de atualizações feitas pelo algoritmo. No modelo de tempo-real deseja-se minimizar este número u de consultas feitas ao oráculo.

Teorema 4.1 (Kahan [28]) *O Algoritmo MaximoDeTempoReal é 1-sortudo.* ∎

Kahan também considera alguns problemas semelhantes ao problema do máximo: o problema da mediana e o problema do menor intervalo.

4.3 Máximo cinético

O *problema do máximo*, no modelo cinético, consiste em *dado*: um conjunto S de n pontos em movimento contínuo sobre o eixo das ordenadas; um plano de vôo para cada ponto, uma seqüência de alterações de planos de vôo que são fornecidas on-line; *manter*: a descrição combinatória do ponto de maior ordenada, ou seja o ponto de maior valor.

Seja y_p $(i = 1 \ldots n)$ a ordenada de cada ponto p. Deseja-se, neste modelo, manter continuamente o ponto de maior ordenada. Consideraremos a seguir três estruturas de dados cinética apresentadas por Basch, Guibas e Hershberger [13, 14] que resolvem o problema do máximo cinético.

4.3.1 Sort cinético

A primeira $\mathbb{EDC}$ para o Problema do máximo, a qual chamaremos de *Sort cinético*, é baseado na cinetização de uma lista de pontos ordenados de acordo com sua y-coordenada. Pode-se usar qualquer algoritmo de ordenação para preprocessar os pontos e criar a lista. Para cada par de pontos consecutivos na lista teremos um certificado que será inserido em uma fila de prioridade ordenada de acordo com o seu prazo de validade. Este prazo de validade consiste no tempo que decorrerá até que estes pontos se encontrem (neste instante a sua ordem relativa será alterada). Quanto menor o prazo de validade, mais próximo do início da fila se encontra um certificado. Quando dois pontos consecutivos se encontram temos um evento, ou seja, a violação de um certificado. Assim que ocorre um evento a ordem na lista dos pontos envolvidos deverá ser trocada e, no máximo, três certificados deverão ser removidos da fila e três novos certificados deverão ser inseridos, como ilustrado na Figura 4.5. Como operações básicas envolvendo a fila de prioridade gastam tempo $O(\log n)$, então esta $\mathbb{EDC}$ é de resposta rápida. Para cada ponto temos que a fila de prioridade tem no máximo dois certificados envolvendo o ponto. Logo, esta $\mathbb{EDC}$ é local. Esta $\mathbb{EDC}$ também é compacta já que o número total de certificados na fila de prioridade é $n-1$. Infelizmente a $\mathbb{EDC}$ não é eficiente. De fato, imagine a metade dos pontos parados e a outra metade passando sobre eles com velocidade constante. Nesta situação a $\mathbb{EDC}$ deve processar $\Theta(n^2)$ eventos, enquanto que o número eventos externos no pior caso (i.e., mudanças na descrição combinatória) é $O(n)$. A situação está ilustrada na Figura 4.6.

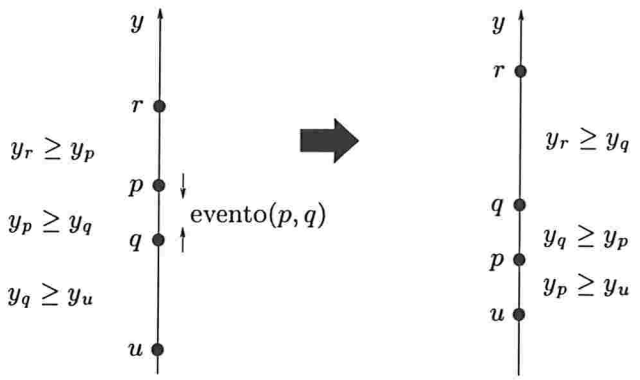


Figura 4.5: Processamento de um evento na $\mathbb{EDC}$ Sort cinético.

4.3.2 Heap cinético

A segunda $\mathbb{EDC}$ para o Problema do máximo, que será chamada de *Heap cinético*, é baseada na cinetização de um heap binário. Um algoritmo estático deverá preprocessar os pontos a

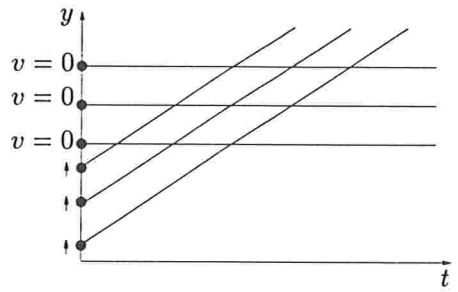


Figura 4.6: Exemplo onde uma estrutura de dados cinética baseada em certificados de consecutividade dos pontos deve processar $\Theta(n^2)$ eventos.

fim de criar um heap que deverá conter os pontos de acordo com o valor da sua ordenada.

A cada aresta do heap faremos corresponder um certificado que garante que o ponto associado ao nó filho está abaixo do ponto associado ao nó pai (por abaixo entenda-se 'tem ordenada menor'). Logo, eventos ocorrem quando pai e filho se encontram. Processar um evento consiste em: trocar as posições dos nós pai e filho no heap; remover da fila no máximo cinco certificados; e inserir na fila no máximo cinco novos certificados. O processamento de um evento está ilustrado na Figura 4.7.

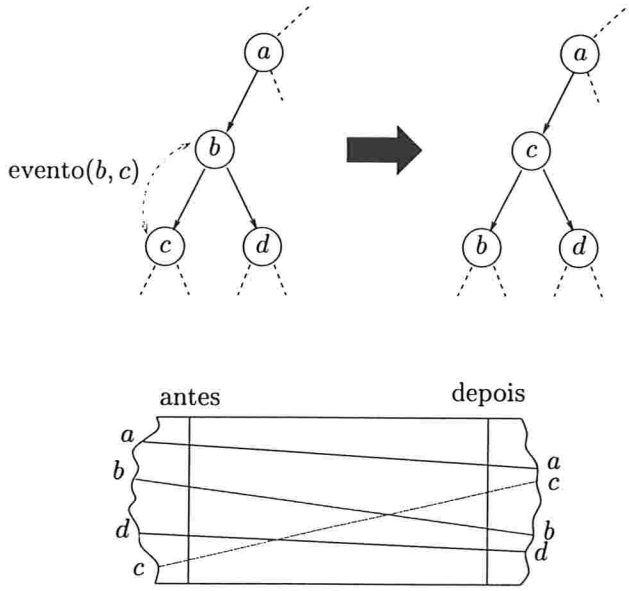


Figura 4.7: Processamento de um evento na $\mathbb{EDC}$ Heap cinético.

Como podemos ver o processamento de um evento gasta tempo $O(\log n)$, assim a $\mathbb{EDC}$ Heap cinético é de resposta rápida. Além disso, para cada ponto existem no máximo três certificados que dependem dele. Logo, esta $\mathbb{EDC}$ é local. Como o número de arestas no

heap é $n-1$, que é igual ao número de certificados na fila de prioridade, a estrutura Heap cinético também é compacta.

A questão agora é se ela é eficiente, ou seja, quantos eventos esta estrutura Heap cinético tem que processar no pior caso. Para decidir se uma $\mathbb{EDC}$ é eficiente ou não, necessitamos, antes de mais nada, do número de eventos externos, no pior caso, para o problema tratado. Os Corolários 2.3 e 2.5 estabelecem que no pior caso o número de eventos externos para o Problema do máximo, quando cada dois pontos se encontram um número limitado de vezes, é $O(n \log^* n)$.

Para a situação em que os pontos se movem a uma velocidade constante o número de eventos processados é $O(n \log^2 n)$ (cf. [15]). Logo, o Heap cinético para o Problema do máximo com pontos se movendo a uma velocidade constante é uma $\mathbb{EDC}$ eficiente (já que a razão entre o número total de eventos processados no pior caso e o número de eventos externos no pior caso é $O(\log^2 n)$) e portanto ótima.

4.3.3 Torneio cinético

A terceira e última estrutura de dados cinética que consideraremos é o chamado *Torneio cinético*. Esta estrutura é proveniente da cinetização de uma árvore de decisão para determinar o máximo de um conjunto de valores. Um algoritmo estático para o Problema do máximo particiona recursivamente o conjunto de pontos em subconjuntos de tamanhos aproximadamente iguais, calcula o máximo de cada subconjunto e devolve maior deles. Chamaremos este algoritmo, que esta descrito logo abaixo, de Algoritmo *MaximoRecursivo*. Denotamos por y_p a ordenada de um ponto p.

Algoritmo MaximoRecursivo. Recebe um conjunto S de n pontos no eixo das ordenadas e devolve um ponto de S com valor máximo.

Alternativa I $|S| = 1$

Devolva o único ponto de S e pare.

Alternativa II $|S| > 1$

Particione o conjunto S em dois subconjuntos, S_e e S_d, de tamanhos aproximadamente iguais.

Seja p e q o ponto máximo de S_e e S_d, respectivamente, ou seja:

$p = \text{MaximoRecursivo}(S_e)$

$q = \text{MaximoRecursivo}(S_d)$

Caso 1 $y_p \geq y_q$

Devolva o ponto p e pare.

Caso 2 $y_p < y_q$

Devolva o ponto q e pare.

A execução do Algoritmo MaximoRecursivo, para determinar o ponto de maior valor, gera uma árvore de decisão, a qual chamaremos de *árvore de torneio*. Este nome se deve ao fato de que, vista de baixo para cima, a árvore da decisão forma um torneio para determinar um campeão (o maior elemento) e cada comparação pode ser considerada como sendo uma partida onde um ponto é o vencedor e o outro ponto é o perdedor. Seja T uma árvore de torneio. Cada nó da árvore de torneio faz referência a um ponto de S. Nas folhas teremos referências a todos os pontos de S. Cada nó interno v de T fará referência ao ponto de maior valor entre os dois pontos referenciados pelos nós filhos de v. A Figura 4.8 ilustra uma árvore de torneio associada a oito pontos. Notemos que uma árvore de torneio tem profundidade $O(\log n)$.

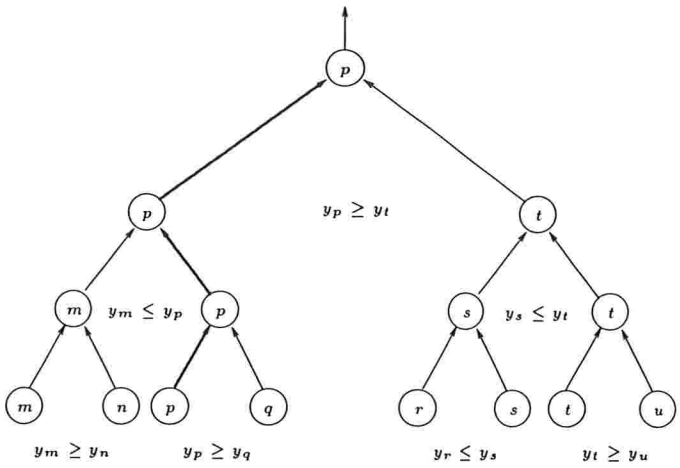


Figura 4.8: Árvore de torneio para o Problema do máximo. Cada nó interno da árvore guarda o ponto de maior valor entre os pontos armazenados nos seus filhos.

Os certificados na $\mathbb{EDC}$ Torneio cinético serão as comparações feitas no Caso 1 ou no Caso 2. Como é habitual, este certificados serão armazenados em uma fila de prioridade de acordo a seu prazo de validade. Esta $\mathbb{EDC}$ manterá ainda como estrutura de dados auxiliar a árvore do torneio. A fim de completarmos a estrutura Torneio cinético apresentaremos a seguir o algoritmo que processa um evento.

Algoritmo ArvRecursão. Recebe como entrada a estrutura Torneio cinético e a violação do certificado $y_p \geq y_q$ que envolve os pontos p e q, e processa tal

evento (isto é, atualiza o Torneio cinético).

A estrutura Torneio cinético consiste de um conjunto S de n pontos, dados através dos seus planos de vôo, uma fila de prioridade contendo os certificados por ordem de prazo de validade, e uma árvore de torneio T. Cada evento na fila de eventos contém referências para os dois nós da árvore de torneio T que contém os pontos associados à partida que originou o evento. O processamento do evento resultante da violação do certificado $y_p \geq y_q$ consistirá em iterações. Antes do início da primeira iteração removemos da fila de prioridade o evento associado ao certificado $y_p \geq y_q$ e inserimos o novo certificado $y_p \leq y_q$. Cada iteração começa com um nó `temp` da árvore T. No início da primeira iteração `temp` é o nó pai de q (q é o nó de T correspondente a 'partida' $y_p \geq y_q$).

Alternativa I `temp` $= p$

Caso 1 `temp` não é raiz de T

Seja w o nó irmão de `temp` em T, isto é, o nó que tem o mesmo pai que `tmp`.
Remova da fila de prioridade o evento associado ao certificado entre `tmp` e w.
Insira na fila de prioridade o certificado entre q e w.
Troque o ponto referenciado por `tmp` pelo ponto q.
Seja `tmp'` o nó pai de `tmp`.
Comece nova iteração com `tmp'` no papel de `tmp`.

Caso 2 `temp` é raiz de T

Troque o ponto referenciado por `tmp` pelo ponto q.
Devolva o Torneio cinético e pare.

Alternativa II `temp` $\neq p$

Devolva o Torneio cinético e pare.

Análise do Algoritmo ArvRecursão

Quando um certificado perde a sua validade temos a inversão entre o vencedor e perdedor de uma das partidas. Neste caso os certificados correspondentes aos nós da árvore do torneio, a partir do nó onde ocorreu a inversão, devem ser, um após outro, removidos da fila de certificados, e um novo certificado de uma partida envolvendo o novo vencedor deve ser calculado e inserido na fila, até que este vencedor perca uma partida do torneio (ou seja considerado o novo campeão). Como a árvore de torneio é balanceada, teremos que no máximo $O(\log n)$ certificados deverão ser removidos da fila, calculados e inseridos na fila,

que leva tempo $O(\log^2 n)$, assim a $\mathbb{EDC}$ é de resposta rápida. O ponto máximo é o vencedor em cada uma das partidas, desde o nível mais baixo até a raiz da árvore de torneio. Logo, a $\mathbb{EDC}$ é local, já que, no pior caso, existem $O(\log n)$ certificados que dependem de um único ponto. A árvore de torneio tem no máximo $O(n)$ nós, e para cada par de nós irmãos existe um evento na fila de prioridade. Portanto, a $\mathbb{EDC}$ é compacta.

O número de eventos totais, para o caso de um k-movimento, é $O(\lambda_k(n) \log n)$, onde $\lambda_k(n)$ é o comprimento de uma seqüência de Davenport-Schinzel de ordem k sobre um alfabeto com n símbolos (veja Capítulo 2). Seja $C(n)$ o número de eventos totais processados pela $\mathbb{EDC}$ para manter o ponto máximo de n pontos movendose sobre o eixo das ordenadas. Devido a recursividade da árvore de torneio, esta quantidade é limitada superiormente por $2C(n/2) + Q(n)$, onde $Q(n)$ é o número de eventos processados no pior caso para manter o ponto máximo obtido através da comparação de dois pontos. Cada um destes pontos muda $O(\lambda_k(n))$ vezes ao longo do tempo, assim o valor de $Q(n)$ é limitado por $O(\lambda_k(n))$. Então o número de eventos totais é $O(\lambda_k(n) \log n)$. Logo, a $\mathbb{EDC}$ Torneio cinético, para pontos que se movimentam de acordo com um k-movimento, é eficiente e portanto ótima.

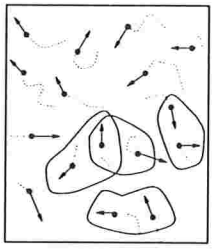
***

Atallah mostra que o comprimento da seqüência de pontos de maior ordenada, é no máximo $\lambda_k(n)$ para um conjunto de n pontos, realizando um k-movimento sobre o eixo das ordenadas. Kahan mostrou que o Algoritmo MaximoDeTempoReal, faz no máximo uma atualização a mais que um algoritmo sortudo. Estes são resultados interessantes do ponto de vista teórico, mas não são apropriados para implementação.

Nesta dissertação, consideramos três estruturas de dados cinética para o problema do máximo, apresentadas por Basch, Guibas e Hershberger, a saber, Sort cinético, Heap cinético e Torneio cinético. A $\mathbb{EDC}$ Torneio cinético, resulta ser ótima e ganha do Heap cinético já que processa um número menor de eventos.

CAPÍTULO 5

Problema do par mais-próximo



Trataremos neste capítulo do Problema do par mais-próximo. A versão estática deste problema consiste em: *dados*: um conjunto S de n pontos; *encontrar*: um par de pontos em S com a menor distância. A versão estática pode ser resolvida em tempo ótimo $O(n \log n)$ através de um algoritmo do tipo divisão-e-conquista (veja o Capítulo 5 de [36]).

No modelo off-line, Atallah [9, 10] mostrou que, para pontos realizando um k-movimento em $\mathbb{R}^d$, o comprimento de uma seqüência dos pares de pontos mais-próximos tem comprimento $O(n^2 \log^* n)$ e que uma tal seqüência pode ser computada em tempo $O(n^2 \log n \log^* n)$. Na Seção 5.1 apresentaremos estes resultados.

Na Seção 5.2 trataremos do Problema do par mais-próximo no modelo de tempo-real. Neste modelo é necessária uma estratégia para atualização das posições dos pontos. Esta estratégia deve basear-se no estado do conhecimento do problema, ou seja, no conjunto de regiões livres dos pontos. O que se deseja de um algoritmo de tempo-real é que retorne, o mais rápido possível, um par de pontos mais-próximos no instante em que a consulta foi feita, e que para isto atualize a posição de um número pequeno de pontos. Não importa qual seja a estratégia de atualização utilizada, quando cada consulta deve ser respondida em um prazo limite, erros são inevitáveis, considere a situação em que todos os pontos estão muito próximos. Kahan [29] propôs algumas estratégias de atualização e discutiu os resultados de simulações que utilizaram estas estratégias.

Os algoritmos estáticos existentes até recentemente para o problema do par mais-próximo (como o algoritmo ótimo clássico) não são apropriados para cinetização pois a estrutura de certificados proveniente destes algoritmos é intrinsicamente não-local. Basch, Guibas e Hershberger [13, 14] desenvolveram um algoritmo estático ótimo para o problema estático do par mais-próximo que é fácil de ser convertido para a versão cinética. Na Seção 5.3 apresentaremos este algoritmo estático ótimo. Finalmente, na Seção 5.4, descre-

veremos a cinetização do algoritmo da seção anterior, ou seja, veremos a estrutura de dados cinética de Basch, Guibas e Hershberger [13, 14] obtida a partir dos certificados provenientes deste algoritmo estático. Mostraremos ainda que esta estrutura de dados cinética é de resposta rápida, local, compacta e eficiente.

5.1 Par mais-próximo off-line

O *problema do par mais-próximo*, no modelo off-line, consiste em *dado:* um conjunto S de pontos em $\mathbb{R}^d$ realizando um k-movimento; *determinar*: uma seqüência de pares de pontos mais-próximos de S ao longo do tempo.

Uma *seqüência de pares mais-próximos* de S é uma seqüência formada por pares de pontos de S. Na seqüência, os pares de pontos são listados na ordem cronológica em que eles aparecem como um par mais-próximo. Assim, o primeiro par de pontos na seqüência é um par mais-próximo no instante inicial $t = t_0$ e o último par de pontos é o par mais-próximo no instante $t = +\infty$.

Os dois teoremas a seguir estabelecem que, para um conjunto de n pontos realizando um k-movimento contínuo ou um k-movimento com um número constante de descontinuidades e transições, o maior comprimento de uma seqüência de pares mais-próximos é $O(n^2 \log^* n)$.

Se p é um ponto denotaremos por $p(t)$ a equação que descreve o seu movimento. Por $\lambda_s(n)$ denotamos o comprimento da maior seqüência de Davenport-Schinzel de ordem s sobre um alfabeto com n símbolos (veja Capítulo 2).

Teorema 5.1 (Atallah [9, 10]) *Seja D um intervalo de $\mathbb{R}$ e S um conjunto de n pontos $p_1, \ldots, p_n$ em $\mathbb{R}^d$ realizando um k-movimento sobre D, então o comprimento máximo da seqüência de pares de pontos mais-próximo é $\lambda_{2k}(n(n-1)/2)$. Em particular para $k = 1$ temos que o comprimento é $O(n^2)$ e para $k > 1$ o comprimento é $O(n^2 \log^* n)$.*

Prova. A seqüência de pares de pontos mais-próximos pode ser obtida através do cálculo do envelope inferior das distâncias entre cada par de pontos distintos. Seja f_{ij} a função que representa o quadrado da distância entre o par de pontos p_i, p_j distintos ($1 \leq i < j \leq n$), ou seja:

$$f_{ij}(t) = |p_i(t) - p_j(t)|^2$$

As $m = n(n-1)/2$ funções f_{ij} satisfazem as hipóteses do Corolário 2.3 para $s = 2k$, ou seja, cada par de funções distintas se interceptam no máximo em $2k$ vezes no intervalo D.

Portanto o comprimento máximo da seqüência de pares de pontos mais-próximo é $\lambda_{2k}(m)$. Se $k = 1$ então $\lambda_2(m) = 2m - 1 = O(n^2)$. Se $k > 1$ então $\lambda_{2k}(m) = O(m \log^* m) = O(n^2 \log^* n)$. ∎

Teorema 5.2 (Atallah [9, 10]) *Seja D um intervalo de $\mathbb{R}$ e S um conjunto de n pontos $p_1, \ldots, p_n$ em $\mathbb{R}^d$ realizando um k-movimento sobre D, exceto em d pontos de descontinuidade e q pontos de transição, então o comprimento máximo da seqüência de pares de pontos mais-próximo é $\lambda_{2k+4d+4q}(n(n-1)/2)$. Em particular, para d e q limitados temos que este comprimento máximo é $O(n^2 \log^* n)$.*

Prova. A prova é similar ao Teorema 5.1. No presente caso as funções f_{ij} tem no máximo $2d$ pontos de descontinuidades e $2q$ pontos de transições. Cada par de funções distintas se interceptam no máximo em $2k$ pontos de D. Logo, o teorema segue do Corolário 2.5. ∎

O seguinte algoritmo calcula a seqüência de pares de pontos mais-próximo sobre um conjunto de pontos em $\mathbb{R}^d$ que realizam um k-movimento contínuo. O mesmo algoritmo também pode ser usado sobre um conjunto de pontos realizando um k-movimento com descontinuidades e transições.

> **Algoritmo ParMaisProximoOff-line.** Recebe um intervalo D em $\mathbb{R}$ e um conjunto S de n pontos $p_1, \ldots, p_n$ em $\mathbb{R}^d$ realizando um k-movimento sobre D, e devolve a seqüência de pares de pontos mais-próximo de S.

Passo 1 Seja C_S o conjunto das $m = n(n-1)/2$ funções que representam o quadrado da distâncias, ao longo do tempo, entre cada par de pontos p_i e p_j distintos, ou seja:

$$C_S = \{f_{ij} : f_{ij}(t) = |p_i(t) - p_j(t)|^2,\ 1 \leq i < j \leq n\}$$

Passo 2 Calcule a seqüência L do envelope inferior de C_S.

Passo 3 Devolva L.

Análise do Algoritmo ParMaisProximoOff-line

A obtenção, no Passo 1, das m funções em C_S pode ser feito em $O(m) = O(n^2)$, pois k é uma constante. Se as funções em C_S estão totalmente definidas no intervalo D, então pelo Teorema 5.1 o comprimento da seqüência L é no máximo $\lambda_{2k}(m)$, e o cálculo do envelope inferior pode ser feito em tempo $O(\lambda_{2k}(m) \log m)$. Para $k = 1$ temos que a complexidade de tempo do algoritmo é $O(n^2 \log n)$ e para $k > 1$ a complexidade é $O(n^2 \log n \log^* n)$.

Se as funções em C_S estão parcialmente definidas no intervalo D, ou seja, existem pontos de descontinuidade e transição, então pelo Teorema 5.2 o comprimento da seqüência L é no máximo $O(n^2 \log^* n)$, e o cálculo do envelope inferior leva tempo $O(n^2 \log n \log^* n)$.

5.2 Par mais-próximo em tempo-real

O *problema do par mais-próximo*, no modelo de tempo-real, consiste em *dados*: um conjunto S de n pontos em movimento contínuo; para cada ponto de S um limite superior sobre sua velocidade; um oráculo que atualiza a posição de um ponto dado; e uma seqüência de consultas do tipo "Qual é o par mais-próximo?" que são fornecidas on-line; *responder em tempo-real*: cada consulta um após outra.

Lembremos que a região livre de um ponto p é a menor região do espaço que sabemos com certeza conter o ponto p. Assim, no momento em que a posição de um ponto é atualizada a sua região livre se restringe a um único ponto. O conjunto de todas as regiões livres é o estado do conhecimento do problema. Algoritmo de tempo-real mantém o estado do conhecimento

Um algoritmo de tempo-real para o problema do par mais-próximo necessita atualizar a posição de alguns pontos a fim de responder a uma consulta. A atualização da posição de um ponto consiste de uma chamada ao oráculo. Devido ao tempo gasto em atualizações a resposta fornecida pode não ser o par mais-próximo corrente. A resposta fornecida pelo algoritmo será o par mais-próximo no instante no qual foi feita a consulta.

Um algoritmo de tempo-real será tão mais eficiente quanto menor for o número de atualizações por consulta, de tal forma que o par mais-próximo retornado coincida com o par mais-próximo corrente.

Um exemplo está ilustrado na Figura 5.1. Na figura temos dois intervalos $[t_0^*, t_1^*]$ e $[t_1^*, t_2^*]$ onde o par mais-próximo não se altera. Se uma consulta é feita no instante t_0 e a resposta é dada no instante t_1, então o par mais-próximo devolvido coincide com o par mais-próximo no instante t_1. Já, se uma consulta é feita no instante t_2 e a resposta é dada no instante t_3 então o par mais-próximo retornado não é o par mais-próximo no instante t_3.

Desejamos obter algoritmos que, baseados no estado do conhecimento do problema, atualizem as posições de um número mínimo de pontos a fim de que o valor da consulta seja o par mais-próximo corrente. Não obstante, erros são inevitáveis no pior caso, quando, por exemplo, todos os pontos estão próximos.

Seja $p_i(\hat{t})$ a posição do ponto p_i no instante $\hat{t}$ e r_i o limite superior para sua velocidade, então a região livre para o ponto p_i no instante $\tilde{t}$ ($\tilde{t} > \hat{t}$) é a esfera com centro em $p_i(\hat{t})$ e

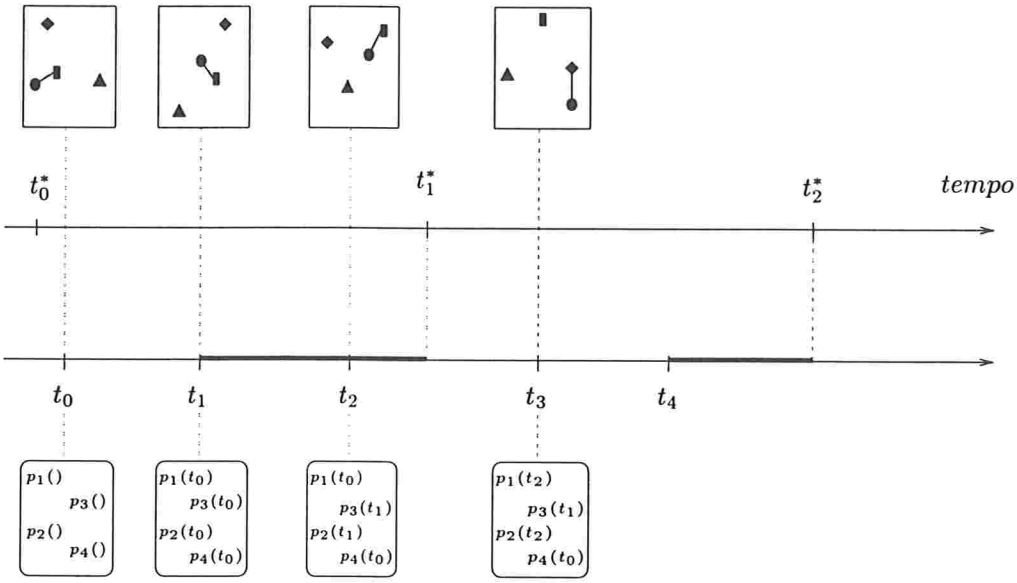


Figura 5.1: Par mais-próximo de tempo-real.

raio $r_i(\tilde{t} - \hat{t})$. Assim posições passadas, ou seja o estado do conhecimento, nos fornecem posições atuais aproximadas.

Um primeiro algoritmo para resolver o problema poderia simplesmente atualizar as posições dos pontos que foram atualizadas há mais tempo. Esta estratégia resultaria em um algoritmo n-fator-sortudo, ou seja, um algoritmo que pode fazer até n vezes o número de atualizações feitas por um algoritmo sortudo (algoritmo não-determinístico que para responder a qualquer consulta realiza um número mínimo de atualizações). De fato, basta considerar a situação em que o par mais-próximo corrente foram os últimos pontos a serem atualizados e a distância entre eles é menor que a distância entre as regiões livres de qualquer outro par de pontos. Neste situação um algoritmo sortudo faria apenas duas atualizações enquanto um algoritmo que implementa a estratégia acima faria n.

Outro algoritmo do tipo guloso, que também é n-fator-sortudo, seleciona um par de pontos cujas regiões livres são as mais próximas e atualiza a posição de um dos pontos, o processo se repete até que o par mais-próximo seja determinado.

A seguir descrevemos um algoritmo 2-fator-sortudo ótimo, isto é o algoritmo é 2-fator-sortudo e não existe um algoritmo k-fator-sortudo com $k < 2$ (cf. Kahan [29]). Este algoritmo, o qual chamaremos de *Algoritmo Pares* é similar ao algoritmo guloso, com a diferença que o Algoritmo Pares atualiza a posição dos dois pontos.

Algoritmo Pares. Recebe um conjunto S de n pontos, o estado do conheci-

mento do problema, o instante t onde é feita a consulta, um oráculo; e devolve o par mais-próximo para o instante t.

Cada iteração começa com dois pares de pontos (p^*, q^*) e (p, q). Para o início da primeira iteração tome para os papéis de p^* e q^* os pontos de S cuja distância entre as regiões livres é mínima. Atualize as posições de p^* e q^*. Sejam agora p e q os pontos de S cuja distância entre as regiões livres é mínima. (Notemos que o estado do conhecimento já foi alterado, pois as posição de p^* e q^* foram atualizadas.) Cada iteração consiste no seguinte:

Alternativa I A distância entre p^* e q^* é maior que a distância entre as regiões livres de p e q.

Se a posição do ponto p, no instante t, ainda não foi atualizada então atualize-a.
Se a posição do ponto q, no instante t, ainda não foi atualizada então atualize-a.

Caso 1 A distância entre p^* e q^* é maior que a distância entre os pontos p e q.
Sejam p' e q' os pontos p e q, respectivamente.
Sejam p'' e q'' pontos de S cuja distância entre as regiões livres é mínima.
Comece nova iteração com p', q', p'' e q'' nos papéis de p^*, q^*, p e q, respectivamente.

Caso 2 A distância entre p^* e q^* é menor ou igual que a distância entre p e q.
Sejam p' e q' os pontos p^* e q^*, respectivamente.
Seja p'' e q'' pontos de S cuja distância entre as regiões livres é mínima.
Comece nova iteração com p', q', p'' e q'' nos papéis de p^*, q^*, p e q, respectivamente.

Alternativa II A distância entre p^* e q^* é menor ou igual a distância entre as regiões livres de p e q.

Devolva p^* e q^* e pare.

Análise do Algoritmo Pares

Para encontrar um par de pontos de S cuja distância entre as regiões livres é mínima precisamos resolver o seguinte problema: *dado*: um conjunto de n esferas; *encontrar*: o par de esferas mais-próxima. Este problema pode ser resolvido em tempo $O(n \log n)$ através de uma variação do algoritmo estático ótimo para o problema da par mais-próximo. Se denotarmos por u o número de atualizações feitas pelo Algoritmo Pares para responder a uma consulta temos que a complexidade de tempo total do algoritmo é $O(un \log n)$.

5.3 Algoritmo estático cinetizável

Na próxima seção descrevemos um estrutura de dados cinética para o problema do par mais-próximo. A estrutura que veremos é proveniente da cinetização do algoritmo estático ótimo que apresentaremos nesta seção.

Os algoritmos estáticos existentes até recentemente para o problema do par mais-próximo não são apropriados para cinetização. Basch, Guibas e Hershberger [13, 14] desenvolveram um algoritmo estático ótimo fácil de ser cinetizado. A estrutura de dados cinética obtida a partir deste algoritmo é de resposta rápida, local, compacta e eficiente.

Um exemplo de um algoritmo estático ótimo para o problema do par mais-próximo, que não é apropriado para cinetização, é o algoritmo clássico de Shamos [36]. O algoritmo consiste no seguinte. Dado um conjunto S de n pontos no plano o algoritmo particiona S através de uma reta vertical $x = x_0$ em dois subconjuntos S_e e S_d com aproximadamente o mesmo número de pontos, como ilustrado na Figura 5.2. Calcule recursivamente a distância δ_e e δ_d entre o par mais-próximo em S_e e S_d, respectivamente. Seja $\delta = \min\{\delta_e, \delta_d\}$. Examine todos os pares de pontos que estão dentro da faixa vertical delimitada pelas retas $x_0 - \delta \leq x \leq x_0 + \delta$ para decidir qual é o par mais-próximo.

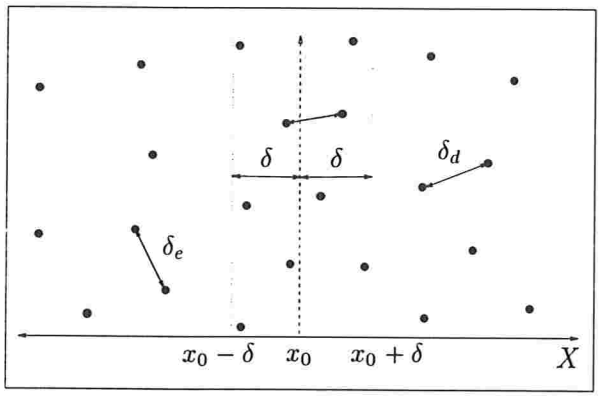


Figura 5.2: Algoritmo de Divisão e Conquista para o par-mais próximo

A versão cinética do algoritmo de Shamos requer certificados para manter a divisão que é feita, em cada nível da recursão, sobre o conjunto de pontos, certificados para os pontos examinados dentro da faixa vertical, certificados para os pontos à esquerda de $x = x_0 - \delta$ e à direita de $x = x_0 + \delta$, mais um certificado para garantir o mínimo entre δ_e e δ_d. Então um dos certificados para os pontos p do lado esquerdo da vertical $x = x_0 - \delta$ é do tipo

$$x_p < x_0 - \delta$$

Quando o certificado que determina o mínimo δ entre δ_e e δ_d é violado temos que

atualizar todos os certificados que envolvem δ podendo este número de certificados ser $O(n)$. De fato, consideremos a situação onde o evento acontece no nível mais profundo da recursão e o novo mínimo entre δ_e e δ_d resultante é a distância do novo par mais-próximo, nesta situação todos os certificados do tipo $x_p < x_0 - \delta$ deverão ser atualizados. Logo, a estrutura de dados cinética resultante não é de resposta rápida.

Descreveremos a seguir o algoritmo estático ótimo proposto por Basch, Guibas e Hershberger [13, 14] para calcular o par mais-próximo de um conjunto de pontos no plano.

Algoritmo estático ótimo

Para cada ponto p particionemos o semi-plano à direita de p em três cones tendo p como vértice (veja Figura 5.3). Veremos que, se esta partição for feita em três cones semelhantes, como os ilustrados na figura, então através de um algoritmo do tipo linha de varredura podemos determinar em tempo $O(n \log n)$ uma lista contendo $O(n)$ pares de pontos tal que entre estes pares se encontra um par de pontos mais-próximo. Com esta lista em mãos podemos determinar em tempo linear um par mais-próximo.

Denotaremos por Cone(p) o cone com vértice p que é limitado por retas que formam um ângulo de $\pm 30°$ com o eixo X das abscissas. Diremos que o Cone(p) está sobre o eixo X e suporemos que o cone inclui a aresta superior, mas não a inferior. Denotaremos ainda por Circ(p, r) a circunferência de centro p e raio r, e por dist(p, q) a distância entre os pontos p e q.

Consideremos a partição do semi-plano a direita de p em três cones semelhantes como mostra a Figura 5.3. Denotaremos por X_1, Y_1 os eixos X, Y; X_2, Y_2 e X_3, Y_3 os eixos X, Y rodados de $+60°$ e $-60°$, respectivamente, como mostrado na Figura 5.3. A argumentação a seguir será feita em termos do cone sobre o eixo X_1. Os mesmos argumentos se aplicam aos outros dois cones através de uma simples rotação.

Se p e q são pontos então escreveremos $p \prec q$ se p está à esquerda de q, ou se p e q têm a mesma x-coordenada e p está abaixo de q. Ao longo desta seção denotaremos por S o conjunto de pontos no plano do qual pretendemos encontrar um par mais-próximo e por (a, b) um par mais-próximo de S. Podemos supor que $a \prec b$ e que $b \in$ Cone(a), caso contrário podemos considerar uma rotação conveniente do plano de $\pm 60°$ para garantirmos que $b \in$ Cone(a).

O algoritmo que determina um par mais-próximo (a, b) baseia-se nos dois lemas a seguir, que em conjunto estabelecem que o ponto b é o ponto mais a esquerda contido no Cone(a).

Lema 5.3 *Seja S um conjunto de pontos no plano e seja (a, b) um par de pontos mais-*

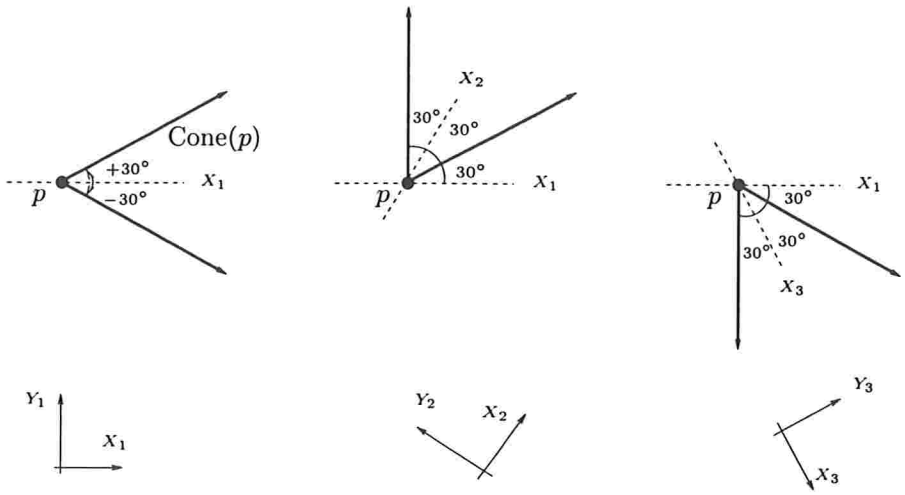


Figura 5.3: Divisão do plano nos três cones

próximo em S. Para todo ponto p de S com $a \prec p$ temos que b não está contido em Cone(p).

Prova. Seja r a distância entre a e b. Sejam c e d os pontos de intersecção entre Circ(a, r) com a aresta superior de Cone(a) e com a reta vertical que passa pelo ponto a, respectivamente. A reta L_1 que passa pelos pontos c e d é paralela a aresta inferior do Cone(a). Analogamente, defini-se a reta L_2 que é paralela a aresta superior do Cone(a), como ilustrado na Figura 5.4(a). Suponhamos que exista um ponto $p \in S$ com $a \prec p$ e tal que $b \in$ Cone(p). Um tal ponto deve pertencer a região R que é limitada pela reta vertical que passa pelo ponto a, e as retas que passam pelo ponto b e são paralelas a L_1 e L_2 (na Figura 5.4(a) esta região encontra-se sombreada). Como p está contido em R então dist$(a, p) <$ dist(a, b), contradição. ∎

Lema 5.4 *Seja S um conjunto de pontos no plano e (a, b) um par de pontos mais-próximo em S. Então o ponto b é o ponto mais à esquerda em Cone(a).*

Prova. Suponha que b não é o ponto mais à esquerda em Cone(a), seja então $\bar{b}$ um tal ponto mais à esquerda.

Denotemos por R_1 e R_2 as regiões ilustradas na Figura 5.4(b). Temos que $\bar{b} \in R_1 \cup R_2$ já que (a, b) é um par mais-próximo. Sejam q e p as intersecções da reta vertical que passa pelo ponto b com a aresta superior e a aresta inferior de Cone(a), respectivamente. Seja α o ângulo $\angle(b, a, p)$. Sem perda de generalidade podemos supor que dist$(b, q) \leq$ dist(p, b). Como os segmentos com um extremo b e perpendiculares às arestas de Cone(a) estão

dentro da circunferência $\mathrm{Circ}(a, \mathrm{dist}(a,b))$, então $\mathrm{dist}(b,q)$ e $\mathrm{dist}(b,p)$ é a maior distância de b a um ponto de R_1 e R_2, respectivamente. Pela lei de senos temos que $\mathrm{dist}(p,b) = \mathrm{dist}(a,b)$ sen α sen $60°$, com $\alpha < 60°$. Como sen $\alpha <$ sen $60°$, então $\mathrm{dist}(p,b) \leq \mathrm{dist}(a,b)$. Logo, $\mathrm{dist}(b,\bar{b}) < \mathrm{dist}(p,b) \leq \mathrm{dist}(a,b)$, pois $\bar{b} \in R_1 \cup R_2$, que é uma contradição. ∎

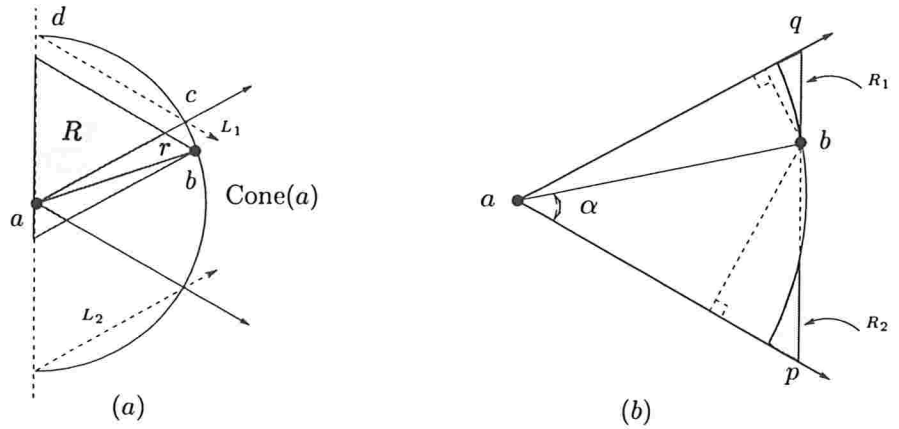


Figura 5.4: (a) O ponto b está contido em $\mathrm{Cone}(a)$. (b) O ponto b é o ponto mais à esquerda em $\mathrm{Cone}(a)$.

Necessitaremos de mais algumas definições a fim de descrevermos o algoritmo que determina a lista de pares de pontos (p,q) de S com $p \prec q$ e q o ponto mais à esquerda contido no $\mathrm{Cone}(p)$.

Denotaremos por $\mathrm{Visíveis}(p)$ o conjunto de pontos de S que estão na fronteira da união dos $\mathrm{Cone}(q)$ para todo ponto q à direita de p, ou seja

$$\mathrm{Visíveis}(p) = \{x \in S : x \in \partial(\bigcup_{\substack{q \in S \\ p \prec q}} \mathrm{Cone}(q))\}.$$

Denotaremos por $\mathrm{Cands}(p)$ o conjunto de pontos de $\mathrm{Visíveis}(p)$ que estão contidos no $\mathrm{Cone}(p)$, ou seja

$$\mathrm{Cands}(p) = \mathrm{Visíveis}(p) \cap \mathrm{Cone}(p),$$

e ainda denotaremos por $\mathrm{lcand}(p)$ o ponto mais à esquerda de $\mathrm{Cands}(p)$. Devido aos lemas anteriores temos que os pares de pontos do tipo $(p, \mathrm{lcand}(p))$ são candidatos a par mais-próximo.

Estas definições estão ilustradas na Figura 5.5 onde $\mathrm{Visíveis}(p) = \{p_1, p_2, p_3, p_4, p_5\}$, $\mathrm{Cands}(p) = \{p_2, p_3, p_4\}$ e $\mathrm{lcand}(p) = p_2$.

Os Lemas 5.3 e 5.4 estabelecem que $\mathrm{lcand}(a) = b$ para o par (a,b) mais-próximo do conjunto S. O seguinte algoritmo calcula uma lista de pares de pontos da forma $(p, \mathrm{lcand}(p))$

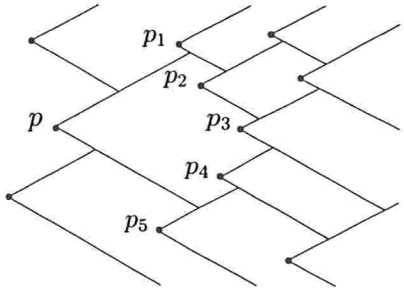


Figura 5.5: Visíveis, Cands e lcand de p.

para todos os pontos p de S. Observemos que cada par de pontos mais-próximo faz parte desta lista.

> **Algoritmo ParesDeCandidatos.** Recebe um conjunto S de n pontos no plano e devolve uma lista de pares de pontos da forma $(p, \text{lcand}(p))$ para todo ponto p em S.

Cada iteração começa com um subconjunto P dos pontos de S que ainda precisam ser examinados pelo algoritmo, um subconjunto M de S, uma lista L de pares de pontos e um ponto p que será examinado na iteração. A primeira iteração começa com $P = S$, $M = \emptyset$, $L = \emptyset$, e p sendo o maior ponto de S em relação à ordem $\prec$. Cada iteração consiste no seguinte:

Alternativa I $P \neq \emptyset$.

Seja Cands(p) o conjunto $M \cap$ Cone(p).

Caso 1 Cands(p) $\neq \emptyset$.

Seja lcand(p) o ponto de Cands(p) com menor x-coordenada e L' a lista L acrescida do par $(p, \text{lcand}(p))$.

Seja M' o conjunto $M - \text{Cands}(p) + p$.

Seja ainda P' o conjunto $P - p$ e seja p' o maior ponto de P' em relação à ordem $\prec$.

Comece nova iteração com P', M', L' e p' nos papéis de P, M, L e p.

Caso 2 Cands(p) $= \emptyset$.

Seja L' a lista L acrescida do par (p, p^∞), onde p^∞ denota um ponto no infinito[1].

Seja M' o conjunto $M + p$.

[1] Inserimos o par de pontos (p, p^∞) em L só para efeito da cinetização.

Seja P' o conjunto $P - p$ e seja p' o maior ponto de P' em relação à ordem $\prec$. Comece nova iteração com P', M', L' e p' nos papéis de P, M, L e p.

Alternativa II $P = \emptyset$.

Devolva L e pare.

Corretude do Algoritmo ParesDeCandidatos

Para entender como e porque o algoritmo funciona, basta observarmos que, no início de cada iteração, vale o seguinte invariante: M é igual a Visíveis(p). Este fato pode ser provado por indução no número de iterações.

Portanto, o Algoritmo ParesDeCandidadtos devolve uma lista de pares de pontos da forma $(p, \text{lcand}(p))$ para todo ponto p em S.

Análise do Algoritmo ParesDeCandidatos

Como em cada iteração removemos um ponto de P, então a Alternativa I será executada n vezes, uma para cada ponto de S.

Para cada ponto q em S existe um único ponto p em S tal que q pertence a Cands(p). Já que em cada iteração removemos Cands(p) de M, então a $\sum_{p \in S} |\text{Cands}(p)|$ é $O(n)$.

Observemos que os pontos de Cands(p) formam uma sub-seqüência consecutiva dos pontos de M ordenados conforme as suas y-coordenadas. Manteremos os pontos de M numa árvore binária balanceada T ordenados conforme as suas y-coordenadas. Assim, o conjunto Cands(p) pode ser calculado em tempo $O(|\text{Cands}(p)| + \log n)$. De fato, podemos encontrar, em tempo $O(\log n)$, os pontos de M com a maior e a menor y-coordenada contidos em Cone(p). Com estes pontos em mãos, podemos percorrer a sub-árvore de T contendo os pontos de Cands(p) em tempo $O(|\text{Cands}(p)|)$.

Dos parágrafos anteriores podemos concluir que o cálculo, ao longo da execução do algoritmo, de todos os conjuntos Cands pode ser feito em tempo $O(n \log n + \sum_{p \in S} |\text{Cands}(p)|)$, ou seja, em tempo $O(n \log n)$.

Encontrar lcand(p) leva tempo $O(|\text{Cands}(p)|)$. Logo, podemos determinar os pares $(p, \text{lcand}(p))$, para todo ponto p em S, em tempo total $O(n)$.

Remover os pontos de Cands(p) de M leva tempo $O(|\text{Cands}(p)| \log n)$. Portanto, o tempo total gasto com a execução de $M - \text{Cands}(p) + p$ é $O(n \log n)$.

Concluímos assim que a complexidade de tempo do Algoritmo ParesDeCandidatos é $O(n \log n)$.

Lembremos que todo par mais-próximo (a, b) satisfaz a igualdade $b = \text{lcand}(a)$ em S ou em uma das rotações $\pm 60°$ dos ponto de S. O seguinte algoritmo calcula um par de pontos mais-próximo baseado nesta observação.

Algoritmo ParMaisPróximo. Recebe um conjunto S de n pontos no plano e devolve um par de pontos cuja distância é mínima.

Passo 1 Sejam L_1, L_2 e L_3 as listas de pares de pontos devolvidas ao executarmos o Algoritmo ParesDeCandidatos sobre o conjunto S, S rotacionado em $+60°$ e S rotacionado em $-60°$, respectivamente.

Passo 2 Seja L a união das listas L_1, L_2 e L_3. Calcule a distância entre cada par de pontos na lista L.

Passo 3 Encontre e devolva um par de pontos em L mais-próximo.

A complexidade do Algoritmo ParMaisPróximo é determinada pelo Passo 1, o qual pode ser executada em tempo $O(n \log n)$. Os Passos 2 e 3 podem ser executados em tempo linear no número de pares em L, que é $O(n)$.

A seguir mostraremos a cinetização do algoritmo estático que acabamos de apresentar.

5.4 Cinetização do algoritmo estático

O *problema do par mais-próximo*, no modelo cinético, consiste em *dado*: um conjunto S de n pontos (no plano) em movimento contínuo; um plano de vôo para cada ponto, uma seqüência de alterações de planos de vôo que são fornecidas on-line; *manter*: a descrição combinatória do par mais-próximo de S ao longo do tempo.

Nesta seção descrevemos uma estrutura de dados cinética para o problema do par mais-próximo. A estrutura que mostraremos é devida Basch, Guibas e Hershberger [13, 14] e consiste na cinetização do Algoritmo ParMaisPróximo, da seção anterior.

A seguir apresentaremos a cinetização do Algoritmo ParesDeCandidatos, que mostra como o Passo 1 do Algoritmo ParMaisPróximo é cinetizado. Já a cinetização do Passo 2 e 3 do Algoritmo ParMaisPróximo equivale à cinetização do Problema do máximo, e para isto utilizaremos a estrutura Torneio cinético descrita no Capítulo 4, Seção 3.

Resumindo, a cinetização do algoritmo ParMaisPróximo será dividida em duas tarefas principais:

1 cinetização do Passo 1, ou seja, do algoritmo ParesDeCandidatos.

2 cinetização do Passo 2 e 3 através da estrutura Torneio cinético.

A cinetização do algoritmo ParesDeCandidatos manterá o "histórico" do algoritmo, para isso definimos o *Diagrama dos Cones* como sendo a união, sobre todo ponto p, da parte da fronteira de Cone(p) que não está contida em $\cup_{q \in \text{Visíveis}(p)} \text{Cone}(q)$.

Se os cones estão sobre o eixo X, então diremos que o Diagrama dos Cones está sobre o eixo X. O Diagrama dos Cones mantém as informações necessária para obtermos a lista de pares de candidatos. Além disso, o conjunto de pontos é particionado numa estrutura boa e simples para cinetização.

Também definimos alvo(p) como sendo o conjunto de pontos q de Visíveis(p) tal que uma aresta de Cone(p) intercepta Cone(q). Na Figura 5.6, para o Diagrama dos Cones sobre o eixo X_1, temos que os pontos q_1 e q_2 pertencem a alvo(p).

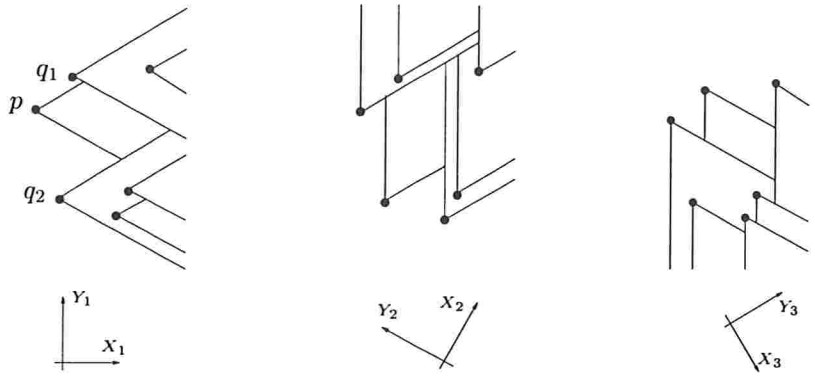


Figura 5.6: Diagrama dos Cones nos três eixos

Descreveremos a seguir os tipos de certificados para manter o Diagrama dos Cones sobre os eixos X_1, X_2 e X_3.

Certificados

Os certificados serão as x_1-, x_2- e x_3-ordem dos pontos. Surpreendentemente estes certificados serão suficientes para manter a descrição combinatória do Diagrama dos Cones em relação a cada um dos eixos X_1, X_2 e X_3. Inicialmente, os certificados serão obtidos através a varredura dos pontos em ordem decrescente em relação a $\prec$, durante o cálculo da lista de candidatos.

Notemos que, por exemplo, para o Diagrama dos Cones sobre o eixo X_1, a x_2-ordem dos pontos induz uma ordem nas arestas inferiores dos cones dos pontos (veja Figura 5.7).

Para tornar as idéias mais claras consideremos um exemplo. Para os pontos p, q e

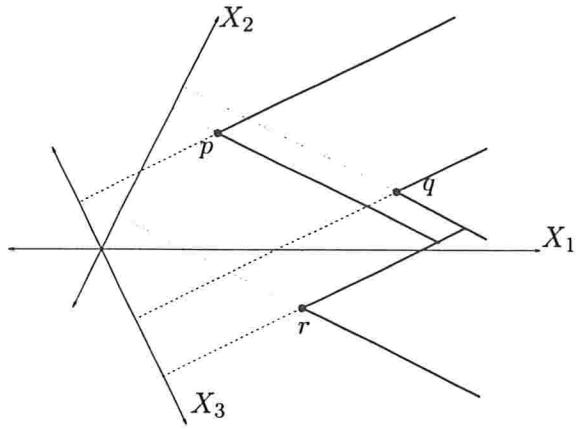


Figura 5.7: Diagrama dos Cones sobre o eixo X_1 e as projeções dos pontos sobre os eixos X_1, X_2, X_3.

r, mostrados na Figura 5.7, os seguintes seis certificados são suficientes para manter a descrição combinatória do Diagrama dos Cones de uma maneira implícita.

$$p <_{x_1} r \qquad r <_{x_1} q$$

$$r <_{x_2} p \qquad p <_{x_2} q$$

$$p <_{x_3} q \qquad q <_{x_3} r$$

Notemos que, para o Diagrama dos Cones sobre o eixo X_1, a x_2-ordem dos pontos induz uma ordem nas arestas inferiores dos cones dos pontos (veja Figura 5.7).

Em geral, para n pontos teremos $3(n-1)$ certificados que atestam as ordens dos pontos em relação a cada um dos três eixos.

Lema 5.5 *Sejam C_1 e C_2 configurações dos pontos de S em dois instantes. Se em C_1 e em C_2 os pontos de S têm as três ordens equivalentes então para cada $p \in S$, Visíveis(p), Cands(p) e lcand(p) sobre o eixo X_1 são os mesmos em C_1 e C_2.*

Prova. A prova é por indução sobre o número de pontos de S.

Seja $S = \{p_1, \ldots, p_n\}$. Por hipótese a $\prec$-ordem dos pontos de S em C_1 e em C_2 são iguais. Suponhamos que $p_1 \prec p_2 \ldots \prec p_n$.

Temos que p_n é ponto de S mais à direita em ambas as configurações C_1 e C_2. Logo, Visíveis(p_n), Cands(p_n) e lcand(p_n) são os mesmos nas duas configurações.

Suponhamos, por hipótese de indução, que os pontos $p_{i+1}, p_{i+2}, \ldots, p_n$, $1 < i < n$, têm os mesmos conjuntos de Visíveis, Cands e lcand na duas configurações. Mostraremos que

os pontos $p_i, p_{i+1}, p_{i+2}, \ldots, p_n$ têm também os mesmos conjuntos Visíveis, Cands e lcand na duas configurações. Para isto basta mostrarmos que os conjuntos Visíveis(p_i), Cands(p_i) e lcand(p_i) são os mesmos nas duas configurações.

O conjunto Visíveis(p_i) é o mesmo nas duas configurações, já que Visíveis(p_i) é igual a Visíveis(p_{i+1}) − Cands(p_{i+1}) + p_{i+1}.

O ponto p_i tem os mesmos alvos em cada configuração, pois as x_2-ordem e x_3-ordem restritas a p_i + Visíveis(p_i) serem as mesmas nas duas configurações. Logo Cands(p_i) deve ser o mesmo nas configurações C_1 e C_2. Finalmente, já que a x_1-ordem é a mesma nas duas configurações, lcand(p_i) também é o mesmo nas duas configurações. ∎

Observe que, do Lema 5.5, também concluímos que os conjuntos Visíveis, Cands e lcand são os mesmo sobre os eixos X_2 e X_3.

Definida a estrutura de certificados, passamos a descrever agora as três únicas estruturas de dados auxiliares e como um evento deve ser processado a fim de mantermos as estruturas auxiliares e a descrição combinatória de cada um dos três Diagramas dos Cones.

Estruturas de dados auxiliares para cinetização

Seja Pais(p) o conjunto dos pontos q de S tal que p pertence a alvo(q). As estruturas de dados auxiliares utilizadas são as seguintes:

- Cands(p): São os pontos da intersecção de Visíveis(p) e Cone(p). Armazenaremos Cands(p) como uma seqüência de pontos, ordenados de acordo com as suas y-coordenadas, em uma árvore balanceada de busca. Cada nó da árvore terá um apontador para seu pai; o nó raiz apontará para o ponto p. Além disso, cada nó da árvore tem um campo que mantém o ponto mais à esquerda nessa sub-árvore. Para a raiz esse campo contém o ponto lcand(p).

- Pais$_a$(p) e Pais$_b$(p): São os pontos de Pais(p) particionados no conjunto dos pontos que estão acima e embaixo de p, respectivamente, ou seja,

$$\text{Pais}_a(p) = \{q \in Pais(p) : p <_y q\}$$

$$\text{Pais}_b(p) = \{q \in Pais(p) : q <_y p\}$$

  Os conjuntos Pais$_a$(p) e Pais$_b$(p) serão armazenados numa árvore balanceada de busca, com os pontos ordenados de acordo com as suas x_2-coordenada e x_3-coordenadas, respectivamente. Cada nó da árvore contém ainda um apontador para o seu pai; o nó raiz aponta para o ponto p.

Para cada ponto p de S teremos referências para os nós das árvores Cands(), $\text{Pais}_a()$, $\text{Pais}_b()$ que representam o ponto p (três referências são necessárias). Se p não pertence a Cands(v) para qualquer v em S então a respectiva referência de p será (digamos) NULL. O mesmo se aplica para os conjuntos $\text{Pais}_a()$ e $\text{Pais}_b()$.

Observemos que a ordem na qual são armazenados os pontos nas árvores $\text{Pais}_a()$ e $\text{Pais}_b()$ é relativa ao Diagrama dos Cones. Por exemplo, consideremos o Diagrama dos Cones sobre o eixo X_2. Para um ponto p, os conjuntos $\text{Pais}_a(p)$ e $\text{Pais}_b(p)$ serão armazenados numa árvore balanceada de busca, com os pontos ordenados de acordo com as suas x_3-coordenadas e x_1-coordenadas, respectivamente.

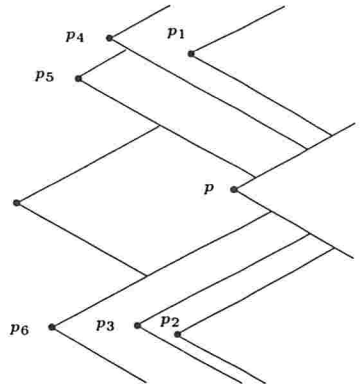


Figura 5.8: $\text{Pais}_a(p)$ e $\text{Pais}_b(p)$.

Na Figura 5.8 o conjunto $\text{Pais}_a(p) = \{p_1, p_4, p_5\}$ e $\text{Pais}_b(p) = \{p_2, p_3, p_6\}$. Observemos que a fronteira inferior de Cone(p_1), Cone(p_4) e Cone(p_5) induzem uma x_2-ordem dos pontos. Analogamente, a fronteira superior de Cone(p_2), Cone(p_3) e Cone(p_6) induzem uma x_3-ordem dos pontos.

Para determinar os conjuntos $\text{Pais}_a(q)$ e $\text{Pais}_b(q)$, para cada ponto q de S, basta fazermos duas buscas sobre o conjunto Visíveis do ponto que está sendo examinado durante uma iteração do algoritmo ParesDeCandidatos. Isto não aumentará a complexidade de tempo do algoritmo. Por exemplo seja p o ponto considerado numa iteração do algoritmo e seja q o ponto em Visíveis(p) com a menor y-coordenada e que q está acima da aresta superior de Cone(p). Nesta situação o ponto p deverá ser inserido em $\text{Pais}_b(q)$. Portanto, ao final do algoritmo teremos calculado os conjuntos $\text{Pais}_a(q)$ e $\text{Pais}_b(q)$, para todo ponto q de S.

Observemos que uma versão análoga do Lema 5.5 também é aplicável para os conjuntos $\text{Pais}_a()$ e $\text{Pais}_b()$. Por exemplo, se p pertence a $\text{Pais}_a(q)$ então q é alvo para p nas duas configurações, já que Visíveis(p) e a x_2-ordem são os mesmos nas duas configurações.

A fim de mantermos as três listas de candidatos do tipo $(p, \text{lcand}(p))$, manteremos a descrição combinatória dos três Diagramas dos Cones. Os seguintes algoritmos mostram como podemos manter implicitamente o Diagrama dos Cones que está sobre o eixo X_1, após processar a violação de um certificado. Na realidade, o que será mantido serão os conjuntos Cands, $\text{Pais}_a()$, $\text{Pais}_b()$ e as referencias lcand.

Processamento dos eventos

Temos três tipos de eventos a serem processados. Estes eventos corresponderão à perda de validade de um certificado envolvendo pares de pontos consecutivos em relação x_1-ordem ou x_2-ordem ou x_3-ordem. Como veremos, as mudanças produzidas no Diagrama dos Cones pelo processamento de um evento serão locais.

Quando um evento ocorre, ele deve ser processado, e a fila de prioridade, que contém os certificados em ordem de prazo de validade, deve ser atualizada. Por exemplo, consideremos quatro pontos consecutivos, segundo a x_1-ordem, r, p, q e s de tal forma que $r <_{x_1} p <_{x_1} q <_{x_1} s$. Temos um certificado associado a cada uma das relações $r <_{x_1} p$, $p <_{x_1} q$, e $q <_{x_1} s$.

Se p e q alteram sua x_1-ordem, ou seja, o certificado $p <_{x_1} q$ deixa de ser válido, devemos então processar o evento correspondente e além disso remover da fila de prioridade os certificados associados a $r <_{x_1} p$, $p <_{x_1} q$, e $q <_{x_1} s$. Finalmente devemos inserir na fila de prioridade os certificados correspondentes a $r <_{x_1} q$, $q <_{x_1} p$, e $p <_{x_1} s$.

Evento envolvendo a x_1-ordem: x_1-evento

Um x_1-evento é a violação da x_1-ordem de dois pontos consecutivos nesta ordem. O seguinte algoritmo mostra como processar um tal evento. A situação está ilustrada na Figura 5.9 para os pontos p e q. Notemos que o Alternativa I é análoga à Alternativa II.

> **Algoritmo ProcessaX_1Evento**(p, q). Recebe o Diagrama dos Cones sobre o eixo X_1, de um conjunto S de n pontos, um par de pontos p e q pertencentes a S, consecutivos segundo a x_1-ordem ($p <_{x_1} q$), e devolve o Diagrama dos Cones logo após processar o evento associado à perda de validade do certificado $p <_{x_1} q$.

Alternativa I $p \in \text{Pais}_b(q)$

> Remova $\text{Cands}(q) \cap \text{Cone}(p)$ de $\text{Cands}(q)$ e insira em $\text{Cands}(p)$.
> Seja $u \in S$ tal que $q \in \text{Pais}_a(u)$. Remova o ponto q de $\text{Pais}_a(u)$.

Insira o ponto q em $\text{Pais}_a(p)$ e remova o ponto p de $\text{Pais}_b(q)$.
Se $\text{Cands}(q)$ é não vazio seja v o ponto em $\text{Cands}(q)$ com a menor y-coordenada. Caso contrário, tome v como sendo o ponto tal que $q \in \text{Pais}_b(v)$. (Caso não exista um tal ponto v, este passo pode ser simplesmente ignorado.) Insira p em $\text{Pais}_b(v)$.
Se existe um ponto $\bar{p}$ tal que p e $q \in \text{Cands}(\bar{p})$ então atualize $\text{lcand}(\bar{p})$ na árvore para $\text{Cands}(\bar{p})$.

Alternativa II $p \in \text{Pais}_a(q)$

Remova $\text{Cands}(q) \cap \text{Cone}(p)$ de $\text{Cands}(q)$ e insira em $\text{Cands}(p)$.
Seja $u \in S$ tal que $q \in \text{Pais}_b(u)$. Remova o ponto q de $\text{Pais}_b(u)$.
Insira o ponto q em $\text{Pais}_b(p)$ e remova o ponto p de $\text{Pais}_a(q)$.
Se $\text{Cands}(q)$ é não vazio seja v o ponto em $\text{Cands}(q)$ com a maior y-coordenada. Caso contrário, tome v como sendo o ponto tal que $q \in \text{Pais}_a(v)$. (Caso não exista um tal ponto v, este passo pode ser simplesmente ignorado.) Insira p em $\text{Pais}_a(v)$.
Se existe um ponto $\bar{p}$ tal que p e $q \in \text{Cands}(\bar{p})$ então atualize $\text{lcand}(\bar{p})$ na árvore para $\text{Cands}(\bar{p})$.

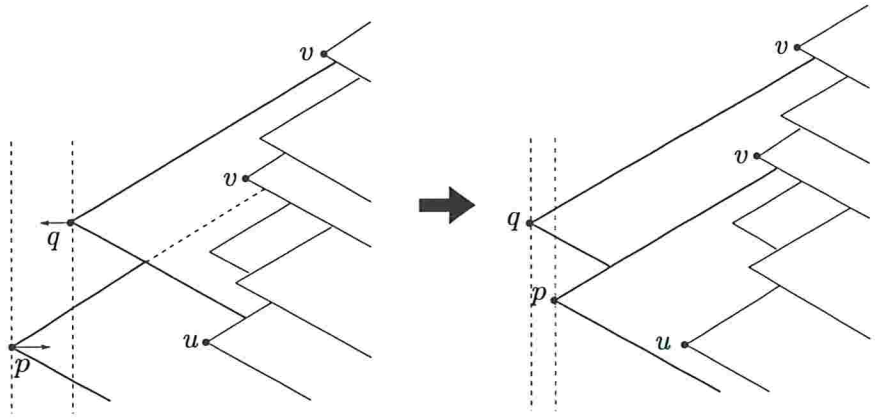


Figura 5.9: Processamento do evento quando p e q alteram sua x_1-ordem.

Corretude do Algoritmo ProcessaX_1Evento

O seguinte lema prova a corretude do algoritmo.

Lema 5.6 *Considere um evento que consiste na perda de validade de um certificado $p <_{x_1} q$, onde p e q são pontos consecutivos na x_1-ordem. Então, após aplicarmos o Algoritmo ProcessaX_1Evento(p, q), temos que os conjuntos Cands() e Pais() e as referências lcand() representam o Diagrama dos Cones sobre o eixo X_1 da configuração atual dos pontos de S.*

Prova. É suficiente provar o lema quando a Alternativa I é executada. O Diagrama dos Cones só muda se $q \in \text{alvo}(p)$. No caso em que $p \in \text{Pais}_b(q)$ os pontos de $\text{Cands}(q) \cap \text{Cone}(p)$ são transferidos de $\text{Cands}(q)$ para $\text{Cands}(p)$ (veja Figura 5.9). O ponto q passa a ser um novo elemento de $\text{Pais}_a(p)$. O ponto p pode passar a pertencer a $\text{Pais}_b(v)$, para algum ponto v tal que $v \in \text{Cands}(q)$ ou $q \in \text{Pais}_b(v)$. Não existe mudanças de alvo() para os pontos à esquerda e à direita de p e q. Se p e q pertencem a $\text{Cands}(\bar{p})$, para algum ponto $\bar{p}$, então devemos, se for o caso, atualizar o campo mais à esquerda na árvore que representa $\text{Cands}(\bar{p})$, que pode causar uma alteração de $\text{lcand}(\bar{p})$[2]. O Algoritmo $\text{Processa}X_1\text{Evento}(p, q)$ toma conta de todas as situações acima. ∎

Mudanças na x_1-ordem podem produzir alterações nos Diagramas dos Cones sobre os eixos X_2 e X_3. Por exemplo, consideremos um x_1-evento envolvendo pontos p e q, como ilustrado na Figura 5.10. Se $p \in \text{Pais}_b(q)$ (Alternativa I) então teremos o seguinte:

- no Diagrama dos Cones sobre o eixo X_2, se q pertence a $\text{Cands}(p)$ então q sai do $\text{Cone}(p)$.
- no Diagrama dos Cones sobre o eixo X_3, se q pertence a $\text{Pais}_a(p)$ então p entra no $\text{Cone}(q)$.

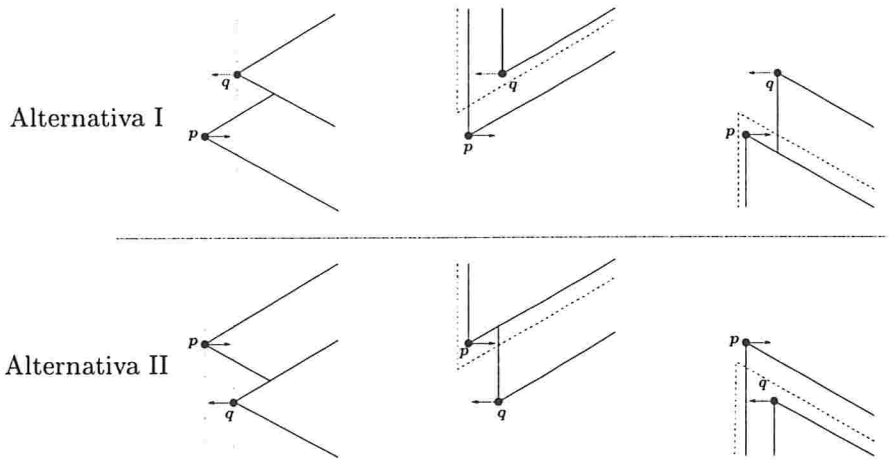


Figura 5.10: Ilustração das possíveis mudanças nos Diagramas dos Cones sobre os eixos X_2 e X_3 após um x_1-evento envolvendo os pontos p e q.

[2]Se $\text{lcand}(\bar{p})$ muda, a função $dist(\bar{p}, \text{lcand}(\bar{p}))$ também muda, o que representa uma mudança no plano de vôo de um ponto na estrutura de dados cinética para o cálculo do mínimo.

Análise do Algoritmo ProcessaX_1Evento

A complexidade do Algoritmo ProcessaX_1Evento é $O(\log n)$. Consideremos a Alternativa I, já que a analise da complexidade computacional da Alternativa II é semelhante a esta. O conjunto Cands(q) $\cap$ Cone(q) pode ser feito em tempo $O(\log n)$ através da operação de *split* de uma lista, armazenada como uma árvore balanceada de busca, em duas listas cuja concatenação é a lista original (cf. Knuth [31], páginas 474–475). Para atualizarmos Cands(q) e Cands(p) precisamos de um algoritmo de concatenação, que insere uma lista L_2 (digamos) à direita de uma lista L_1 sem destruir o balanço das representações. Esta concatenação também pode ser feita em tempo $O(\log n)$ (cf. Knuth [31], páginas 474–475).

As operações para atualizarmos os conjuntos Pais() consistem em simples buscas, inserções e remoções em árvores balanceadas de busca e podem ser executadas em tempo $O(\log n)$.

A atualização de lcand($\bar{p}$), ao final da alternativa, também pode ser feita em tempo $O(\log n)$. Primeiramente devemos percorrer, de baixo para cima, as árvores Cands() que contém os pontos p e q para decidirmos se estes pontos pertencem ao mesmo conjunto Cands(). Caso p e q não estejam contidos em um mesmo conjunto Cands() então não há nada a ser feito. Se p e q estão contidos em Cands($\bar{p}$), para algum ponto $\bar{p}$, devemos encontrar o primeiro ancestral comum w de p e q na árvore Cands($\bar{p}$), isto pode ser realizado em tempo $O(\log n)$. Agora, só resta atualizarmos as referências para o ponto mais à esquerda nas sub-árvores que tem como raiz os nós entre w e a raiz da árvore Cands($\bar{p}$), fazendo com que as eventuais referências a p sejam trocadas por referências a q.

Evento envolvendo a x_2-ordem: x_2-evento

Um x_2-evento é a violação da x_2-ordem entre dois pontos consecutivos nesta ordem. O seguinte algoritmo mostra como processar um tal evento. A situação está ilustrada na Figura 5.11 para os pontos p e q.

> **Algoritmo ProcessaX_2Evento**(p, q). Recebe o Diagrama dos Cones sobre o eixo X_1, de um conjunto S de n pontos, um par de pontos p e q pertencentes a S consecutivos segundo a x_2-ordem tal que $p \prec q$, e devolve o Diagrama dos Cones logo após processar o evento associado a inversão da x_2-ordem entre p e q.

Alternativa I $p \in \text{Pais}_a(q)$

Se existe $v \in S$ tal que $q \in$ Cands(v) então remova q de Cands(v).

Insira q em Cands(p) e remova p de $\text{Pais}_a(q)$.
Seja T o conjunto dos pontos em $\text{Pais}_b(q)$ à direita de p.

Caso I.1 $T \neq \emptyset$

Seja t o menor ponto de T em relação à x_3-ordem e M o conjunto dos pontos de $\text{Pais}_b(q)$ que são menores do que t em relação à x_3-ordem.
Remova os pontos em M de $\text{Pais}_b(q)$ e insira-os em $\text{Pais}_b(p)$.
Insira p em $\text{Pais}_a(t)$.

Caso I.2 $T = \emptyset$ e existe t tal que $q \in \text{Pais}_a(t)$

Remova os pontos de $\text{Pais}_b(q)$ e insera-os em $\text{Pais}_b(p)$.
Insira p em $\text{Pais}_a(t)$.

Caso I.3 $T = \emptyset$ e não existe t tal que $q \in \text{Pais}_a(t)$

Remova os pontos de $\text{Pais}_b(q)$ e insera-os em $\text{Pais}_b(p)$.

Alternativa II $q \in \text{Cands}(p)$

Se existe $t \in S$ tal que $p \in \text{Pais}_a(t)$ então remova p de $\text{Pais}_a(t)$.
Insira p em $\text{Pais}_a(q)$ e remova q de Cands(p).
Seja M o conjunto dos pontos de $\text{Pais}_b(p)$ que são maiores do que q em relação à x_3-ordem. Remova os pontos em M de $\text{Pais}_b(p)$ e insira-os em $\text{Pais}_b(q)$.
Se $\text{Pais}_b(p) \neq \emptyset$ então insira q em Cands(v), onde v é o maior ponto de $\text{Pais}_b(p)$ em relação à x_3-ordem. Se $\text{Pais}_b(p) = \emptyset$ e existe v tal que $p \in \text{Cands}(v)$ então insira q em Cands(v).

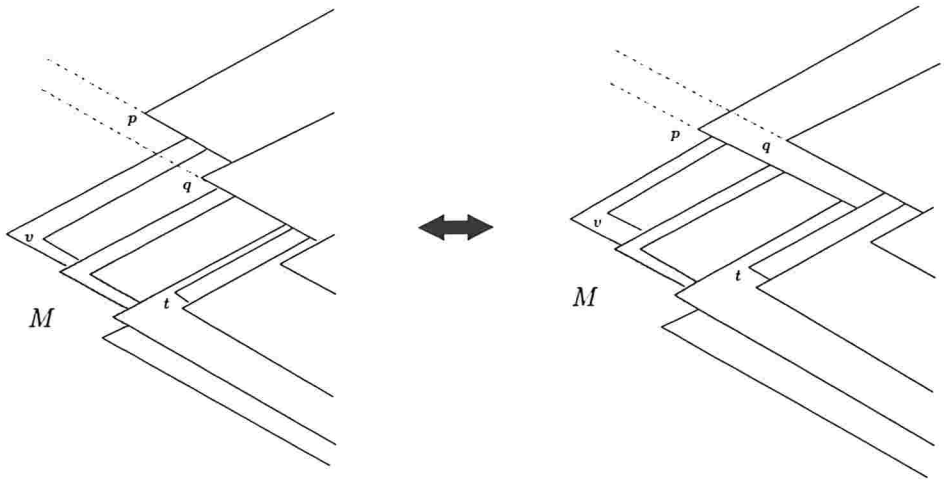


Figura 5.11: Processamento do evento quando p e q alteram sua x_2-ordem.

Corretude do Algoritmo ProcessaX_2Evento

O seguinte lema prova a corretude do algoritmo.

Lema 5.7 *Considere um evento que consiste na perda de validade de um certificado $p <_{x_2} q$ ou $q <_{x_2} p$, onde p e q são pontos consecutivos na x_2-ordem tal que $p \prec q$. Então, após aplicarmos o Algoritmo ProcessaX_2Evento(p, q), temos que os conjuntos Cands() e Pais() e as referências lcand() representam o Diagrama dos Cones sobre o eixo X_1 da configuração atual dos pontos de S.*

Prova. Suponhamos que q entra em Cone(p). Se p não pertence a Pais$_a(q)$ então q não pertence a Visíveis(p) e portanto a alteração da x_2-ordem entre p e q não afeta o Diagrama dos Cones sobre o eixo X_1. Se p pertence a Pais$_a(q)$ então a alteração da x_2-ordem entre p e q não muda o alvo dos pontos à direita de p (menores que p em relação à x_1-ordem). O alvo inferior de p precisa ser atualizado. Somente os pontos à esquerda de p que têm q como alvo superior precisam ter seu alvo alterado para p (veja a Figura 5.11). O ponto q entra em Cands(p) e deixa Cands(v), onde v é tal que q pertence a Cands(v) (caso exista um tal v).

No caso em que q sai de Cone(p) as mudanças são as 'inversas' das consideradas acima. O Algoritmo ProcessaX_2Evento(p, q) toma conta de todas as situações acima. ∎

Mudanças na x_2-ordem podem produzir alterações nos Diagramas dos Cones sobre os eixos X_2 e X_3, além das alterações já consideradas no Diagrama dos Cones sobre o eixo X_1. Por exemplo, consideremos um x_2-evento envolvendo pontos p e q, como ilustrado na Figura 5.12.

Análise do Algoritmo ProcessaX_2Evento

A complexidade do algoritmo é $O(\log n)$.

Na Alternativa I, para determinarmos se existe $v \in S$ tal que Cands(v), devemos seguir a referência de p para o nó que representa p em uma das árvores balanceadas de busca que representam os conjuntos Cands(). Depois basta seguirmos os apontadores para o pai de cada no da árvore até o nó raiz que, por sua vez, faz referência ao ponto v desejado. Tudo isto pode ser realizado em tempo $O(\log n)$. As inserções e remoções de pontos na árvores balanceadas de busca que representam Pais$_a$(), Pais$_b$() podem ser feitas em tempo $O(\log n)$. A remoção do conjunto de pontos M de Pais$_b(q)$ e respectiva inserção em Pais$_b(p)$ também pode ser feito em $O(\log n)$ utilizando-se as já mencionados concatenações em Knuth [31], já que M forma um segmento contíguo dos pontos em Pais$_b(q)$ ordenados segundo a x_3-ordem.

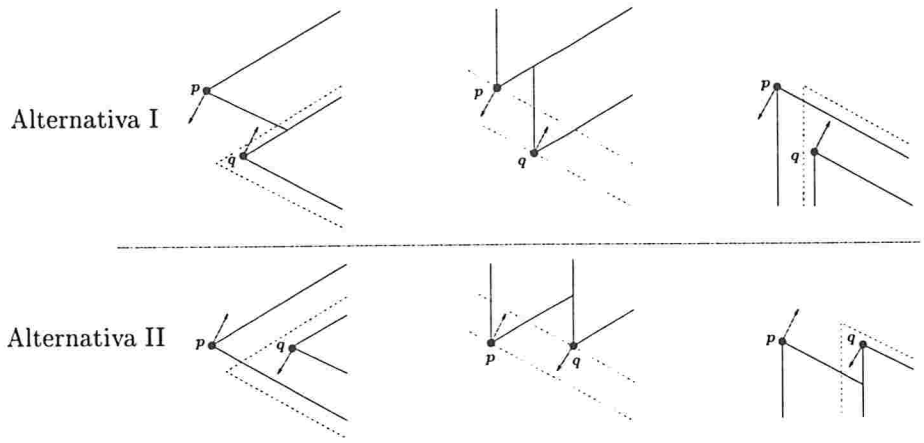


Figura 5.12: Ilustração das possíveis mudanças nos Diagramas dos Cones sobre os eixos X_2 e X_3 após um x_2-evento envolvendo os pontos p e q.

As operações realizadas na Alternativa II são semelhantes as executadas na Alternativa I: basicamente, inserções, remoções e buscas em árvores balanceadas de busca.

Evento envolvendo a x_3-ordem: x_3-evento

Um x_3-evento é a violação da x_3-ordem entre dois pontos consecutivos nesta ordem. A situação está ilustrada na Figura 5.13. O Algoritmo ProcessaX_3Evento que trata deste evento é simétrico ao algoritmo ProcessaX_2Evento e, portanto, será omitido.

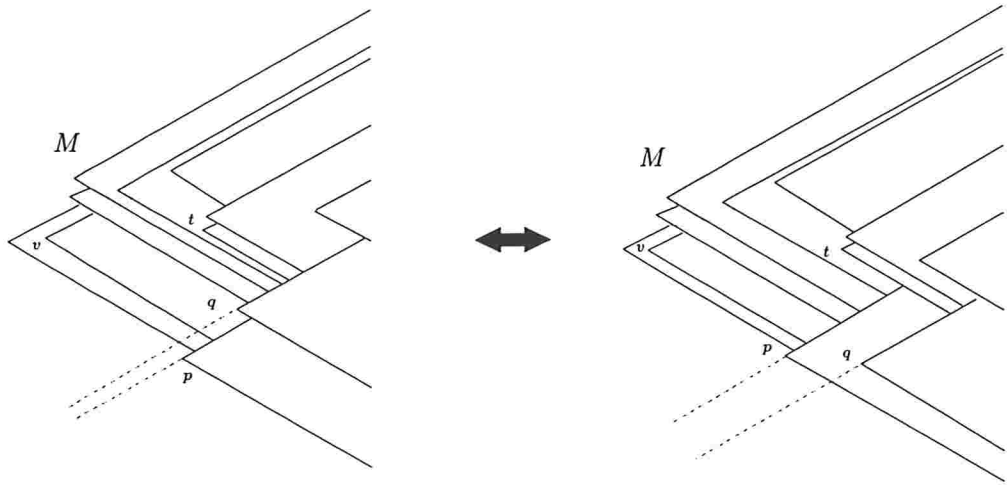


Figura 5.13: Processamento do evento quando p e q alteram sua x_3-ordem.

Mudanças na x_3-ordem podem produzir alterações nos Diagramas dos Cones sobre os

eixos X_2 e X_3, além das alterações no Diagrama dos Cones sobre o eixo X_1. A Figura 5.14 ilustra algumas possíveis alterações.

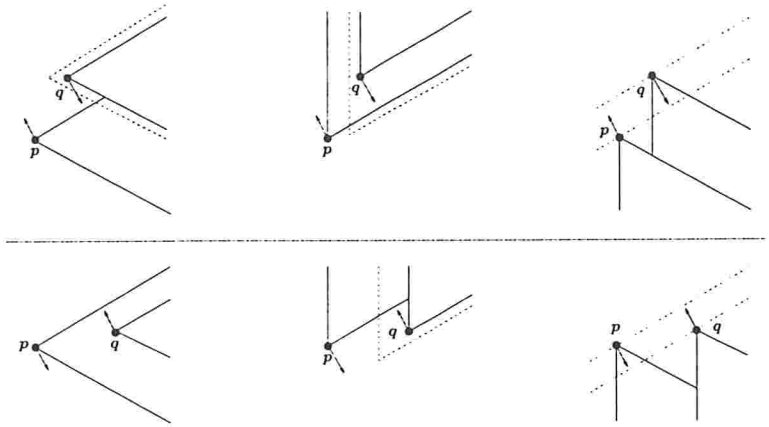


Figura 5.14: Ilustração das possíveis mudanças nos Diagramas dos Cones sobre os eixos X_2 e X_3 após um x_3-evento envolvendo os pontos p e q.

Análise da estrutura de dados cinética

Teorema 5.8 *A estrutura de dados cinética proveniente da cinetização do Algoritmo ParMaisProximo é eficiente, de resposta rápida, compacta e local.*

Prova. Ao analisarmos a estrutura de dados cinética ($\mathbb{EDC}$) para o par mais-próximo, devemos considerar:

1 a $\mathbb{EDC}$ Torneio cinético (cf. Capítulo 4, Seção 5) para o cálculo do mínimo,

2 a $\mathbb{EDC}$ proveniente da cinetização do Algoritmo ParesDeCandidatos, o qual chamaremos de primeira $\mathbb{EDC}$.

A mudança de lcand(p), para algum ponto p, e portanto a alteração da distância entre p e lcand(p) é corresponderá a uma mudança do plano de vôo, do ponto que representa a distância entre p e lcand(p) na $\mathbb{EDC}$ Torneio cinético que mantém o mínimo. Lembremos que a $\mathbb{EDC}$ Torneio cinético, para pontos que se movimentam de acordo com um k-movimento, é eficiente, de resposta rápida, compacta e local e, portanto, ótima.

A $\mathbb{EDC}$ para o Algoritmo ParMaisProximo é eficiente. De fato, o número de eventos externos, no pior caso, para o problema do par mais-próximo de um conjunto de n pontos no plano é $O(n^2 \log^* n)$ (cf. Teorema 5.1). O número total de alterações na x_1-ordem,

x_2-ordem e x_3-ordem dos pontos é $O(n^2)$ e, portanto, o número de eventos internos que deverão ser processados é, no pior caso, $O(n^2)$. Assim, a lista de pares $(p, \text{lcand}(p))$, para todo $p \in S$, fornecida para o cálculo do mínimo pode ser alterada $O(n^2)$ no pior caso. Isto significa que, no pior caso, teremos que processar $O(n^2)$ mudanças de planos de vôo na $\mathbb{EDC}$ Torneio cinético que mantém o mínimo. Cada mudança de plano de vôo em um ponto do Torneio cinético implica na atualização de, no máximo, $O(\log n)$ certificados. Desta forma, o número total de eventos processados é, no pior caso, $O(n^2 \log n)$. Logo, a $\mathbb{EDC}$ para o Algoritmo ParMaisPróximo é de eficiente, pois a razão entre o número de eventos processados, no pior caso, e do número de eventos externos, no pior caso, é $O(\log n)$.

O custo de processar um evento na $\mathbb{EDC}$ da cinetização do Algoritmo ParesDeCandidatos é $O(\log n)$. Este processamento envolve a atualização de $O(1)$ certificados. Ademais, sabemos que a $\mathbb{EDC}$ Torneio cinético é de resposta rápida. Logo, a $\mathbb{EDC}$ para o Algoritmo ParMaisPróximo é de resposta rápida.

A $\mathbb{EDC}$ para o Algoritmo ParMaisPróximo é compacta. De fato, temos um total de $3(n-1)$ certificados na fila de prioridade de eventos da $\mathbb{EDC}$ proveniente da cinetização do Algoritmo ParesDeCandidatos e a $\mathbb{EDC}$ Torneio cinético (que mantém $3n$ pontos em movimento) também é compacta.

Finalmente, resta verificarmos que a $\mathbb{EDC}$ para o Algoritmo ParMaisProximo é local. Seja p um ponto de S. A $\mathbb{EDC}$ para o Algoritmo ParesDeCandidatos mantém no máximo seis certificados envolvendo o ponto p (dois para manter a x_1-ordem, dois para manter a x_2-ordem, e dois para manter a x_3-ordem). Temos um par de pontos da forma $(p, \text{lcand}(p))$ para cada um dos Diagramas dos Cones. Logo, na lista de pares de pontos fornecidas temos no máximo seis pares contendo o ponto p e, como a $\mathbb{EDC}$ Torneio Cinético é local, temos $O(\log n)$ certificados envolvendo cada um desses pares. Logo, o número total de certificados envolvendo um dado ponto p de S é no máximo $6 + 6\log n = O(\log n)$. Como isto mostramos que a $\mathbb{EDC}$ proveniente da cinetização do Algoritmo ParMaisProximo é também local e portanto ótima. ∎

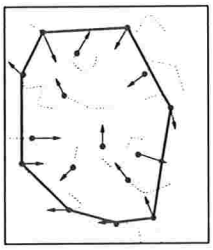
***

Os resultados obtidos para o problema do par mais-próximo, no modelo de Atallah e Kahan, são interessantes do ponto de vista teórico, mas não são apropriados para implementação.

A estrutura de dados cinética para o par mais-próximo no plano não é apropriado para inserção e remoção de pontos. Isto pode requerer um número linear de mudanças no pior caso.

CAPÍTULO 6

Problema do fecho convexo



Trataremos neste capítulo do problema do fecho convexo. A versão estática deste problema consiste em: *dado*: um conjunto S de n pontos movendo-se de maneira contínua no plano; *determinar*: a descrição combinatória do fecho convexo de S. Este problema tem sido extensivamente estudado em geometria computacional (veja, por exemplo, o Capítulo 3 de [33] e o Capítulo 3 de [36]) e são conhecidos algoritmos ótimos de complexidade de tempo $\Theta(n \log n)$.

No modelo off-line, Atallah [9, 10] mostrou, entre outras coisas, que para um conjunto S de n pontos que realizam um k-movimento no plano, a seqüência de descrições combinatórias dos fechos convexos de S, obtida ao longo do tempo, tem comprimento $O(n^2 \log^* n)$. Na Seção 6.1 apresentaremos este e outros resultados relacionados.

Descrevemos, na Seção 6.2, uma estrutura de dados cinética ótima, proposta por Basch, Guibas e Hershberger [13, 14], para o problema do envelope superior cinético. Esta estrutura de dados é baseada na cinetização de um algoritmo do tipo divisão-e-conquista para o cálculo do envelope superior de um conjunto de retas no plano.

Na Seção 6.3 consideraremos o problema do fecho convexo cinético. Este problema é equivalente, via dualidade, ao problema do envelope superior cinético. De fato, a cadeia superior (inferior) do fecho convexo de S corresponde ao envelope superior (inferior) do arranjo de retas obtidas a partir de S através de dualidade. Assim, o problema do fecho convexo cinético pode ser resolvido algoritmicamente pela estrutura de dados cinética apresentada na Seção 6.2. Na realidade, o problema do fecho convexo cinético foi a motivação para que Basch, Guibas e Hershberger desenvolvessem a estrutura de dados cinética para o envelope superior.

Finalmente, na Seção 6.4 descrevemos aplicações da $\mathbb{EDC}$ para o problema do fecho convexo na solução de outros problemas cinéticos. Mais especificamente, descrevemos como

Agarwal, Guibas, Hershberger e Veach [2] adaptaram a estrutura de dados cinética do fecho convexo para os problemas cinéticos do diâmetro e largura de um conjunto de pontos no plano.

6.1 Fecho convexo off-line

O *problema do fecho convexo*, no modelo off-line, consiste em *dado*: um conjunto S de n pontos no plano realizando um k-movimento; *determinar*: uma seqüência de fechos convexos de S ao longo do tempo.

Por uma *seqüência de fechos convexos* de S entenda-se uma seqüência formada pela descrições combinatórias de fechos convexos de S. Na seqüência, as descrições combinatórias ocorrem na ordem cronológica em que representam o fecho convexo. Assim, a primeira descrição combinatória da seqüência representa o fecho convexo no instante inicial $t = t_0$ e a última descrição combinatória representa o fecho no instante $t = +\infty$.

Devido ao movimento contínuo dos pontos, a descrição combinatória do fecho convexo só muda em certos instantes críticos. Assim, no problema do fecho convexo off-line, estamos interessados em determinar uma seqüência de fechos convexos obtida ao longo do tempo.

Nesta seção apresentaremos dois resultados devidos a Atallah [9, 10]. Será estimada a complexidade de tempo para o cálculo dos intervalos de tempo durante os quais um dado ponto de S pertence ao fecho convexo (Teorema 6.4). O outro resultado determina um limite superior para o comprimento da seqüência de fechos convexos (Corolário 6.6). Este último resultado será de particular interesse quando analisarmos a eficiência da estrutura de dados cinética mostrada na Seção 6.2.

Seja $S = \{p_1, \ldots, p_n\}$ um conjunto de n pontos no plano realizando um k-movimento. A posição do ponto p_i, no instante t, é dada pelo par $(x_i(t), y_i(t))$, onde x_i e y_i são polinômios de grau máximo k $(i = 1, \ldots, n)$.

Seja ainda $\theta_{ij}(t)$ o ângulo, no instante t, que o eixo das abscissas forma com o segmento orientado $\overline{p_i(t)p_j(t)}$ (vai de $p_i(t)$ até $p_j(t)$). Consideraremos os ângulos como sendo valores no intervalo $(-\pi, \pi]$.

Defina γ_{ij} como sendo igual a θ_{ij} quando $\theta_{ij} \geq 0$ e indefinido caso contrário, e defina β_{ij} como sendo igual a θ_{ij} quando $\theta_{ij} \leq 0$ e indefinido caso contrário. A Figura 6.1 ilustra uma típica função γ_{ij}.

Lembremos que uma função $g(t)$ tem uma transição em $t = t_0$ se é indefinida antes de t_0 e definida depois de t_0 ou se é definida antes de t_0 e indefinida depois de t_0.

Observemos que γ_{ij} e β_{ij} são funções contínuas por partes (existem pontos de descontinuidades e transições). Definamos sobre elas as seguintes funções:

$$a_i(t) = \min_{j \neq i}\{\gamma_{ij}(t)\} \qquad b_i(t) = \max_{j \neq i}\{\gamma_{ij}(t)\} \qquad c_i(t) = \min_{j \neq i}\{\beta_{ij}(t)\} \qquad d_i(t) = \max_{j \neq i}\{\beta_{ij}(t)\}.$$

Observe que as funções a_i e b_i representam o envelope inferior e superior, respectivamente, de $n-1$ funções γ_{ij}. Seja s um inteiro positivo. Lembremos que $\lambda_s(n)$ denota o maior comprimento de uma seqüência de de Davenport-Schinzel de ordem s sobre um alfabeto de n símbolos. Seja $f_s : \mathbb{N} \longrightarrow \mathbb{R}$ a função definida da seguinte maneira:

$$f_s(n) = \begin{cases} \lambda_{4s}(n) & \text{se } s > 1 \\ \lambda_3(n) & \text{se } s = 1 \end{cases}$$

De acordo ao resultado de Szemerédi [42], para s fixo, $f_s(n) = O(n \log^* n)$.

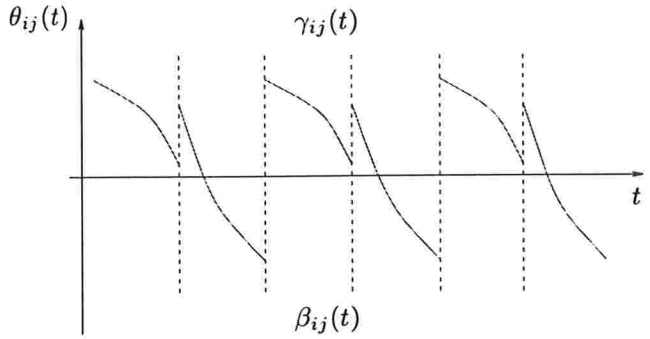


Figura 6.1: A função θ_{ij}.

Lema 6.1 *O número de transições e descontinuidades de cada uma das funções a_i, b_i, c_i e d_i é $O(n)$.*

Prova. Seja $i \in \{1, \ldots, n\}$ fixo. Provaremos o lema para a função a_i, a prova para as demais funções são similares. Como $\tan\theta_{ij} = \frac{y_j - y_i}{x_j - x_i}$, então o número de transições e descontinuidades da função γ_{ij} é, no máximo, k. Assim o número de transições e descontinuidades das $n-1$ funções γ_{ij} é $O(n)$ $(j = 1, 2, \ldots, i-1, i+1, \ldots, n)$. Como cada transição ou descontinuidade da função a_i coincide com uma transição ou descontinuidade de uma das funções γ_{ij}'s, então o número de transições e descontinuidades da função a_i é $O(n)$. ∎

Lema 6.2 *Cada um das funções a_i, b_i, c_i e d_i tem no máximo $f_k(n)$ partes.*

Prova. Seja $i \in \{1, \ldots, n\}$ fixo. Provaremos o lema para a função b_i, a prova para as demais funções são similares. Duas funções distintas θ_{ij} e θ_{il} se interceptam no máximo

$2k$ vezes, já que uma solução da equação $\theta_{ij}(t) = \theta_{il}(t)$ é também uma solução da equação $\frac{y_j(t)-y_i(t)}{x_j(t)-x_i(t)} = \frac{y_l(t)-y_i(t)}{x_l(t)-x_i(t)}$. Logo as funções γ_{ij} e γ_{il} também se interceptam no máximo $2k$ vezes. A soma do número de descontinuidades e transições da função γ_{ij} é, no máximo, k. Como b_i é o máximo de funções γ_{ij}'s, então pelo Corolário 2.5 a função b_i esta formada,por no máximo, $\lambda_{4k}(n)$ partes. Se $k = 1$, Atallah [8] mostra que b_i é composta, por no máximo, $\lambda_3(n)$ partes. ∎

Lema 6.3 *No instante t, o ponto p_i pertence ao fecho convexo de S se e só se uma das seguintes condições é válida:*

1 $a_i(t) - d_i(t) \geq \pi$,

2 $b_i(t) - c_i(t) \leq \pi$,

3 *a_i e b_i são indefinidas no instante t,*

4 *c_i e d_i são indefinidas no instante t.*

Prova. Se o ponto p_i pertence ao fecho convexo de S, então é claro que só uma das quatro condições mostradas na Figura 6.2 pode acontecer.

A seguir provaremos que se alguma das quatro condições é válida, então o ponto p_i pertence ao fecho convexo de S. Suponhamos que a primeira condição é valida, ou seja, $\alpha(t) = a_i(t) - d_i(t) \geq \pi$. Então o cone de vértice p_i e ângulo $2\pi - \alpha$ contém todos os pontos de S, assim p_i pertence ao fecho convexo de S. A prova para a segunda condição é análoga. Se a terceira condição é válida, então p_i é o ponto com maior ordenada, assim p_i pertence ao fecho convexo. Finalmente consideremos que a quarta condição é válida, então p_i é o ponto com menor ordenada, assim ele pertence ao fecho convexo. ∎

Teorema 6.4 (Atallah [9, 10]) *Seja $S = \{p_1, \ldots, p_n\}$ um conjunto de pontos que realizam um k-movimento. Então o cálculo dos intervalos de tempo durante os quais o ponto p_i pertence ao fecho convexo de S leva tempo $O(f_k(n) \log n)$ $(i = 1, \ldots, n)$.*

Prova. Seja $T(n)$ o tempo gasto para o cálculo dos intervalos de tempo durante os quais o ponto p_i pertence ao fecho convexo de S.

Seja também $T'(n)$ o tempo gasto para o cálculo da representação da função a_i. Resolver a equação $\theta_{ij}(t) = \theta_{il}(t)$, ou seja encontrar os instantes de tempo nos quais os segmentos $\overline{p_ip_j}$ e $\overline{p_ip_l}$ são paralelos, leva tempo constante (já que k é fixo). Lembremos que a função a_i representa o envelope inferior de $n-1$ funções γ_{ij} e que este envelope pode ser calculado por um algoritmo do tipo divisão-e-conquista. Então, considerando o Lema 6.2, temos que

$$T'(n) \leq 2T'(n/2) + O(f_k(n)).$$

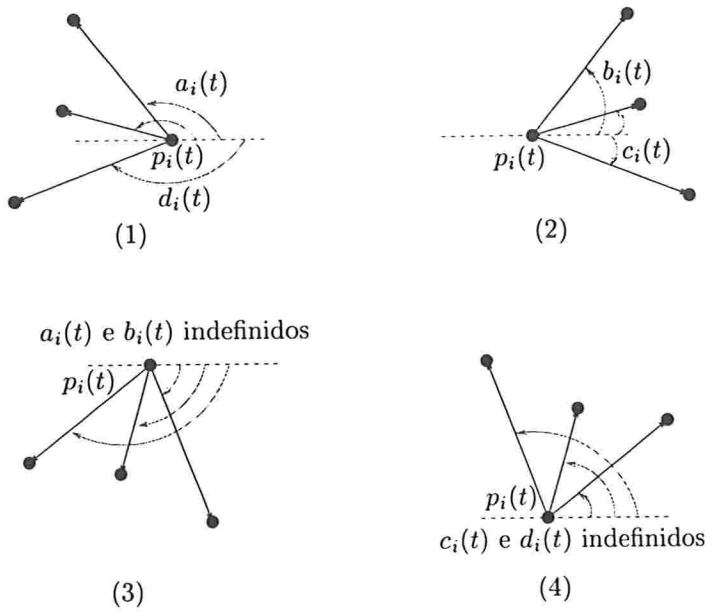


Figura 6.2: As quatro condições para que o ponto $p_i(t)$ pertença ao fecho convexo.

Assim $T'(n) = O(f_k(n) \log n)$. Calcular as representações de b_i, c_i e d_i também leva tempo proporcional a $T'(n)$. Calcular as representações de $a_i - d_i$ e $b_i - c_i$ leva tempo $O(f_k(n))$. Obter os instantes de tempo nos quais as condições do Lema 6.3 é satisfeito, ou seja resolver uma equação do tipo $\theta_{ij}(t) - \theta_{il}(t) = \pi$, leva tempo $O(f_k(n))$. Portanto, temos que

$$T(n) = 4T'(n) + O(f_k(n)).$$

Ou seja, $T(n) = O(f_k(n) \log n)$. ∎

Teorema 6.5 (Atallah [9, 10]) *Seja $S = \{p_1, \ldots, p_n\}$ um conjunto de pontos que realizam um k-movimento. Então o ponto p_i muda $O(f_k(n))$ vezes entre pertencer e não pertencer ao fecho convexo de S ($i = 1, \ldots, n$).*

Prova. Seja $C(n)$ o número de vezes que o ponto p_i muda entre pertencer e não pertencer ao fecho convexo de S. Então $C(n)$ é igual ao número de vezes que cada uma das condições do Lema 6.3 muda entre ser verdadeiro e falso. O Lema 6.1 implica que a terceira e quarta condição do Lema 6.3 mudam entre ser verdadeira e falsa $O(n)$ vezes.

Vejamos agora quantas vezes muda de sinal a primeira condição do Lema 6.3 (um argumento similar se aplica para a segunda condição). Sejam p, q e r o número de descontinuidades, transições e mínimos locais da função $a_i - d_i$. O número de vezes que $a_i - d_i$ muda de sinal é $O(p + q + r)$. Do Lema 6.1 temos que $p + q = O(n)$. Do Lema 6.2 temos

que $a_i - d_i$ é formada por $O(f_k(n))$ partes, onde cada parte tem um número constante de mínimos locais. Logo, $r = O(f_k(n))$ e a primeira condição muda entre ser verdadeiro e falso $O(f_k(n))$ vezes. Portanto, $C(n)$ é igual a $O(f_k(n))$. ∎

Corolário 6.6 *Para n pontos que realizam um k-movimento no plano, a seqüência de descrições combinatórias de fechos convexos tem no máximo $O(nf_k(n))$ elementos.* ∎

6.2 Envelope superior cinético

Nesta seção descrevemos um estrutura de dados cinética para o problema do envelope superior. A estrutura que veremos é baseada na cinetização de um algoritmo do tipo divisão-e-conquista para o cálculo do envelope superior de um arranjo de retas no plano. Uma versão análoga desta estrutura de dados resolve o problema do envelope inferior.

O *problema do envelope superior*, no modelo cinético, consiste em *dado*: um conjunto S de n pontos em movimento contínuo, um plano de vôo para cada ponto, uma seqüências de alterações de planos de vôo que são fornecidas on-line; *manter*: a descrição combinatória do envelope superior de $\mathbb{D}(S)$, ao longo do tempo.

O envelope superior (inferior) de um conjunto de retas é formado por segmentos de retas que chamaremos de *arestas*. Ademais, chamaremos de *vértices* os extremos destas arestas.

Representaremos o envelope superior de um conjunto de retas através de uma lista duplamente ligada de arestas e vértices, obtidos ao percorrermos o envelope superior (digamos) da esquerda para a direita. Esta representação será dita uma *cadeia*.

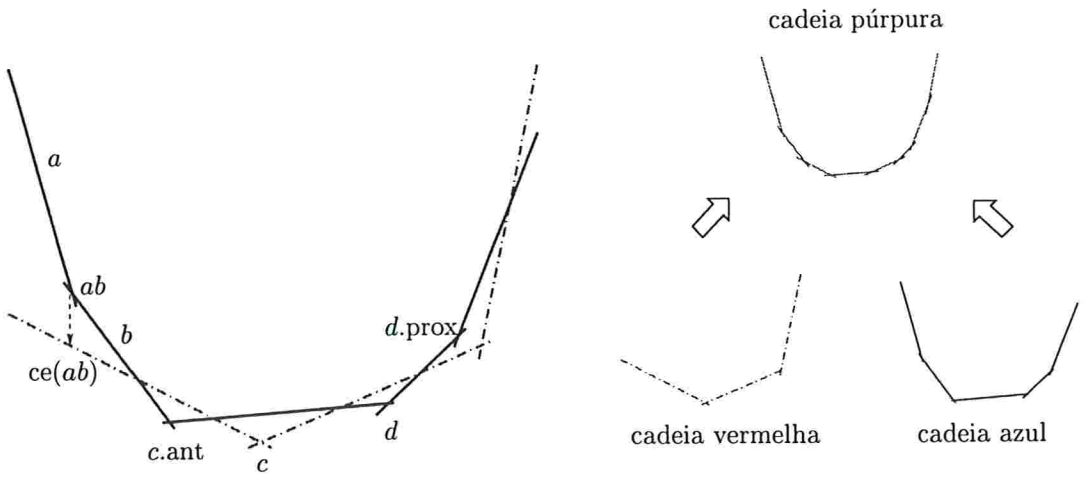


Figura 6.3: Representação de duas cadeias e as notações definidas sobre elas.

Algoritmo estático ótimo

Consideremos o seguinte algoritmo estático, do tipo divisão-e-conquista, para o problema do envelope superior de um conjunto de retas no plano. Particione recursivamente o conjunto de retas em subconjuntos de tamanhos aproximadamente iguais, calcule o envelope superior R e B de cada um desses subconjunto e devolva o envelope resultante da combinação de R e B.

A execução do algoritmo estático anterior pode ser melhor visualizado através de uma árvore de recursão, onde cada nó contém o envelope superior obtido através da combinação dos envelopes mantidos pelos nós filhos. Chamaremos esta estrutura de *árvore dos envelopes.*

Introduziremos neste parágrafo a notação que necessitaremos a seguir (veja Figura 6.3). Se a e b são duas arestas consecutivas num envelope, então denotaremos por ab o vértice entre elas. Consideremos um nó da árvore dos envelopes e os envelopes R e B cuja combinação resulta no envelope contido por este nó. Se a e b são arestas consecutivas em um dos envelopes então chamaremos de *aresta concorrente* de ab a aresta que está acima ou abaixo de ab no outro envelope. Denotaremos a aresta concorrente de ab por $\mathrm{ce}(ab)$. Se ab é um vértice de R ou B então ab.ant denotará o vértice de R ou B, à esquerda de ab, com maior x-coordenada. Analogamente, se ab é um vértice de R ou B então ab.prox denotará o vértice de R ou B, à direita de ab, com menor x-coordenada.

Cinetização do algoritmo estático

Consideremos a árvore de envelopes. Todos os envelopes contidos em nós que não representam folhas desta árvore foram obtidos através da combinação dos envelopes contidos nos filhos deste nó. Tal combinação é o único cálculo feito em cada iteração do algoritmo estático. Assim, a cinetização deste algoritmo estático consistirá na manutenção da descrição combinatória de cada um destes envelopes obtidos ao longo da execução do algoritmo, junto com os certificados que provam as suas corretudes.

Para entendermos a cinetização do algoritmo estático acima basta considerarmos a cinetização do algoritmo que combina dois envelopes 'irmãos' na árvore dos envelopes. Portanto, no que resta desta seção, consideraremos dois envelopes, um envelope R de cor vermelha e outro B de cor azul, representando os dois envelopes a serem combinados. O envelope superior obtida através da combinação de R e B será o envelope P de cor púrpura.

Certificados

Devemos obter certificados que mantêm a corretude do envelope púrpura P que resulta da combinação entre o envelope vermelho R e azul B. Estes certificados são obtidos das comparações feitas durante o processo de combinação. Este processo envolve comparações entre a x-ordem dos vértices, denotado por $<_x$ (ordem de acordo com a x-coordenada), e a y-ordem de um vértice com respeito à sua aresta concorrente, denotada por $<_y$ (ordem de acordo com a y-coordenada). O conjunto destes certificados são suficientes para provar a corretude do envelope superior resultante, mas a estrutura de dados cinética resultante não seria local (uma aresta poderia ser aresta concorrente de um número linear de vértices). Este problema foi contornado por Basch, Guibas e Hershberger através de novos certificados que envolvem comparações de tangente de arestas, denotado por $<_s$.

O conjunto de certificados envolvendo elementos (vértices e arestas) dos envelopes R (vermelho) e B (azul) serão classificados da seguinte maneira:

x-certificados: provam a x-ordem entre dois vértices de cores (envelopes) diferentes;

y-certificados: provam a y-ordem entre um vértice e sua aresta concorrente; e

s-certificados: que envolvem o valor da tangente de arestas. Estes certificados servirão para comprovar a não-participação de arestas vermelhas e azuis no envelope púrpura resultante.

Em cada uma das tabelas a seguir, a e b denotarão duas arestas consecutivas de uma mesma cor, vermelha ou azul. Ademais, se v é um vértice do envelope vermelho ou azul então cor(v) denotará a cor do vértice v. Em cada tabela, a primeira coluna contém o nome do certificado, a segunda coluna contém a comparação que o certificado prova e a terceira coluna contém as condições para que o certificado esteja presente na estrutura de dados cinética.

Nome	Comparação	Condição
x[ab]	$ab <_x ab$.prox	cor(ab) $\neq$ cor(ab.prox)

Tabela 6.1: x-certificados.

Os x-certificados provam a ordem horizontal entre os vértices de duas cadeias. Se a cor(ab) = cor(ab.prox) então os vértices ab e ab.prox pertencem ao mesmo envelope (vermelho ou azul).

Os y-certificados indicam a posição vertical entre um vértice e sua aresta concorrente. Os y-certificados estão presentes na estrutura de dados cinética se existe uma intersecção

Nome	Comparação	Condição
yli[ab]	$ab <_y$ ce(ab) ou $ab >_y$ ce(ab)	$b \cap$ ce(ab) $\neq \emptyset$
yri[ab]	$ab <_y$ ce(ab) ou $ab >_y$ ce(ab)	$a \cap$ ce(ab) $\neq \emptyset$

Tabela 6.2: y-certificados.

entre as arestas a ou b e a aresta concorrente ce(ab) (o certificado de tangência yt, o qual veremos a continuacão, tambem é um y-certificado). Cada intersecção entre os envelopes vermelho e azul esta 'cercada', no máximo, pelos y-certificados yli e yri.

Descrevemos agora os certificados para o caso no qual não existe intersecção entre as arestas a ou b e a aresta concorrente ce(ab). Estes certificados são classificados em dois sub-grupos, a saber, certificados de tangência e certificados de não-tangência.

Nome	Comparação	Condições
yt[ab]	ce(ab) $<_y ab$	$a <_s$ ce(ab) $<_s b$
slt[ab]	$a <_s$ ce(ab)	ce(ab) $<_y ab$
srt[ab]	ce(ab) $<_s b$	

Tabela 6.3: Os certificados de tangência.

Os certificados de tangência indicam, em particular, que a aresta ce(ab) não pertence ao envelope superior resultante. Estes certificados existem em conjunto. Observe que yt é um y-certificado.

Nome	Comparação	Condições
sl[ab]	$b <_s$ ce(ab)	$b <_s$ ce(ab) $ab <_y$ ce(ab) cor(ab) $\neq$ cor(ab.prox)
sr[ab]	ce(ab) $<_s a$	ce(ab) $<_s a$ $ab <_y$ ce(ab) cor(ab) $\neq$ cor(ab.ant)

Tabela 6.4: Os certificados de não-tangência.

As condições do certificado de não-tangência indicam que não existe certificado de tangência envolvendo ab. Por exemplo, a existência do certificado sl[ab] implica a não existência dos certificados de tangência yt[ab], slt[ab] e srt[ab].

Para completar a lista de certificados é necessário considerarmos certificados que envol-

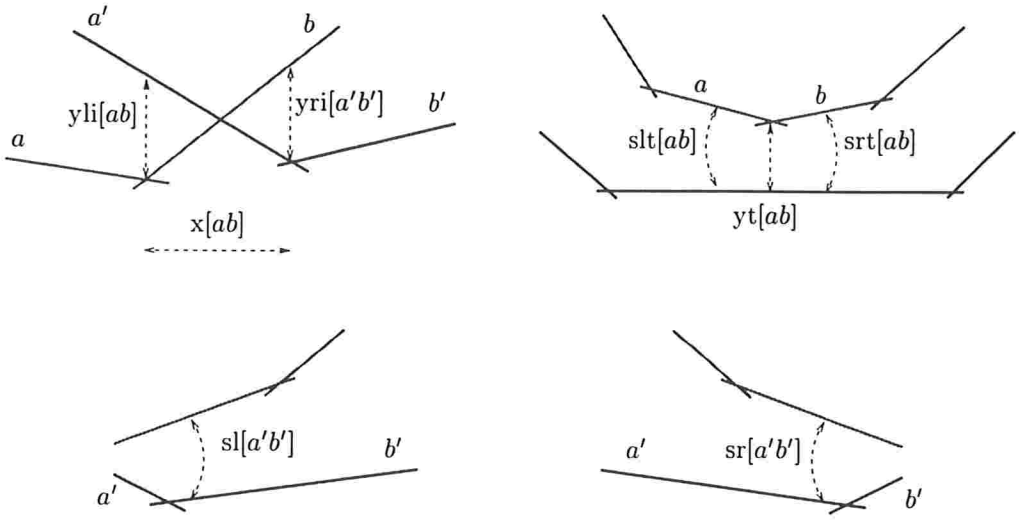


Figura 6.4: Lista de certificados que mantêm a corretude do envelope que resulta da combinação de dois sub-envelopes.

vam comparações entre os valores das tangentes das duas arestas mais à esquerda (uma de cada cor) e as duas mais à direita. Todos os tipos de certificados, mostrados nas tabelas anteriores, são ilustrados na Figura 6.4.

Uma *configuração* dos envelopes vermelho e azul é qualquer disposição dos seus vértices e arestas que mantenham as descrições combinatórias de ambos os envelopes.

Teorema 6.7 (Basch, Guibas e Hershberger [13, 14]) *Seja R e B os envelopes superiores de cor vermelho e azul, respectivamente. Então o conjunto de certificados definidos sobre R e B prova a corretude da descrição combinatória do envelope púrpura que resulta da combinação dos envelopes vermelho e azul.*

Prova. Seja C uma configuração de R e B, e L o conjunto de certificados definidos sobre ambos envelopes. Seja C' outra configuração de R e B na qual os certificados em L também são válidos. É suficiente mostrarmos que o envelope superior de C' tem os mesmos vértices que o envelope superior de C.

Os x-certificados atestam que a aresta concorrente de cada vértice de R e B, em ambas as configurações, são as mesmas. Qualquer vértice que tem um y-certificado em L é um vértice do envelope superior de C' e C. Finalmente, devemos mostrar que os vértices sem um y-certificado não podem estar situados em posições diferentes nos envelopes superiores de C e C'.

Considere uma seqüência consecutiva maximal I de vértices de R e B sem y-certificados

em L. Podemos supor sem perda de generalidade que o envelope vermelho está acima do envelope azul no intervalo I. Dado um ponto p, seja r a aresta vermelha e b a aresta azul que se interceptam com a reta vertical que passa pelo ponto p. Denotemos por $\Delta(p)$ a diferença entre o valor da tangente de r e o valor da tangente de b. Em vértices de I a função Δ cresce em vértices vermelhos e decresce em vértices azuis. Não existe mudança de sinal da função Δ num vértice vermelho contido em I, caso contrário existiria um certificado yt em L para tal vértice. Logo, pode existir uma mudança de sinal da função Δ num vértice azul contido na seqüência maximal, como está ilustrado na Figura 6.5.

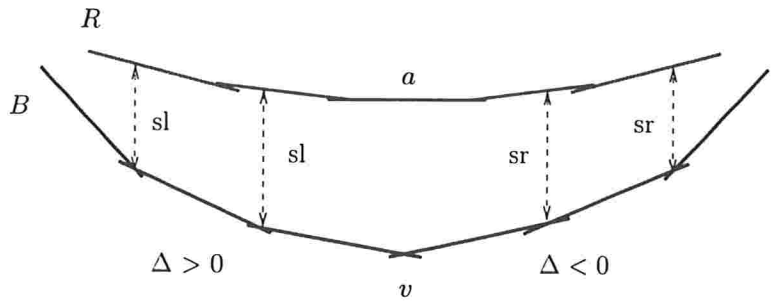


Figura 6.5: Teorema 6.7.

Assim, no intervalo I, $\Delta(p)$ é positiva até um vértice v azul, e logo depois é negativa. Seja a a aresta concorrente do vértice v. Então existem certificados do tipo sl para os vértices azuis de I que estão à esquerda de a e certificados do tipo sr para os vértices azuis em I que se encontram à direita de a. Estes certificados garantem que, no intervalo I, o envelope vermelho R permanece acima do envelope azul B. Portanto, o envelope superior de C' tem os mesmos vértices que o envelope superior de C. ∎

Processamento dos eventos

A violação de um y-certificado produz uma mudança no envelope superior púrpura resultante. Como conseqüência de tal violação uma aresta entra ou sai do envelope púrpura. Consideremos o exemplo ilustrado na Figura 6.6. Sejam R e B os envelopes vermelho e azul a serem combinados. Sejam também b, c e d arestas consecutivas no envelope B. Analisemos o caso em que a aresta c deixa de fazer parte do envelope púrpura e as arestas b e d passam a ser consecutivas no envelope púrpura. Isto corresponde à violação do certificado yli[bd]. Neste caso, podemos afirmar que c não é uma aresta concorrente para qualquer vértice na cadeia R (se c fosse uma aresta concorrente para algum vértice de R então a violação de um x-certificado teria ocorrido antes da violação do certificado yli[bd]). Assim, um número constante de certificados serão removidos da fila de eventos. Mais especificamente, os certificados que envolvem os vértices bc e cd serão removidos da fila. Os certificados

removidos são trocados por certificados que envolvem o vértice bd. Considere agora a o caso inverso onde a aresta c passa a fazer parte do envelope púrpura e o vértice bd deixa de existir. Neste caso dois novos vértices ab e bc surgem no envelope resultante. Estes novos vértices envolvem um número constante de novos certificados.

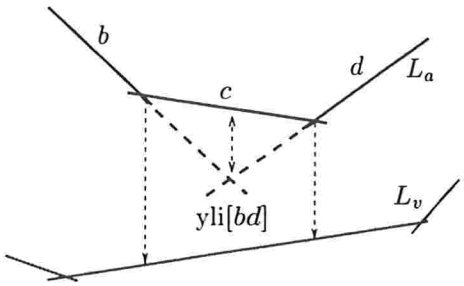


Figura 6.6: A aresta c some do envelope que resulta da combinação das cadeias L_a e L_v quando acontece a violação do certificado yli[bd].

Descrevemos a seguir os algoritmos que processam os diferentes tipos de eventos. Cada algoritmo recebe como entrada a estrutura de dados cinética para o envelope superior e o vértice ab envolvido na violação de tal certificado. Os algoritmos que processam eventos devem atualizar a estrutura de dados cinética.

Processamento de um yli-evento

Um yli-evento é a violação de um certificado do tipo yli. O seguinte algoritmo mostra como processar um tal evento. A situação está ilustrada nas Figuras 6.7, 6.8 e 6.9.

> **Algoritmo ProcessaYLIevento**. Recebe a estrutura árvore dos envelopes, de um arranjo de n retas no plano; a fila de eventos Q, contendo o conjunto de certificados; referências para os envelopes vermelho R e azul B que envolvem o evento (os envelopes que se encontram em nós irmãos na árvore dos envelopes); e um par de arestas a e b consecutivas em R. Devolve a árvore dos envelopes atualizada após processar o evento associado à perda de validade do certificado yli[ab].

Remova os certificados yri[ab.prox] e yli[ab] da fila de eventos.

Alternativa I existe yri[ab] em Q

Remova yli[ab.ant] e yri[ab] de Q.

Crie slt[ab], srt[ab] e yt[ab] e insira-os em Q.

Remova a aresta ce(ab) do envelope B.

Alternativa II não existe yri[ab] em Q

Crie yri[ab] e yli[ab.ant].

Caso 1 $ab <_y$ ce(ab)

Insira a no envelope púrpura resultante da combinação entre R e B.

Caso 2 ce(ab) $<_y ab$

Remova b do envelope púrpura resultante da combinação entre R e B.

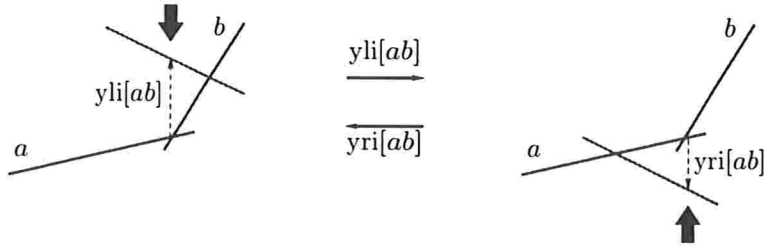


Figura 6.7: A aresta a passa a fazer parte do envelope púrpura, logo após a violação do certificado yli[ab]. Se o certificado yri[ab] é violado então a aresta a deixa o envelope púrpura.

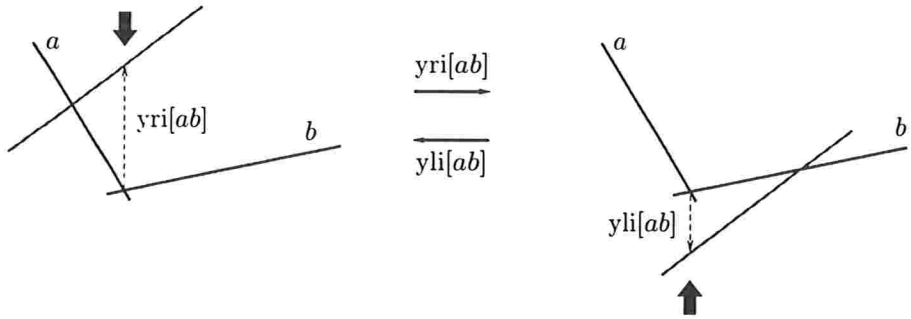


Figura 6.8: A aresta b passa a fazer parte do envelope púrpura logo após a violação do certificado yri[ab]. Se o certificado yli[ab] é violado então a aresta b deixa o envelope púrpura.

Processamento de um yt-evento

Um yt-evento é a violação de um certificado do tipo yt. O seguinte algoritmo mostra como processar um tal evento. A situação está ilustrada na Figura 6.9.

Algoritmo ProcessaYTevento. Recebe a estrutura árvore dos envelopes, de um arranjo de n retas no plano; a fila de eventos Q, contendo o conjunto de certificados; e um par de arestas a e b consecutivas em um dos envelopes vermelho R ou azul B. Devolve a árvore dos envelopes atualizada após processar o evento associado à perda de validade do certificado yt[ab].

Remova yt[ab], slt[ab] e srt[ab] de Q.

Crie yli[ab] e yri[ab.prox] e insira-os em Q.

Crie yri[ab] e yli[ab.ant] e insira-os em Q.

Insira a aresta ce(ab) no envelope púrpura resultante.

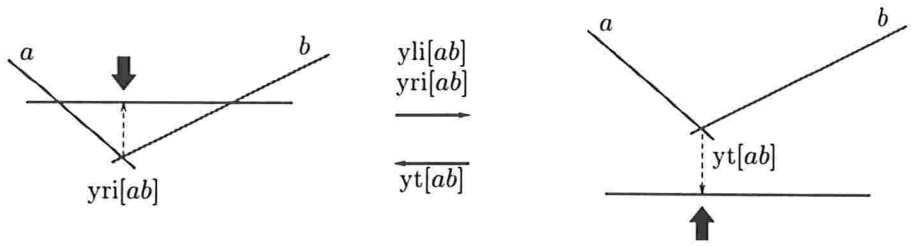


Figura 6.9: A aresta ce(ab) sumi logo depois da violação do certificado yri[ab] ou yli[ab]. Se o certificado yt[ab] é violado então a aresta ce(ab) surge no envelope resultante.

Processamento de um slt-evento

Um slt-evento é a violação de um certificado do slt. O seguinte algoritmo mostra como processar um tal evento. A situação está ilustrada nas Figuras 6.10 e 6.11.

Algoritmo ProcessaSLTevento. Recebe a estrutura árvore dos envelopes, de um arranjo de n retas no plano; a fila de eventos Q, contendo o conjunto de certificados; e um par de arestas a e b consecutivas em um dos envelopes vermelho R ou azul B. Devolve a árvore dos envelopes atualizada após processar o evento associado à perda de validade do certificado slt[ab].

Remova os certificados slt[ab], srt[ab] e yt[ab] da fila Q.
Seja $cd := ab$.ant.

Alternativa I $d = a$

Crie slt[cd], srt[cd] e yt[cd] e insira-os em Q.

Alternativa II $d \neq a$

Crie sl[cd] e insira-o em Q.

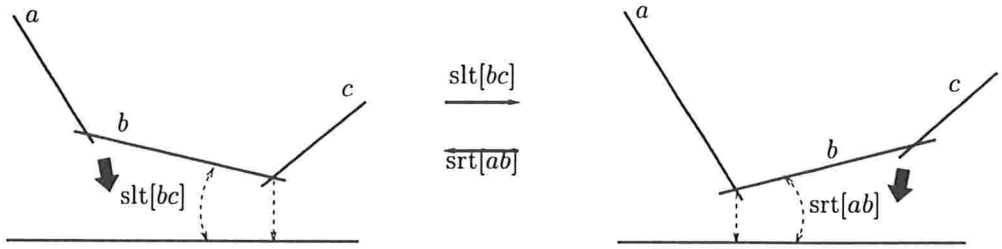


Figura 6.10: Violação de certificados do tipo slt e srt.

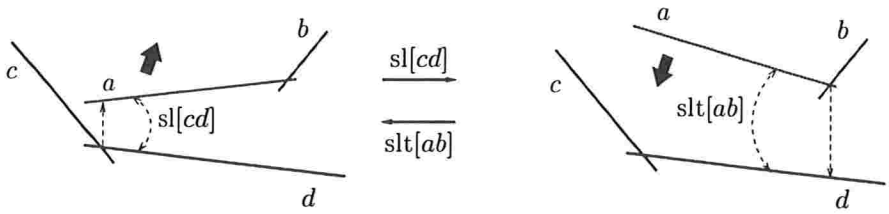


Figura 6.11: Violação de certificados do tipo sl e slt.

Processamento de um sl-evento

Um sl-evento é a violação de um certificado do tipo sl. O seguinte algoritmo mostra como processar um tal evento. A situação está ilustrada nas Figuras 6.11 e 6.12.

> **Algoritmo ProcessaSLevento.** Recebe a estrutura árvore dos envelopes, de um arranjo de n retas no plano; a fila de eventos Q, contendo o conjunto de certificados; e um par de arestas a e b consecutivas em um dos envelopes vermelho R ou azul B. Devolve a árvore dos envelopes atualizada após processar o evento associado à perda de validade do certificado sl[ab].

Remova o certificado sl[ab] da fila Q.
Seja $cd := ab$.prox.

Alternativa I cor(cd) $\neq$ cor(ab)

Crie slt[cd], srt[cd] e yt[cd] e insira-os em Q.

Alternativa II cor(cd) = cor(ab)

Crie sr[cd]e insira-o em Q.

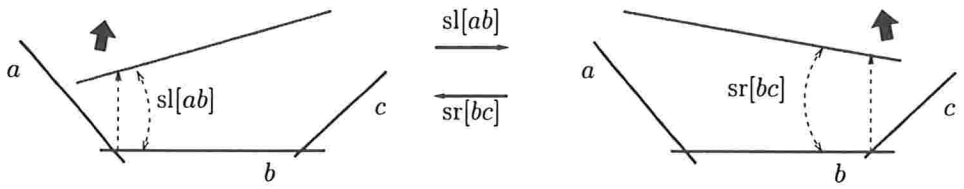


Figura 6.12: Violação de certificados do tipo sl e sr.

Processamento de um x-evento

Um x-evento é a violação de um certificado do tipo x. O seguinte algoritmo mostra como processar um tal evento.

> **Algoritmo ProcessaXevento.** Recebe a estrutura árvore dos envelopes, de um arranjo de n retas no plano; a fila de eventos Q, contendo o conjunto de certificados; e um par de arestas a e b consecutivas em um dos envelopes vermelho R ou azul B. Devolve a árvore dos envelopes atualizada após processar o evento associado à perda de validade do certificado x[ab].

Remova o certificado x[ab] de Q.
Sejam cd := ab.prox, ce(ab) := d e ce(cd) := a.

Alternativa I.A cor(cd.ant) = cor(cd)

Remova x[cd.ant] de Q.

Alternativa I.B cor(cd.ant) $\neq$ cor(cd)

Crie x[cd.ant] e insira-o em Q.

Alternativa II.A cor(ab.prox) = cor(ab)

Remova x[cd] de Q.

Alternativa II.B cor(ab.prox) $\neq$ cor(ab)

Crie x[ab] e insira-o em Q.

Alternativa III existe yri[ab] em Q

Remova yri[ab] de Q. Crie yri[cd] e insira-o em Q.

Alternativa IV existe yli[cd] em Q

Remova yli[cd] de Q. Crie yli[ab] e insira-o em Q.

Alternativa V.A existe sl[ab] em Q

Atualize sl[ab] para o novo ce(ab).

Alternativa V.B não existe sl[ab] em Q

Caso V.B.1 existe yt[cd] em Q

Remova slt[cd], srt[cd] e yt[cd] de Q. Crie sl[ab] e insira-o em Q.

Caso V.B.2 não existe yt[cd] em Q

Suponhamos que existe sr[ab] em Q. Se $a <_s d$ então remova sr[ab] de Q e crie slt[cd], srt[cd] e yt[cd] e insira-os em Q. Se $d <_s a$ então atualize sr[ab] para o novo ce(ab).

Alternativa VI.A existe sl[cd] em Q

Atualize sl[cd] para o novo ce(cd).

Alternativa VI.B não existe sl[ab] em Q

Caso VI.B.1 existe yt[ab] em Q

Remova slt[ab], srt[ab] e yt[ab] de Q. Crie sr[cd] e insira-o em Q.

Caso VI.B.2 não existe yt[ab]

Suponhamos que existe sr[cd] em Q. Se $d <_s b$ então remova sr[cd] de Q e crie slt[ab], srt[ab] e yt[ab] e insira-os em Q. Se $b <_s d$ então atualize sr[cd] para o novo ce(cd).

Os eventos que correspondem a os certificados yri[ab], sr[ab] e srt[ab] são simétricos a yli[ab], sl[ab] e slt[ab], respectivamente. A violação dos certificados sr e srt estão ilustrados na Figura 6.13.

Análise da estrutura de dados cinética

Lema 6.8 *A estrutura de dados cinética para manter a descrição combinatória do envelope superior de n retas é local.*

Prova. Uma reta r está presente num envelope como uma aresta (segmento de r). Para provarmos que a $\mathbb{EDC}$ para o envelope superior é local basta mostrarmos que o número

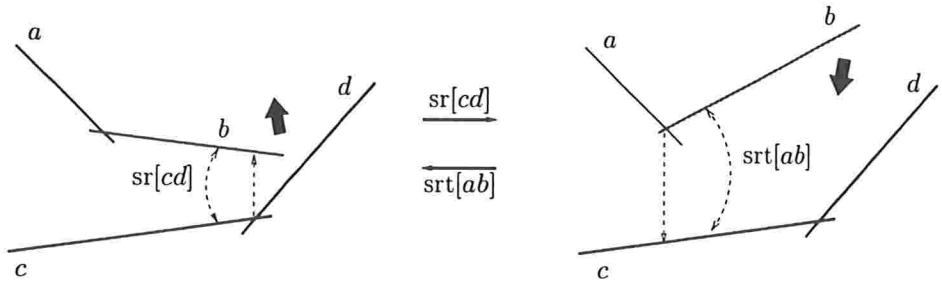


Figura 6.13: Violação de certificados do tipo sr e srt.

de certificados envolvendo cada reta é pequeno, ou, em outras palavras, que o número de certificados envolvendo arestas formadas por segmentos de uma reta é pequeno.

Seja r uma reta. Como a árvore dos envelopes tem altura logarítmica, então a reta r está presente em no máximo $O(\log n)$ envelopes. Seja E um dos envelopes que contém uma aresta a que é um segmento de r. Mostraremos que o número de certificados, com relação ao envelope E, que envolvem a aresta a é constante, e assim a $\mathbb{EDC}$ é local, já que $O(\log n)$ certificados dependerão de r (e de seu plano de vôo, ou da alteração do seu plano de vôo).

Primeiramente contemos os certificados que envolvem a aresta a devido a seus extremos (vértices). Cada vértice pode estar envolvido em dois x-certificados, em dois y-certificados ou um y-certificado e um s-certificado. Assim, a aresta a, devido a seus extremos, está envolvida em um número constante de certificados.

Contemos os certificados que envolvem a aresta a quando ela é considerada como uma aresta concorrente de algum vértice. A aresta a tem intersecção não-vazia com no máximo duas arestas do outro envelope, o que envolveria um certificados do tipo yli e yri. Consideremos agora que a aresta a tem intersecção vazia com as arestas do outro envelope. Suponha que a aresta concorrente a está abaixo do outro envelope. Então, não é difícil perceber que só pode existir no máximo um certificado do tipo slt, um do tipo yt, e um do tipo srt envolvendo a aresta a. Finalmente, suponha que a aresta a está acima da outra cadeia. Neste caso, só pode existir certificados de não-tangência envolvendo a. A Figura 6.14 ilustra a presente situação. No pior caso a é a aresta concorrente de um número linear de vértices. Todos estes vértices são da mesma cor (pertencem ao mesmo envelope). Assim, só os vértices mais à esquerda e mais à direita da parte do envelope abaixo de a podem envolver a aresta a em certificados do tipo sl ou sr. Com isto terminamos a prova do lema. ∎

Teorema 6.9 (Basch, Guibas e Hershberger [13, 14]) *A estrutura de dados cinética para manter a descrição combinatória do envelope superior de n retas é ótima.*

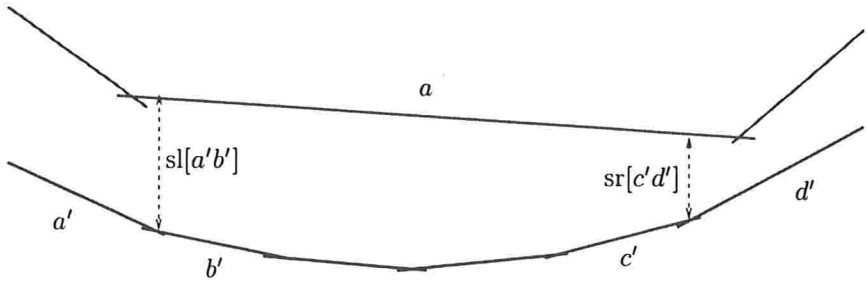


Figura 6.14: A aresta a é aresta concorrente de todos os vértices da outra cadeia que estão sob ela.

Prova. Pelo Lema 6.8, segue que a $\mathbb{EDC}$ é local. Resta, portanto, mostrarmos que a $\mathbb{EDC}$ é de resposta rápida, compacta e eficiente.

Lembremos que a violação de um y-certificado muda o envelope superior, uma aresta pode entrar ou sair do envelope e esta alteração pode se propagar até o nó raiz na árvore dos envelopes. Assim, o processamento da violação de um y-certificado leva tempo $O(\log^2 n)$. O processamento da violação dos outros tipos de certificados consiste em comparações e em um número constante de inserções e remoções de certificados, e portanto pode ser feitos em tempo $O(\log n)$. Com isto concluímos que a $\mathbb{EDC}$ é de resposta rápida.

Consideremos um determinado nível na árvore dos envelopes e seja E um envelope representado por um nó neste nível (E é o resultado da combinação dos envelopes mantidos pelos nós filhos). O número de certificados que envolvem uma aresta de E é constante. O envelope superior de um conjunto de n retas tem no máximo n partes que são representadas por no máximo n arestas. Então, o número total de certificados que mantêm a corretude da descrição combinatória de todos os envelopes neste nível é $O(n)$. Logo, o total número de certificados contidos na fila de prioridade é $O(n \log n)$. Portanto, a $\mathbb{EDC}$ é compacta.

Mostraremos agora que a $\mathbb{EDC}$ é eficiente. Para isto devemos estimar o número total de eventos processados pela $\mathbb{EDC}$ no pior caso e o número de eventos externos no pior caso.

Seja $C(n)$ o número de eventos totais processados pela $\mathbb{EDC}$ para manter o envelope superior de n retas. Devido a recursividade da árvore dos envelopes, esta quantidade é limitada superiormente por $2C(n/2) + Q(n)$, onde $Q(n)$ é o número de eventos processados no pior caso para manter o envelope superior obtido através da combinação de dois envelopes (o vermelho e o azul), cada um destes formados por aproximadamente metade das n retas.

Denotemos por t a variável 'tempo'. A família de retas $L_i : y(x,t) = m_i(t)x + b_i(t)$, correspondente ao ponto $p_i(t) = (m_i(t), b_i(t))$ $(i = 1, \ldots, n)$, é uma família de funções algébricas em duas variáveis. Seja $\mathcal{E}(t)$ o envelope superior das n retas no instante t e $\mathcal{M}$ a

superfície descrita por $\mathcal{E}(t)$, quando t varia ao longo do tempo. Observemos que a superfície $\mathcal{M}$ corresponde ao envelope superior de n superfícies formadas pelo movimento das n retas. Observe também que um vértice no envelope superior $\mathcal{E}(t)$ (púrpura) corresponde a uma aresta sobre a superfície $\mathcal{M}$.

A violação de um y-certificado corresponde a um vértice sobre a superfície $\mathcal{M}$. Quando uma aresta passa a fazer parte do envelope $\mathcal{E}(t)$ temos que um vértice sobre superfície $\mathcal{M}$ passa a dividir-se em duas arestas. Quando uma aresta deixa de fazer parte do envelope $\mathcal{E}(t)$ temos que dois arestas sobre a superfície $\mathcal{M}$ passa a unir-se num vértice. Como as superfícies descritas pela família de retas são algébricas de grau limitado, então o a superfície $\mathcal{M}$ tem complexidade combinatória $O(n^{2+\epsilon})$ para algum $\epsilon > 0$ (cf. Sharir [40]). Portanto, o número de eventos produzidos pela violação de um y-certificado $(\mathrm{yt}, \mathrm{yri}, \mathrm{yli})$ é $O(n^{2+\epsilon})$.

A intersecção das projeções sobre o plano xt ($y = 0$) entre arestas vermelhas e azuis corresponde a violação de um x-certificado. A complexidade de tais intersecções é $O(n^{2+\epsilon})$ para algum $\epsilon > 0$ (cf. Agarwal, Schwarzkopf, e Sharir [4]). Assim, o número de eventos produzidos pela violação de um x-certificado é $O(n^{2+\epsilon})$.

Cada par de retas ficam paralelas um número constante de vezes. Assim, o número de eventos produzidos pela violação de um s-certificado $(\mathrm{slt}, \mathrm{srt}, \mathrm{sl}, \mathrm{sr})$ é $O(n^2)$. Logo, $Q(n)$ é igual a $O(n^{2+\epsilon})$. Portanto, $C(n) = O(n^{2+\epsilon})$ para algum $\epsilon > 0$.

Uma mudança no fecho convexo de um conjunto de pontos descrevendo um k-movimento corresponde a uma mudança no envelope superior ou inferior no arranjo de retas duais, e vice-versa. No pior caso o número de mudanças do fecho convexo de n pontos descrevendo um k-movimento é $\Omega(n)$ (cf. Agarwal, Guibas, Hershberger e E.Veach [2]). Logo, a $\mathbb{EDC}$ é eficiente. ∎

6.3 Fecho convexo cinético

O *problema do fecho convexo*, no modelo cinético, consiste em *dados*: um conjunto S com n pontos (no plano) em movimento contínuo, um plano de vôo para cada ponto, uma seqüências de alterações de planos de vôo que são fornecidas on-line; *manter*: a descrição combinatória do fecho convexo de S ao longo do tempo.

O problema dual do problema de construir o fecho convexo de um conjunto S de pontos no plano consiste em calcular o envelope superior e inferior do arranjo dual $\mathbb{D}(S)$ de S, onde cada ponto (p, q) de S será associado à reta $y = px + q$ de $\mathbb{D}(S)$. Através da dualidade temos que as cadeias superior e inferior do fecho convexo de S correspondem aos envelopes superior e inferior de $\mathbb{D}(S)$, repectivamente. Assim, para mantermos a descrição combinatória do fecho convexo de um conjunto S de pontos em movimento basta manter-

mos o envelope superior e inferior de $\mathbb{D}(S)$. Portanto, dos resultados apresentados na seção anterior concluímos o seguinte teorema.

Teorema 6.10 (Basch, Guibas e Hershberger [14, 13]) *A estrutura de dados cinética (resultante da estrutura de dados cinética para manter o envelope superior e inferior) para manter a descrição combinatória do fecho convexo é ótima* ∎

***

Os resultados obtidos para o problema do fecho convexo, no modelo de Atallah, são interessantes do ponto de vista teórico, mas não é apropriado para implementação.

A estrutura de dados cinética para o fecho convexo no plano, não é apropriado para inserção e remoção de pontos. Isto pode requerer um número linear de mudanças no pior caso. Basch, Guibas e Hershberger, sugerem cinetizar a estrutura de dados dinâmica para o fecho convexo devido a Overmars e Van Leeuwen [35].

6.4 Aplicações do Fecho convexo cinético

Nesta seção, veremos a estrutura de dados cinética, proposta por Agarwal, Guibas, Hershberger e Veach [2], para o problema do diâmetro e largura. Estas estruturas cinéticas são baseadas na estrutura de dados cinética para o fecho convexo, vista na Seção 6.3, e na estrutura Torneio cinético.

6.4.1 Diâmetro cinético

O *problema do diâmetro*, no modelo cinético, consiste em *dados*: um conjunto S de n pontos (no plano) em movimento contínuo; um plano de vôo para cada ponto; uma seqüência de alterações de planos de vôo que são fornecidas on-line; *manter*: a descrição combinatória de um par de pontos que determinam o diâmetro de S, ao longo do tempo.

O *diâmetro* do conjunto de pontos S, que denotaremos por $\mathrm{Diam}(S)$, é a máxima distância entre dois pontos de S. A prova do lema a seguir pode ser encontrada, por exemplo, no Capítulo 4 de Preparata e Shamos [36].

Lema 6.11 *O $\mathrm{Diam}(S)$ é igual ao diâmetro dos pontos do fecho convexo de S.* ∎

Seja P um polígono convexo no plano. Nesta seção, os vértices de P serão chamados de pontos. Assim, o polígono P é determinado por uma seqüência consecutiva de pontos.

Sejam $v_1, \ldots, v_n$ os pontos de P, listados em sentido anti-horário. Um par de pontos de P que admitem retas de suporte paralelas são ditos *antípodas* (veja Figura 6.15). Denotaremos o conjunto de todos os pares de pontos antípodas de P por Antip(P).

Lema 6.12 *Se P é um polígono convexo, então Diam(P) = $\max\{dist(p,q) : (p,q) \in Antip(P)\}$.*

Prova. Seja (u, v) um par de pontos de P cuja distância entre eles é máxima (e, portanto, igual a Diam(P)). É suficiente provarmos que u e v são pontos antípodas de P. Sejam L_u e L_v retas ortogonais ao segmento $\overline{uv}$, que passam pelos pontos u e v, respectivamente. Suponhamos, que L_v não é uma reta de suporte para v. Seja M o semiplano determinado por L_v que não contém o ponto u. Então existe um ponto w de P, contido em M tal que $\text{dist}(u, v) < \text{dist}(u, w)$, que é um absurdo. Analogamente, é possível verificarmos que L_u é uma reta de suporte para u. Portanto, o par (u, v) pertence ao conjunto Antip(P). ∎

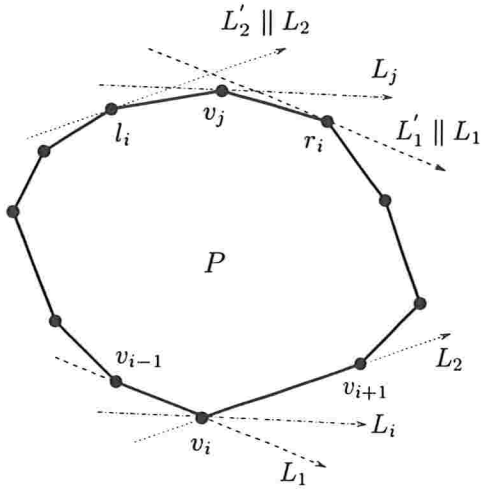


Figura 6.15: O ponto $(v_i, l_i), (v_i, v_j), (v_i, r_i)$, são exemplos de pares de pontos antípodas.

Lema 6.13 *Se P é um polígono convexo, então o conjunto Antip(P) tem no máximo $O(|P|)$ pares de pontos.*

Prova. Sejam v_{i-1}, v_i e v_{i+1} três pontos consecutivos de P, listados em sentido anti-horário. Seja r_i o primeiro ponto mais-distante do segmento $\overline{v_{i-1}v_i}$ que encontramos ao percorrer o polígono no sentido anti-horário a partir de v_i. Analogamente, seja l_i o primeiro ponto mais-distante de $\overline{v_i v_{i+1}}$, que encontramos ao percorrer o polígono no sentido anti-horário a partir de v_i.

Seja agora $A(v_i)$ o conjunto de pontos de P da cadeia entre r_i e l_i (inclusive) que não contém v_i. Seja v_j um ponto qualquer de $A(v_i)$ e L_j uma reta de suporte a v_j. Existe uma reta L_i de suporte a v_i que é paralela a L_j, como está ilustrado na Figura 6.15. $A(v_i)$ contém os pontos de P que são antípodas de v_i. Devido a maneira que escolhemos r_i e l_i, os pontos de $A(v_i)$ são os únicos pontos que são antípodas de v_i.

Os conjuntos $A(v_i)$ e $A(v_{i+1})$ contém o ponto l_i, e no máximo tem dois pontos em comum. Portanto, o número de pares de pontos antípodas é $O(|P|)$. ∎

Uma caracterização de pares de pontos antípodas

Seja P um polígono convexo. Consideremos v_i e v_j pontos pertencentes a cadeia superior e inferior de P, respectivamente, como ilustrado na Figura 6.16. Sejam m_i, m_{i+1}, m_j e m_{j+1} os valores das tangentes das retas que passam pelos pontos v_{i-1}, v_i, v_i, v_{i+1}, v_{j-1}, v_j e v_j, v_{j+1}, respectivamente. Observemos que o valor da tangente de qualquer reta de suporte passando por v_i, está contido no intervalo $[m_i, m_{i+1}]$. Analogamente, o valor da tangente de qualquer reta de suporte passando por v_j, está contido no intervalo $[m_j, m_{j+1}]$. Portanto, v_i e v_j são pontos antípodas de P se e somente se os intervalos $[m_i, m_{i+1}]$ e $[m_j, m_{j+1}]$ se interceptam.

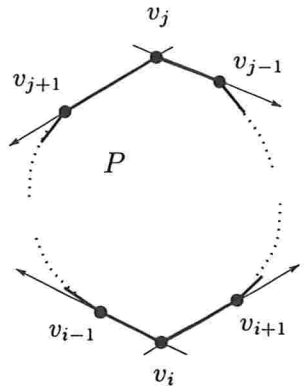


Figura 6.16: Caracterização de pontos antípodas

Determinação dos pares de pontos antípodas via dualidade

Considere o dual de um conjunto de pontos no plano. No dual, cada ponto (p, q) de S será representado pela reta $y = px + q$. O dual dos pontos da cadeia superior (inferior, resp.) do fecho convexo de S é o envelope superior(inferior, resp.), do conjunto das retas no arranjo dual $\mathbb{D}(S)$. Assim, cada ponto do fecho convexo de S se dualiza num segmento do envelope superior ou inferior de $\mathbb{D}(S)$.

Seja P um polígono convexo. Sejam r e q dois pontos consecutivos da cadeia superior ou inferior que representam P e seja (x_1, y_1) a intersecção entre as retas $\mathbb{D}(r)$ e $\mathbb{D}(q)$ duais de r e q no arranjo $\mathbb{D}(P)$. Então, o valor da tangente da reta que passa pelos pontos r e q é $-x_1$, como está ilustrado na Figura 6.17. Se consideramos os vértices dos envelope superior e inferior listados segundo as suas abscissas (de acordo a x-ordem), então, devido à caracterização de pares antípodas, podemos encontrar todos os pares de pontos antípodas simplesmente varrendo ambas as listas simultaneamente e determinando os intervalos (formados pelas x-coordenadas de vértices consecutivos em uma das listas) que se intersectam.

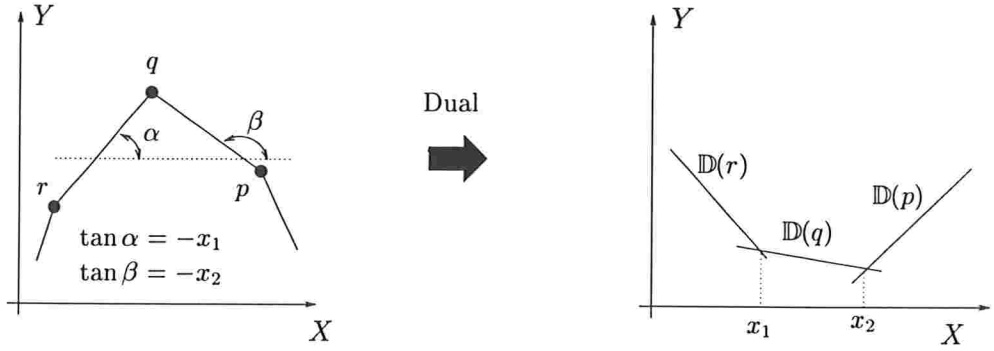


Figura 6.17: O valor da tangente da reta que passa pelos pontos r e q é $-x_1$.

Algoritmo estático ótimo

Dos Lemas 6.11 e 6.12 temos que

$$\mathrm{Diam}(S) = \max\{\mathrm{dist}(p, q) : (p, q) \in \mathrm{Antip}(\mathrm{FC}(S))\}$$

onde $\mathrm{FC}(S)$ denota o polígono que representa o fecho convexo de S. O seguinte algoritmo estático determina o diâmetro de um conjunto de pontos no plano, com base no resultado anterior.

> **Algoritmo Diâmetro.** Recebe um conjunto S de n pontos no plano e devolve um par de pontos de S cuja distância entre eles é máxima (entre todos os pares de pontos de S).

Determine o polígono P que representa o fecho convexo de S.

Calcule o conjunto $\mathrm{Antip}(P)$.

Encontre um par (u, v) de $\mathrm{Antip}(P)$ tal que a distância entre u e v seja máxima.

Devolva (u, v).

Análise do Algoritmo Diâmetro

A determinação do fecho convexo de um conjunto de n pontos pode ser feita em tempo $O(n \log n)$. O cálculo de todos os pares de pontos antípodas pode ser feito em tempo linear. O Lema 6.13 garante que a busca de pontos em Antip(P) pode também ser realizada em tempo linear. Portanto, a complexidade de tempo do Algoritmo Diâmetro é $O(n \log n)$.

Cinetização do algoritmo estático

A estrutura de dados cinética para o problema do diâmetro é baseada na estrutura de dados cinética para o fecho convexo e da estrutura Torneio cinético.

A fim de cinetizar o Algoritmo Diâmetro empregaremos certificados para manter a descrição combinatória da lista L dos vértices dos envelopes superior e inferior de $\mathbb{D}(P)$ ordenados segundo as suas x-coordenadas. Esta lista será armazenada em uma árvore balanceada de busca para que as alterações na lista possam ser processadas em tempo $O(\log n)$ mais o tempo gasto para atualizarmos o conjunto Antip(P) de pares antípodas. Isto garante a corretude dos pares que formam Antip(P). Quando um certificado envolvendo a ordem entre dois vértices é violado, um número constante de certificados envolvendo os intervalos formados por estes vértices na lista L deverão ser atualizados. Quando ocorre uma mudança na descrição combinatória do fecho convexo de S (ou seja, na descrição combinatória P) , algumas arestas podem ser inseridos ou removidos em $\mathbb{D}(P)$ e vértices devem ser inseridos e removidos na lista L.

Teorema 6.14 *A estrutura de dados para manter o diâmetro de um conjunto de n pontos é compacta e eficiente.*

Prova. Lembremos que a estrutura de dados cinética para manter o fecho convexo e a estrutura Torneio cinético são eficientes e compactas.

Pelo Lema 6.13, o número de pares de pontos antípodas é linear no número de pontos, assim a $\mathbb{EDC}$ para o diâmetro é compacta.

Mostraremos agora que a estrutura de dados cinética para manter o diâmetro é eficiente. Para isto precisamos limitar o número total eventos processados pela $\mathbb{EDC}$, no pior caso.

Seja L a lista dos vértices dos envelopes superior e inferior de $\mathbb{D}(P)$ ordenados segundo as suas x-coordenadas. O número de eventos envolvendo a lista L pode ser limitado pelo número de pares pontos que passam e deixam de ser antípodas ao longo do tempo. Considere o tempo t como a terceira dimensão. Então, em qualquer instante, o conjunto de pares de pontos antípodas são determinados pela sobreposição do envelope superior e

inferior. Observemos que o número total de pares de pontos antípodas é igual ao número de traços cujas projeções no xt-plano se intersectam. Esta quantidade é limitada por $O(n^{2+\epsilon})$ (cf. Agarwal, Erickson e Guibas [4]). No Teorema 4 da Seção 4.1 de [2] é demostrado que o número total de pares de pontos que determinam o diâmetro, ao longo do tempo, é $\Omega(n^2)$. ∎

Observemos que a estrutura de dados cinética obtida através da cinetização do Algoritmo Diâmetro é não local, já que um ponto pode pertencer a um número linear de pares de pontos antípodas.

6.4.2 Largura cinética

O *problema da largura*, no modelo cinético, consiste em *dado*: um conjunto S de n pontos (no plano) em movimento contínuo; uma seqüência de alterações de planos de vôo que são fornecidas on-line; *manter*: a descrição combinatória de três pontos que determinam a largura de S ao longo do tempo.

A *largura* de um conjunto S de n pontos no plano, é a menor distância entre duas retas de suporte de S que sejam paralelas.

Não é difícil vermos que uma das retas que determinam a largura de S deve conter uma aresta do fecho convexo de S e que a outra reta deve conter um vértice deste fecho. Seja a uma aresta e v um vértice do fecho convexo de S que se encontram em cadeias distintas do fecho (um deve estar na cadeia superior e o outro na cadeia inferior). Chamaremos o par (v, a) de um *par vértice-aresta antípoda* se existe uma reta que passa pelo vértice v e que seja paralela à aresta a.

Teorema 6.15 *A largura de um conjunto S de n pontos é o mínimo das distâncias entre retas paralelas de suporte passando através de um par vértice-aresta antípoda do fecho convexo de S.* ∎

Algoritmo estático ótimo

O seguinte algoritmo estático determina a largura de um conjunto de pontos no plano. Este algoritmo baseia-se no resultado do Teorema 6.15.

> **Algoritmo Largura.** Recebe um conjunto S de n pontos no plano e devolve um par vértice-aresta antípoda que determina a largura de S.

Calcule o polígono P que representa o fecho convexo de S.

Calcule o conjunto $\mathcal{A}$ dos pares vértice-aresta antípodas de P.

Encontre um par (v, a) de $\mathcal{A}$ tal que sua distância seja mínima.

Devolva o par (v, a).

O Algoritmo Largura é similar ao Algoritmo Diâmetro e tem a mesma complexidade de tempo que este último, ou seja $O(n \log n)$.

Cinetização do algoritmo estático

No dual, uma aresta do fecho convexo de um conjunto de pontos S corresponde a um vértice do envelope superior ou inferior do arranjo de retas $\mathbb{D}(S)$. Assim, um par (v, a) vértice-aresta antípoda corresponde a uma aresta $\mathbb{D}(v)$ sobre o envelope superior ou inferior tal que o intervalo determinado pelas x-coordenadas dos seus extremos contém a x-coordenada do vértice $\mathbb{D}(a)$ (que pertence ao envelope superior de $\mathbb{D}(S)$ caso $\mathbb{D}(v)$ pertença ao envelope inferior e vice-versa).

Para encontrar todos os pares vértice-aresta antípodas, devemos construir uma lista obtida através da intercalação das arestas dos envelopes superior e inferior de $\mathbb{D}(P)$ de acordo com a x-ordem das arestas dos dois envelopes, observando se a x-coordenada de um vértice pertence ao x-intervalo de uma aresta. Para calcular um par vértice-aresta o qual determina a largura, devemos encontrar a mínima distância de todos os pares vértice-aresta antípodas.

A estrutura de dados cinética para manter a largura é similar ao do diâmetro. A única diferença está no tipo de pares antípodas. A estrutura do Torneio cinético mantêm o par vértice-aresta cuja distância é mínima. O seguinte resultado é devido a Agarwal, Guibas, Hershberger e Veach [2].

Teorema 6.16 *A estrutura de dados cinética para manter a largura é compacta e eficiente.* ∎

***

Uma grande parte de problemas geométricos envolvem movimento. Estes problemas requerem o cálculo de algum atributo dos pontos em movimento em instantes discretos. Geralmente, consultas envolvendo movimento podem ser formuladas como a manutenção de atributos discretos, que dependem das posições dos objetos. O objetivo para resolver este tipo de problema, é usar a coerência dada pela continuidade do movimento dos pontos, e assim anular cálculos feitos previamente.

As soluções de Atallah e Kahan, para este tipo de problemas, são interessantes do ponto de vista teórico, mas não são apropriados para implementação. Ademais, eles são restritos a simples tipos de movimentos. Já a solução de Basch, Guibas e Hershberger é apropriada para implementação e considera um tipo de movimento apropriado para simulação de sistemas dinâmicos.

Referências Bibliográficas

[1] P.K. Agarwal, J. Erickson e L.J. Guibas, Kinetic binary space partitions for intersecting segments and disjoint triangles, *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, CA), ACM Press, 1998, pp. 107–116.

[2] P.K. Agarwal, L.J. Guibas, J. Hershberger e E. Veach, Maintaining the extent of a moving point set, *Proceedings 5th International Workshop on Algorithms and Data Structures* (Berlin) (F. Dehne, A. Rau-Chaplin, J.-R. Sack e R. Tamassia, eds.), Lecture Notes in Computer Science, vol. 1272, Springer-Verlag, 1997, http://www-graphics.stanford.edu/papers/extent/, pp. 31–44.

[3] P.K. Agarwal, L.J. Guibas, T. Murali e J. Vitter, Cylindrical static and kinetic binary space partitios, *Proceedings of the 13th Annual ACM Symposium on Computational Geometry* (Nice, France), ACM Press, 1997, pp. 39–48.

[4] P.K. Agarwal, O. Schwarzkopf e M. Sharir, The overlay of lower envelopes and its applications, *Discrete Computational Geometry* **15** (1996), 1–13.

[5] P.K. Agarwal e M. Sharir, *Davenport-Schinzel sequences and their geometric applications*, Cambridge University Press, New York, 1995.

[6] ———, Davenport-Schinzel sequences and their geometric applications, *Handbook of Computational Geometry*, North-Holland, to appear, http://www.math.tau.ac.il/~sharir/.

[7] A.V. Aho, J.E. Hopcroft e J.D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.

[8] M.J. Atallah, *Dynamic computational geometry*, Tech. Report 450, Dept. of Computer Sciences, Purdue University.

[9] ———, Dynamic computational geometry, *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science* (Tucson, Arizona), 1983, pp. 92–99.

[10] ———, Some dynamic computational geometry problems, *Computational Mathematics with Applications.* **11** (1985), no. 12, 1171–1181.

[11] J. Basch, *Kinetic data structures*, Ph.D. thesis, Department of Computer Science of Stanford University, 1999.

[12] J. Basch, J. Erickson, L.J. Guibas, J. Hershberger e L. Zhang, Kinetic collision detection between two simple polygons, *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999, http://compgeom.cs.uiuc.edu/~jeffe/pubs/pubs.html.

[13] J. Basch, L.J. Guibas e J. Hershberger, Data structures for mobile data, *Journal of Algorithms* (199?), to appear, http://graphics.stanford.edu/~jbasch/publications/index.html.

[14] ———, Data structures for mobile data, *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms* (New Orleans, Louisiana), ACM Press, 1997, http://graphics.stanford.edu/~jbasch/publications/index.html, pp. 747–756.

[15] J. Basch, L.J. Guibas e G.D. Ramkumar, Sweeping lines and line segments with a heap, *Proceedings of the 13th Annual ACM Symposium on Computational Geometry* (Nice, France), ACM Press, 1997, http://graphics.stanford.edu/~jbasch/publications/index.html.

[16] J. Basch, L.J. Guibas, C.D. Silverstein e L. Zhang, A practical evaluation of kinetic data structures, *Proceedings of the 13th Annual ACM Symposium on Computational Geometry* (Nice, France), ACM Press, 1997, http://graphics.stanford.edu/~jbasch/publications/index.html, pp. 388–390.

[17] J. Basch, L.J. Guibas e L. Zhang, Proximity problems on moving points, *Proceedings of the 13th Annual ACM Symposium on Computational Geometry* (Nice, France), ACM Press, 1997, http://graphics.stanford.edu/~jbasch/publications/index.html, pp. 344–351.

[18] Y.-J. Chiang, J.T. Klosowski, C. Lee e J.S.B. Mitchell, Geometric algorithms for conflict detection/resolution in air traffic management, *Proceedings of the 36th IEEE Conference on Decision and Control*, December 1997, http://cis.poly.edu/chiang/, pp. 1835–1840.

[19] Y.-J. Chiang e R. Tamassia, Dynamic algorithms in computational geometry, *Proceedings of the IEEE* **80** (1992), no. 9, 1412–1434, http://cis.poly.edu/chiang/.

[20] H. Davenport e A. Schinzel, A combinatorial problem connected with differential equations, *American Journal of Mathematics* **87** (1965), 684–694.

[21] P.J. de Resende e J. Stolfi, *Fundamentos de geometria computacional*, IX Escola de Computação, 1994.

[22] H. Edelsbrunner, *Algorithms in combinatorial geometry*, EATCS Monographs on Theoretical Computer Science, no. 10, Springer-Verlag, Berlin, 1987, QA758 E21a.

[23] J. Erickson, L. J. Guibas, J. Stolfi e L. Zhang, Separation-sensitive collision detection for convex objects, *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM Press, 1999, http://compgeom.cs.uiuc.edu/~jeffe/pubs/pubs.html.

[24] L.H. Figueiredo e P.C.P. Carvalho, *Introdução à geometria computacional*, 18º Colóquio Brasileiro de Matemática, IMPA, 1991, QA758 F475i.

[25] L.J. Guibas, J.S.B. Mitchell e T. Roos, Voronoi diagrams of moving points in the plane, *Proceedings of the 17th International Workshop Graph-Theoretic Concepts in Computer Science*, Lecture Notes Computer Science, vol. 570, Springer-Verlag, 1991, pp. 113–125.

[26] L.J. Guibas e M. Sharir, Combinatorics and algorithms of arrangements, *New Trends in Discrete and Computational Geometry* (J. Pach, ed.), Springer-Verlag, 1993, pp. 9–36.

[27] L.J. Guibas e J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *ACM Transactions on Graphics* **4** (1986), no. 2, 74–123.

[28] S. Kahan, A model for data in motion, *Proceedings of the 23th Annual ACM Symposium on Theory of Computing* (New Orleans, Lousiana), ACM Press, 1991, pp. 267–277.

[29] ———, Real-time closest pair of moving points, *Animation of Geometric Algorithms: A Video Review* (M.H. Brown e J. Hershberger, eds.), no. 87a, Digital Equipment Corporation, Palo Alto, CA, June 1992, Companion to the video, pp. 1–3.

[30] N. Katoh, T. Tokuyama e K. Iwano, On minimum and maximum spanning trees of linearly moving points, *Discrete Computational Geometry* **13** (1995), 161–176.

[31] D.E. Knuth, *The art of computer programming. Volume 3: Sorting and searching*, 2nd edition ed., Addison-Wesley, Reading, MA, 1968.

[32] K. Mulmuley, *Computational geometry: An introduction through randomized algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1994.

[33] J. O'Rourke, *Computational geometry in C*, Cambridge University Press, Cambridge, 1993.

[34] T. Ottmann e D. Wood, Dynamical sets of points, *Computer Vision, Graphics, and Image Processing* **27** (1984), 157–166.

[35] M.H. Overmars e J. van Leeuwen, Maintenance of configurations in the plane, *Journal of Computer and System Science* **23** (1981), 166–204.

[36] F.P. Preparata e M.I. Shamos, *Computational geometry: An introduction*, Texts and Monographs in Computer Science, Springer-Verlag, New York, 1985, QA758 P927c.

[37] T. Roos, Voronoi diagrams over dynamic scenes, *Discrete Applied Mathematics* **43** (1993), 243–259.

[38] T. Roos e H. Noltemeier, Dynamic Voronoi diagrams in motion planning, *Proceedings Computational Geometry: Methods, Algorithms and Applications*, Lecture Notes Computer Science, vol. 553, Springer-Verlag, 1991, pp. 227–236.

[39] ———, Dynamic Voronoi diagrams in motion planning, *Proceedings of the 15th IFIP Conference*, vol. 180, Springer-Verlag, 1992, pp. 102–111.

[40] M. Sharir, Almost tight upper bounds for lower envelopes in higher dimensions, *Discrete Computational Geometry* **12** (1994), 327–345.

[41] D.D. Sleator e R.E. Tarjan, Armotized efficiency of list update and paging rules, *Communication of the ACM* **28** (1985), 202–208.

[42] E. Szemerédi, On a problem by Davenport and Schinzel, *Acta Arithmetica* (1974), 213–224.