Busca em Subespaços em Várias Dimensões[1]

Renato Fileto[2]

Departamento de Ciência da Computação
IMECC – UNICAMP

Banca Examinadora:

- Jorge Stolfi (Suplente)[3]
- Pedro Jussieu de Rezende (Orientador)[3]
- Ronaldo Marinho Persiano[4]
- Geovane Cayres Magalhães[3]

[1] Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação da UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

[1] Este trabalho foi desenvolvido com o apoio financeiro do CNPq e da FAPESP.

[2] O autor é Bacharel em Ciência da Computação pela Universidade Federal de Uberlândia.

[3] Professor do Departamento de Ciência da Computação - IMECC - UNICAMP.

[4] COPPE - Universidade Federal do Rio de Janeiro.

Busca em Subespaços em Várias Dimensões

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Renato Fileto e aprovada pela Comissão Julgadora.

Campinas, 29 de junho de 1994.

Prof. Pedro Jussieu de Rezende
Orientador

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Busca em Subespaços em Várias Dimensões

Renato Fileto

em memória de Cláudio Sérgio da Rós de Carvalho

Agradecimentos

Aos colegas com quem tenho convivido aqui, por terem me suportado e aos amigos, pelo companheirismo, incentivo e compreensão.

Aos funcionários desta universidade, que sempre têm recebido as solicitações de serviços com gentileza e competência, algumas vezes prestativos até além de suas obrigações, especialmente os integrantes da secretaria do DCC, das bibliotecas e do centro de computação.

Aos professores do DCC pela orientação e pela paciência, em especial ao meu orientador, professor Pedro Jussieu de Rezende.

A Jirka Matoušek, da república Tcheca, que tem me fornecido seus trabalhos em desenvolvimento, os quais têm sido muito úteis e instrutivos, me auxiliando no levantamento bibliográfico e a mantê-lo atualizado.

Ao Dr. Commini, advogado na comarca de Uberlândia, pela dedicação e idealismo com que empregou seus conhecimentos de justiça em defesa de meu direito de prosseguir os estudos, concedendo-me também um pouco de sua experiência de vida, por meio de conselhos e incentivos de valor incalculável. Sem este apoio, o presente trabalho não seria possível.

Ao CNPq e à FAPESP, pelo apoio financeiro recebido.

A todas as pessoas que contribuíram de uma forma ou de outra para a produção deste trabalho, com dicas e sugestões, compartilhando seus conhecimentos e tornando nosso ambiente de pesquisa agradável e produtivo. Os nomes são tantos que se torna difícil citá-los todos aqui.

Ao povo brasileiro, que tem contribuído para sustentar tudo isso, na esperança de alcançar um desenvolvimento tecnológico, científico e cultural que lhe proporcione uma vida melhor.

'' ... the human brain is much better at
secondary key retrieval than computers are;
in fact it's rather easy for us
to recognize faces, or melodies, etc.,
from only fragmentary information
while computers
have barely been able to do this at all.
Therefore it is not unlikely that
a completely new approach to machine design
will someday be discovered,
which solves the problem ...,
making this entire section obsolete ...''

Donald E. Knuth
"The Art of Computer Programming
Sorting and Searching" (Vol.3)

Resumo

O tema central deste trabalho é a pesquisa de soluções para problemas de busca em subespaços (*range search*), sob o enfoque de projeto de algoritmos eficientes e geometria computacional, considerando objetos de dados em forma de pontos dispersos num espaço multidimensional e explorando diversos formatos de subespaços de busca encontrados na literatura. O objetivo é reunir diversas formulações e métodos de solução em um compêndio, onde estes são descritos sob uma mesma ótica, com notação uniforme e de forma mais simples que nos textos originais, de modo a facilitar um estudo mais detalhado e comparações, no que diz respeito à natureza e ao funcionamento das soluções. Pretende-se com isso tornar as idéias provenientes da pesquisa atualmente em processo na área de algoritmos acessíveis de forma mais integrada e simples, tanto aos interessados na pesquisa de métodos mais eficientes e adequados para problemas em teoria da computação, quanto àqueles mais interessados na aplicação dessas idéias.

Um estudo abrangente das soluções encontradas na literatura permite perceber diversas semelhanças de concepção nos métodos empregados. Freqüentemente, pode-se observar a ocorrência de abordagens e técnicas comuns em diversas situações. A estas abordagens e técnicas de aplicação geral atribuímos o nome de paradigmas de algoritmos. O estudo e a utilização de paradigmas de algoritmos possibilitam um certo grau de sistematização das soluções de problemas de busca em subespaços, uma vez que eles permitem encarar diversas soluções distintas, de diversas variações do problema como manifestações de um mesmo fundamento racional. Além disso, o estudo de paradigmas é instrutivo, pois promove o desenvolvimento de raciocínios sistemáticos, aplicáveis na resolução de diversos problemas em computação.

A divisão do conteúdo é efetuada de maneira a fornecer primeiro o fundamento teórico necessário à compreensão dos métodos de solução, que são tratados posteriormente. No capítulo 1, são fornecidos os conceitos e classificações básicos, relativos a problemas de busca em geral e particularmente busca em subespaços, a fim de prover uma fundamentação teórica e situar a área de estudo. No capítulo 2, são descritos alguns paradigmas de algoritmos aplicados à problemas de busca em subespaços, com o intuito de prover ao leitor maneiras alternativas de relacionar as soluções apresentadas posteriormente, induzindo-o a desenvolver raciocínios que lhe habilitem a perceber os fundamentos e técnicas em comum. Nos capítulos 3 a 6, são abordados os sub-problemas caracterizados pelos formatos clássicos de subespaços de busca encontrados na literatura, ordenados da maneira que parece mais conveniente e que reflete a complexidade das soluções, a natureza das mesmas e sua evolução histórica. Em cada um destes capítulos, os sub-problemas são discutidos em detalhes, algumas soluções e limites inferiores são descritos superficialmente e há uma seção de notas bibliográficas, com referências para assuntos específicos. Finalmente, no capítulo 7, são sintetizadas as contribuições do trabalho e relacionados alguns assuntos para possíveis extensões no futuro.

Abstract

The main objective of this work is the study of solutions found in the literature to range search, from the view point of algorithm design and computational geometry, considering only data objects in the form of points embedded in a multidimensional space, and investigating various shapes of ranges. Several formulations and solutions to range search problems are surveyed. These are described under one abstract view, with uniform notation and in a form hopefully clearer than the original sources, in such a way that comparisons of the nature and functionality of the solutions and more detailed studies may be facilitated. Our purpose is to make the ideas deriving from the research on range search available in a more integrated and simpler way, to people interested in the discovery of more suitable and efficient methods for problems in theoretical computer science as well as to those interested in the applications of these ideas.

A wide study of the solutions found in the literature shows many conceptual similarities in the employed methods. Frequently, the same approaches and techniques are seen in distinct situations. These general purpose approaches and techniques are called "algorithm paradigms". The study and application of these paradigms allow a certain level of generalization of the solutions to range search problems, because they allow one to perceive several solutions of various instances of a general problem as the manifestation of the same rationale. The study of algorithm paradigms is instructive in its own right, since it propitiates the development of systematic reasoning, useful in the solution of many problems in computer science.

The contents herein are arranged so as to first give the theoretical basis necessary to understanding the methods given later. In chapter 1, we provide the basic concepts and classifications related to search problems in general and to range search in particular, and establish the scope of our research. In chapter 2, we describe some algorithm paradigms applied to range search problems, with the purpose of supplying the reader with alternative ways of establishing connections among the solutions presented later, leading him to develop a reasoning that allows the identification of the fundamentals and techniques shared by the solutions. In chapters 3 to 6, we deal with the variations of the range search problem characterized by the classical shapes of ranges considered in the literature. These chapters are arranged in a convenient way in order to reflect the complexity of the discussed solutions, their nature and the historical evolution. In each one of these chapters the problems are discussed in detail, some solutions and lower bounds are briefly described and bibliographic notes containing references to specific subjects are presented. Finally, in chapter 7, we summarize the contributions of this work and extensions that can be undertaken in the future.

Conteúdo

Capítulo 1

Introdução

Problemas de busca aparecem freqüentemente em computação, onde diversos problemas envolvem operações de busca ou podem ser vistos como instâncias desta classe de problemas.

Muitas aplicações, particularmente as relacionadas com bancos de dados [Cox91] e projeto auxiliado por computador (*CAD*), podem se beneficiar da utilização de estruturas (e/ou métodos) que possibilitem efetuar buscas por diversos atributos (*multikey access*) e/ou acesso a dados geométricos em várias dimensões, eficientemente, no que diz respeito aos tempos de busca, atualização e pré-processamento, bem como à quantidade de memória utilizada para armazenar a estrutura pré-processada empregada para suportar as buscas. Há atualmente grande número de estruturas proporcionando tais tipos de acesso e prosseguem as pesquisas tanto com o objetivo de medir o desempenho destas (teórico e empírico) quanto visando a obtenção de novas estruturas mais adequadas e eficientes para diversas variações de problemas de busca.

Na área de pesquisa de algoritmos eficientes têm sido propostas diversas formulações para problemas de busca e concebidas diversas soluções para formulações específicas, muitas delas baseadas em abordagens e técnicas comuns a outras soluções. Algumas das idéias provenientes da pesquisa de algoritmos, além da possibilidade de serem aperfeiçoadas e aplicadas para acesso a bases de dados, têm relevância também na teoria da computação, devido ao suporte que oferecem à solução de outros problemas na própria área de algoritmos e em computação gráfica. Além disso, o estudo das abordagens e técnicas empregadas nas soluções de problemas de busca é instrutivo por si só, pois muitas delas têm ampla aplicação e permitem desenvolver raciocínios sistemáticos para solucionar diversos problemas em computação.

Entretanto, faltam trabalhos que congreguem as soluções concebidas para os muitos subproblemas e que permitam compará-las, não necessariamente para determinar as soluções mais adequadas para aplicações particulares, mas também para identificar as abordagens empregadas e as características em comum dessas soluções. Particularmente, no campo de pesquisa de algoritmos identificamos uma carência de publicações recentes que descrevam as diversas soluções que têm sido concebidas de forma integrada e acessível aos interessados em sua aplicação, permitindo compará-las do ponto de vista de sua natureza e funcionamento. É justamente este o

enfoque que exploramos aqui.

1.1 Objetivo do trabalho

O tema central deste trabalho é o problema de busca em subespaços (*range search*)[1] (veja definição na seção 1.2), sob o ponto de vista de projeto de algoritmos eficientes e geometria computacional. O objetivo é oferecer um compêndio integrando várias formulações e soluções encontradas na literatura para diversas variações do problema, procurando colocá-las sob uma mesma ótica e com notação uniforme de modo a facilitar comparações e um estudo mais detalhado. Pretende-se, entre outros usos, que esta resenha seja útil para apoiar o desenvolvimento de outros trabalhos no futuro, por exemplo, no tocante a algum sub-problema específico ou a comparações de eficiência nas implementações. Esta tese deve fornecer parte da base teórica necessária e um levantamento, com um estudo preliminar, das soluções propostas na literatura.

O presente texto é destinado principalmente às pessoas interessadas no projeto e funcionamento dos algoritmos e estruturas de dados, mas pretende-se que seja útil também aos envolvidos com áreas de aplicação. Ele visa propiciar um estudo mais completo e compreensível de um subconjunto representativo dos problemas de busca em subespaço em várias dimensões e suas soluções.

Uma vez que pretendemos que o trabalho sirva de compêndio, oferecemos farta quantidade de referências. Uma das contribuições deste trabalho é justamente levantar e organizar a bibliografia de forma a apresentar, da forma mais clara possível, o que tem sido pesquisado na área e facilitar o estudo de assuntos específicos. Cabe salientar, entretanto, que alguns trabalhos são citados aqui na esperança de que sejam úteis aos interessados em assuntos específicos.

Neste primeiro capítulo, nos ocupamos das informações preliminares, definições e classificações relativas aos assuntos a serem tratados em mais detalhes posteriormente, de modo a prover os conceitos básicos necessários à compreensão dos capítulos seguintes. Também descrevemos possíveis extensões do tema central e damos uma noção mais precisa da abrangência da tese e do enfoque escolhido.

1.2 Caracterização do problema

1.2.1 Problemas de busca

Problemas de busca são fartamente encontrados na literatura. Definições para problemas de busca em geral podem ser encontradas em [Knu73, Ben79a, Mel84, Sed88, IRKV88]. Este tipo de problema envolve uma base de dados a respeito dos quais deseja-se extrair alguma informação ou saber se satisfazem determinadas propriedades.

[1] Como não há uma tradução estabelecida na literatura em português para o termo *range*, das traduções de *range search*: "busca em sub-região" e "busca em subespaços", preferimos esta última.

Há duas possíveis formas de efetuar consultas relativas a problemas de busca:

Consultas unitárias (*single shot queries*): A consulta é feita *somente uma vez* para uma dada base de dados, dispensando qualquer tipo de pré-processamento que pudesse agilizar as buscas.

Exemplos: calcular a dominância dos pontos de um conjunto (seção 2.1), conjunto dominante, par mais próximo (enunciados destes problemas podem ser encontrados em [PS85]).

Consultas em modo repetitivo (*repetitive-mode queries*): São realizadas *repetidas consultas* sobre uma mesma coleção de dados, cada qual com relação a um *objeto de consulta*, obtido de um conjunto de objetos de consulta possivelmente infinito. Em cada consulta, deseja-se identificar (ou simplesmente contar) os objetos da base de dados satisfazendo uma dada propriedade com relação ao objeto de consulta. Pode ser útil arranjar a informação em uma estrutura organizada e mantê-la armazenada, a fim de facilitar as buscas, o que, se realizado, implica num custo de memória e pré-processamento (e de reorganização, no caso de estruturas dinâmicas). Esses custos são compensados pela agilização dos tempos das buscas, que são realizadas repetidas vezes.

Exemplos: busca em subespaços (*range search*), dominância de um dado ponto, k vizinhos mais próximos a um dado ponto (capítulo 4).

A análise de desempenho dos métodos de busca *com consultas em modo repetitivo* para um conjunto P de N objetos de dados armazenados, usualmente leva em consideração quatro (muitas vezes somente as três primeiras: veja seção 1.4.3) medidas de custo distintas enumeradas abaixo como funções de N:

1. *C:* tempo de consulta: o tempo de processamento de uma consulta;
2. *M:* memória: a quantidade de espaço de memória requerida para armazenar a estrutura de dados pré-processada destinada a suportar as buscas;
3. *P:* tempo de pré-processamento: o tempo necessário para montar a estrutura de dados;
4. *A:* tempo de atualização: pode ser o tempo despendido para inserir um elemento (*I:*), o tempo para remover um elemento (*R:*) ou o tempo para atualizar (reorganizar) a estrutura após um certo número de inserções e remoções (*A:*).

As análises das complexidades acima podem ser tanto de pior caso quanto de caso médio e neste trabalho as medidas de complexidade dizem respeito ao pior caso, a não ser quando especificado de outro modo.

O tempo de consulta (*query time*), no caso de problemas de enumeração (veja seção 1.2.2) costuma ser denotado por $(f(N) + k)$ onde $f(N)$, denominado tempo de busca (*search time*), é o tempo despendido para procurar na estrutura pré-processada os elementos satisfazendo à propriedade em questão e k é o tamanho da saída, isto é, a quantidade de elementos satisfazendo à

propriedade e portanto, o tempo mínimo necessário para retorná-la (*report time*). Esta separação permite contornar o limite inferior trivial de pior caso $O(N)$ para o tempo total de consulta, possibilitando uma comparação mais acurada entre os métodos nas situações em que $k \in o(N)$.

As duas medidas de complexidade mais importantes em grande parte das situações são o tempo de consulta e o uso de memória. Os diversos métodos existentes para um mesmo problema costumam proporcionar diversas relações distintas entre essas grandezas. Assim, é comum referir-se aos métodos com tempo de consulta e uso de memória proporcionais a $f(N)+k$ e $g(N)$, respectivamente, como sendo de ordem $O(f(N)+k, g(N))$. O tempo de pré-processamento algumas vezes pode ser desprezado, por este ser efetuado uma única vez, ao passo que a memória dispensada para a estrutura pré-processada fica comprometida durante todo o tempo em que se desejar efetuar consultas.

1.2.2 O problema de busca em subespaço

Um desafio persistente na área de problemas de busca é conseguir soluções eficientes para problemas de busca em várias dimensões. Dois termos comumente empregados para designar busca em várias dimensões são busca multidimensional (*multidimensional search*) [DL76] e busca geométrica (*geometric retrieval*) [CY85, PS85].

Este trabalho trata especificamente de alguns problemas de busca em várias dimensões denominados problemas de busca em subespaços (*range search*). Consideramos o caso em que os objetos de dados são *pontos* dispersos no espaço multidimensional. Na versão *consultas em modo repetitivo*, um problema de busca em subespaços pode ser formalmente enunciado da seguinte maneira:

Busca em subespaço:

Sejam $\gamma_1, \gamma_2, \ldots, \gamma_d$ espaços topológicos totalmente ordenados.

Seja $\Gamma = \gamma_1 \times \gamma_2 \times \ldots \times \gamma_d$ o produto cartesiano desses espaços.

Seja $\Delta_N \subseteq \binom{\Gamma}{N}$ uma família de bases de dados admissíveis de tamanho N, definida através de alguma coleção de equações ou inequações algébricas.

Seja $P = \{p_1, p_2, \cdots, p_N\}$ uma base de dados de Δ_N. Denota-se um elemento $p \in P$ por $p = (x_1, x_2, \ldots, x_d)$ com $x_i \in \gamma_i$ $(1 \leq i \leq d)$.

Seja $\Sigma \subseteq 2^\Gamma$ uma família de subespaços de busca, definida através de alguma coleção de predicados.

Deseja-se pré-processar P de modo a responder eficientemente a consultas relativas a quais pontos de P estão contidos em um subespaço de busca $\sigma \in \Sigma$.

Os espaços topológicos $\gamma_1, \gamma_2, \ldots, \gamma_d$ são os espaços coordenadas do espaço d-dimensional Γ, de onde são tomados os pontos da base de dados (d-tuplas de valores em $\gamma_1 \times \gamma_2 \times \ldots \times \gamma_d$). Esses pontos podem também estar restritos, por exemplo, a alguma variedade algébrica mergulhada no espaço Γ. Esta condição pode ser estabelecida por meio de restrições na família de bases de dados Δ_N. Os subespaços de busca utilizados nas consultas são sub-regiões do espaço Γ, cuja natureza é determinada pelas restrições impostas em Σ.

Os tipos de respostas a serem fornecidas para problemas de busca em subespaços permitem uma classificação das consultas em três tipos:

Consultas de enumeração (*report queries*): Enumerar os pontos em $P \cap \sigma$.

Consultas de contagem (*count queries*): Determinar o número de pontos em $P \cap \sigma$ ($|P \cap \sigma|$).

Consultas de vacuosidade (*emptiness queries*): Determinar se algum ponto de P está contido em σ ($P \cap \sigma \neq \emptyset$?).

Definição generalizada

Usualmente, o problema de busca em subespaço é investigado em uma definição mais geral, a qual consiste em assumir funções peso $w : P \rightarrow (S, +)$, mapeando os pontos da base de dados P em um semi-grupo aditivo $(S, +)$, e perguntar o peso acumulado dos elementos da base de dados contidos no subespaço de busca $\left(\sum_{p \in (P \cap \sigma)} w(p)\right)$. O semi-grupo $(\mathbf{Z}, +)$ pode ser usado para consultas de contagem e o semi-grupo $(2^P, \cup)$ é adequado para consultas em modo de enumeração. Esta generalização é muito utilizada para provar cotas inferiores para a complexidade dos problemas (seção 1.5), mas tem também outras vantagens. Por exemplo, dados N pontos no espaço tridimensional, encontrar o ponto mais alto em uma região ortogonal de busca no plano xy pode ser encarado como um problema de busca no plano, onde as alturas dos pontos (pesos) são mapeadas no semi-grupo $(\mathbf{R}, max)$.

Dimensão de Vapnik-Chervonenkis

O conjunto de subespaços de busca Σ é o principal determinante de um problema de busca em subespaços. A cada par (P, Σ) corresponde uma coleção $Sub(P, \Sigma) = \{\mathcal{P} = P \cap \sigma \mid \sigma \in \Sigma\}$, composta de todos os subconjuntos de P satisfazendo alguma consulta relativa ao conjunto de subespaços de busca[2] Σ. Uma vez definido o par[3] (P, Σ), pode-se pré-processar o conjunto de pontos P, produzindo uma estrutura de dados T para auxiliar a responder consultas para qualquer $\sigma \in \Sigma$. No fundo, o que a estrutura T deve possibilitar é a determinação eficiente, para cada consulta dada, do elemento de $Sub(P, \Sigma)$ que denota a resposta da consulta.

[2]Observe que se todo $\mathcal{P} \subseteq P$ puder ser obtido pela interseção de P com algum subespaço de Σ ($\mathcal{P} = P \cap \sigma \mid \sigma \in \Sigma$), então $|Sub(P, \Sigma)| = 2^N$.

[3]O par (P, Σ) é tratado na literatura com o nome de *range space*.

A dimensão de Vapnik-Chervonenkis ou dimensão-*VC* (*VC-dimension*) [AHW87] é uma característica importante de um problema de busca em subespaço determinado pelo conjunto de subespaços de busca Σ. Ela é denotada neste trabalho por *VC* e equivale ao maior valor de cardinalidade tal que qualquer subconjunto de um conjunto $Q \in \Gamma$ desta cardinalidade pertence à coleção $Sub(\Gamma, \Sigma)$, isto é, pode ser obtido da interseção de Γ com algum subespaço de busca $\sigma \in \Sigma$. Prova-se [AHW87] que qualquer problema de busca em subespaço com dimensão-*VC* igual a 1 permite consultas em tempo $o(N + k)$, onde N é o número de pontos na base de dados P e k é o tamanho da saída. Busca em tempo sub-linear é possível também para muitos problemas de busca em sub-espaços com dimensão-*VC* finita.

Os formatos de subespaços de busca clássicos

Há várias formas de se descrever um subespaço de busca σ. Por exemplo, ele pode ser expresso por um conjunto de pontos (com o valor de todas ou apenas algumas das coordenadas especificados) ou por restrições como as do tipo: $a_i \leq x_i \leq b_i$, onde $a_i, b_i \in \gamma_i$ e $1 \leq i \leq d$.

Os problemas de busca em subespaço podem também ser classificados de acordo com os formatos dos subespaços de busca σ permitidos. Entre os problemas clássicos encontrados na literatura para $\gamma_i = \mathbf{R}$ $(1 \leq i \leq d)$ encontram-se (veja ilustrações na figura 1.1):

- busca em subespaço ortogonal: É também conhecida por busca em subespaço hiper-retangular (*hiper-rectangular range*) . O subespaço de busca é $\sigma = \sigma_1 \times \sigma_2 \times \ldots \times \sigma_d$, o produto cartesiano dos intervalos $\sigma_i = [a_i, b_i] \subseteq \gamma_i$ $(1 \leq i \leq d)$, denotando um "hiper-retângulo", com as arestas paralelas aos eixos coordenados (figura 1.1-a).

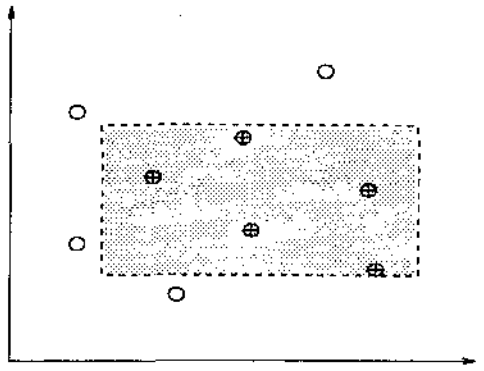
- busca em subespaço circular: O subespaço de busca é um "hiper-disco" $\sigma(o, r)$ de centro o e raio r arbitrários[4] (figura 1.1-b).

- busca em subespaço em forma de trapézio (*slanted range search*): Este sub-problema é definido em $\mathbf{R}^2$. O subespaço de busca é um trapézio com a base apoiada no eixo x mais o seu interior (figura 1.1-c).

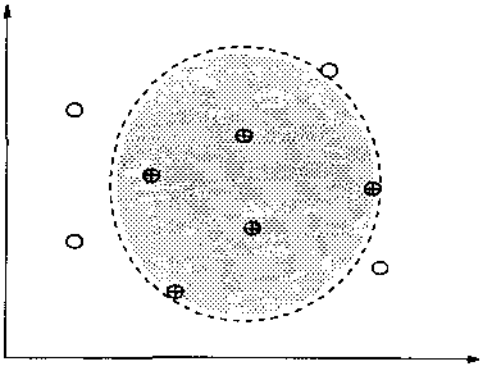
- busca em subespaço poligonal: Este sub-problema também é definido em $\mathbf{R}^2$. O subespaço de busca é uma região poligonal, isto é, uma região cuja fronteira é constituída de uma seqüência de segmentos de reta ou raios mas, não necessariamente limitada em todas as direções (figura 1.1-d).

- busca em semi-espaço (*halfspace range search*): O subespaço de busca é um semi-espaço $\sigma(h)$ do espaço Γ, isto é, uma região do espaço d-dimensional limitada por um hiperplano de dimensão $(d-1)$. A figura 1.1-e ilustra um semi-espaço em $\mathbf{R}^2$.

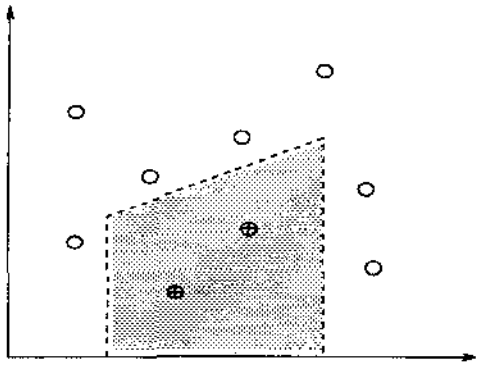
[4] Busca em subespaço circular (*circular range search*) usualmente refere-se a este problema em duas dimensões. Embora a maioria dos resultados existentes digam respeito ao caso bidimensional, neste trabalho, usamos o termo busca em subespaço circular para designar o problema em dimensão arbitrária.



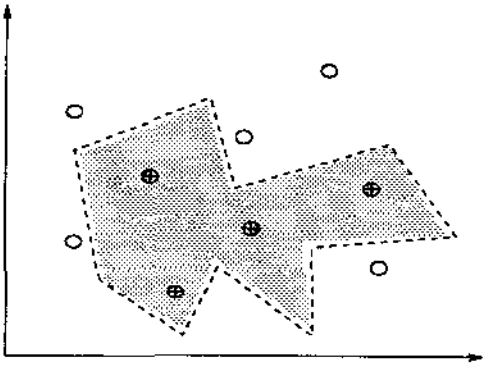
(a) - ortogonal



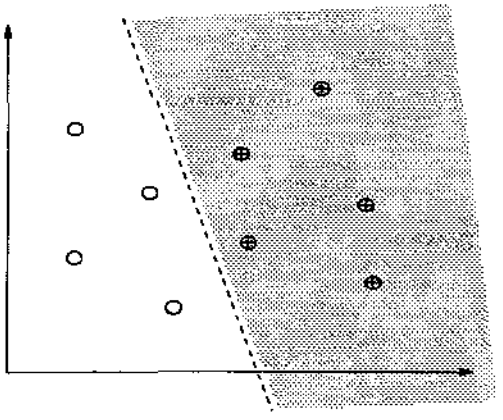
(b) - circular



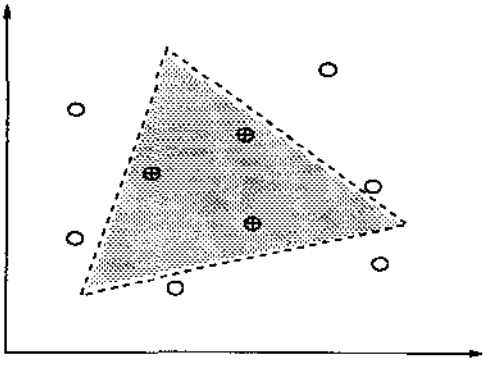
(c) - trapézio



(d) - poligonal



(e) - semi-espaço



(f) - simplexo

Figura 1.1: Tipos de subespaço de busca no plano euclideano.

- busca em subespaço em forma de simplexo (*simplex range search*): O subespaço de busca é delimitado por um simplexo[5]. A figura 1.1-f ilustra um simplexo em $\mathbf{R}^2$ (um triângulo), o qual pode ser visto como a interseção de $d + 1 = 3$ semi-espaços (semi-planos).

O subespaço de busca ortogonal representa a forma mais simples do problema, pois neste caso o espaço Γ pode ser decomposto em faixas para as quais as buscas podem ser realizadas mais facilmente. O problema torna-se cada vez mais difícil de ser solucionado a medida que se aumenta a complexidade dos subespaços de busca.

Busca em subespaço circular é um problema mais difícil de ser tratado, pelo fato de sua fronteira ser não linear. O problema de busca em subespaço circular envolve proximidade e está intimamente relacionado com o problema de determinar os vizinhos mais próximos de um dado ponto, de modo que existem métodos para solucionar ambos os problemas utilizando diagramas de Voronoi (capítulo 4).

O problema de busca em subespaço em forma de trapézio no plano é também bastante simples, sendo comparável em simplicidade ao problema de busca em subespaço ortogonal. Problemas de busca em subespaços em forma de trapézios e outras figuras geométricas simples no plano (retângulos e triângulos, por exemplo) se tornam mais importantes ao considerar reduções de busca em regiões poligonais a coleções de buscas nestas figuras mais simples, que podem ser tratadas eficientemente. Todavia, nem toda região poligonal pode ser particionada em um número limitado de trapézios.

Busca em região poligonal é uma restrição ao plano do problema de busca em poliedros de dimensão arbitrária. Alguns dos primeiros métodos para busca em semi-espaços e simplexos em dimensão arbitrária surgiram da generalização de soluções do problema de busca em subespaço poligonal.

Finalmente, cabe observar que busca em simplexos tem conexão direta com busca em semi-espaços, pois um simplexo pode ser visto como a interseção de semi-espaços. Freqüentemente, as mesmas soluções se aplicam a ambos os problemas.

As observações acima demonstram algumas das conexões entre os problemas de busca em subespaços dos formatos ilustrados na figura 1.1. Analisando as soluções destes problemas encontradas na literatura, pode-se ainda identificar uma série de aspectos semelhantes, resultando em abordagens e técnicas aplicáveis em muitas situações. Entretanto, não é comum encontrar métodos para busca em subespaços de forma genérica e, mesmo quando estes são possíveis, eles são menos eficientes e adequados em situações específicas que os métodos baseados em restrições nos formatos dos subespaços de busca.

---

[5] Um δ-simplexo em $\mathbf{R}^d$ ($\delta \leq d$) é um conjunto δ-dimensional formado pela envoltória convexa de $\delta + 1$ pontos $\{q_0, q_1, \ldots, q_\delta\}$, seus vértices, que definem $\delta + 1$ vetores linearmente independentes. O interior de um δ-simplexo é o conjunto de todos os pontos da forma $\sum_{i=0}^{\delta} (\alpha_i \cdot q_i)$; sendo $\sum_{i=0}^{\delta} \alpha_i = 1$ e $\alpha_i > 0$ para todo $0 \leq i \leq \delta$. Um δ-simplexo pode ser considerado como a interseção de $(\delta + 1)$ semi-espaços onde os hiperplanos limitantes estão em posição geral. Quando $\delta = d$ chamamos o δ-simplexo simplesmente de simplexo.

Subespaços polinomiais e semi-algébricos

Um formato de subespaço de busca mais genérico, englobando os formatos descritos acima e que recentemente começou a ser considerado na literatura [AM92a, Mat93a] é o dos subespaços determinados por inequações polinomiais. Os subespaços polinomiais são aqueles determinados por uma única inequação polinomial:

Subespaços polinomiais:

Seja $f(x,a) = f(x_1,\cdots,x_d,a_1,\cdots,a_K)$ um polinômio de grau limitado em $d+K$ variáveis, onde $x = (x_1,x_2,\cdots,x_d)$ é um ponto no espaço $\mathbf{R}^d$ e $a = (a_1,a_2,\cdots,a_K)$ é um vetor de K parâmetros, sendo K uma constante.
O conjunto Σ_f de subespaços de busca polinomiais determinados por f é dado por $\Sigma_f = \{\sigma_f(a) \mid a \in \mathbf{R}^K\}$, onde $\sigma_f(a) = \{x \in \mathbf{R}^d \mid f(x,a) \geq 0\}$.

O polinômio f especifica os tipos de subespaços de busca a serem considerados (por exemplo, discos, cônicas, cilindros, etc.) e o vetor de parâmetros a determina um subespaço particular do tipo considerado.

Por meio de conjunções, disjunções e complementos de inequações polinomiais, obtém-se diversos formatos de subespaços, que são denominados **subespaços semi-algébricos**[6]. O conjunto dos subespaços semi-algébricos admite obviamente subespaços com fronteira não linear, como é o caso dos subespaços circulares. Portanto, uma solução para busca em subespaços semi-algébricos se aplica a uma ampla variedade de subespaços de busca. Todavia, é importante que os subespaços de busca tenham **complexidade de descrição limitada**, isto é, que possam ser descritos por um número limitado de polinômios de grau limitado.

Os formatos básicos de subespaços de busca considerados neste trabalho são ortogonal (capítulo 3), circular (capítulo 4), poligonal (capítulo 5) e por último semi-espaços e simplexos (capítulo 6). Em cada um desses capítulos são apresentadas descrições de algumas das soluções e sintetizados os limites inferiores obtidos para o sub-problema em questão. Por razões de espaço e para manter a discussão a um nível que permita um entendimento global, omitimos muitos detalhes. O leitor interessado em assuntos específicos pode recorrer às referências encontradas no decorrer do texto e nas notas bibliográficas de cada capítulo.

Soluções para busca em subespaços polinomiais e semi algébricos podem ser obtidas a partir de soluções originalmente propostas para busca em semi-espaços e simplexos, por meio da linearizaçao de subespaços de busca (seção 2.8), através da qual uma busca em subespaço polinomial

---

[6] Um conjunto semi-algébrico (*semialgebraic set*) em $\mathbf{R}^d$ é definido formalmente como a região obtida a partir de uma quantidade finita de conjuntos da forma $\{x \in \mathbf{R}^d \mid f(x) \geq 0\}$, onde $f(x)$ é um polinômio com d variáveis (as coordenadas do ponto x) e coeficientes reais, mediante a aplicação de operações básicas da teoria dos conjuntos: união, interseção e complemento. Os conjuntos semi-algébricos definidos por um número constante de polinômios de grau constante são chamados células de Tarski.

(inclusive busca em subespaço circular) pode ser reduzida a uma busca em *semi-espaço* numa dimensão maior.

Para dar uma idéia do potencial de aplicação das soluções para problemas de busca em subespaços em geral, basta notar que uma consulta a uma base de dados, expressa por um conjunto de restrições nos atributos pode ser vista como uma busca em subespaço, de formato determinado pela natureza das restrições, se convenientemente mapeada num sistema de coordenadas.

1.3 Abordagens e técnicas gerais na solução de problemas de busca em subespaços

Uma das contribuições pretendidas com este trabalho é a apresentação dos assuntos de uma forma abrangente, explorando aspectos gerais das soluções, a fim de identificar técnicas o mais genéricas possível. Assim, não nos limitamos a tratar as soluções propostas isoladamente, mas procuramos abordá-las dando ênfase às características comuns entre elas e procurando manter uma visão unificadora. A fim de auxiliar nesta tarefa e com o intuito de obter soluções mais sistemáticas, realçamos algumas abordagens e técnicas gerais empregadas nas soluções de problemas de busca encontradas na literatura. Ao conjunto dessas abordagens e técnicas gerais damos o nome de paradigmas de algoritmos.

Os paradigmas de algoritmos possibilitam um certo grau de sistematização das soluções de problemas de busca em subespaços, uma vez que permitem encarar diversas soluções distintas de diversos problemas como variações de uma mesma concepção básica e promovem a aplicação de raciocínios idênticos ou semelhantes em muitas situações. Algumas das soluções assintoticamente ótimas e soluções generalizadas para diversos tipos de problemas foram alcançadas com o emprego (nem sempre explicitado) de abordagens e técnicas gerais.

No capítulo 2, descrevemos alguns paradigmas de algoritmos empregados em soluções de problemas de busca em subespaços, de modo a prover ao leitor um embasamento que lhe permita identificar com mais facilidade os aspectos em comum das soluções.

1.4 Assuntos relacionados

O objetivo central deste trabalho é estudar problemas de busca em subespaços e métodos de solução, para várias dimensões e vários formatos de subespaços de busca. Entretanto, não se pode considerar este problema totalmente separado de outros problemas de busca, uma vez que estes estão bastante relacionados. Além disso, há questões que podem ser levantadas a respeito das variações do problema consideradas neste trabalho, das operações a serem suportadas e mesmo questões relativas à aplicação prática e à implementação das soluções. A seguir, discutimos tais assuntos e a que profundidade eles são tratados neste trabalho, fornecendo algumas referências para as questões específicas.

1.4.1 Outros problemas de busca

Há muitos tipos de problemas de busca. Além de busca em subespaços, onde o par (consulta, itens da base de dados) é do tipo (região, pontos), tem-se localização de pontos, onde o referido par é do tipo (ponto, regiões) e muitos outros problemas envolvendo formas geométricas distintas como polígonos e segmentos.

Cabe salientar que os diversos tipos de problemas de busca estão bastante relacionados. O surgimento de uma solução eficiente para um determinado problema pode levar a aperfeiçoamentos nas soluções de outros problemas, quer pela aplicação de técnicas concebidas juntamente com a nova solução, quer pela utilização do novo algoritmo como "sub-rotina" para compor a solução de outros problemas. Assim, consideramos resultados obtidos em assuntos como:

- localização de pontos (*point location/enclosure*),
- vizinhos mais próximos,
- interseção de segmentos,
- dominância de um conjunto de pontos,
- decomposição de polígonos,

entre outros, quando estes puderem ser úteis em soluções para problemas de busca em subespaços. Porém, neste trabalho, estamos mais interessados em utilizar os resultados relevantes para os nossos propósitos. O estudo de certos problemas poderia até justificar a produção de outros trabalhos com objetivos semelhantes aos deste. Na medida do possível, oferecemos referências adicionais no decorrer do texto, relativas a resultados empregados e correlatos.

1.4.2 Tipos de objetos de dados

Considere estender as formas de objetos de dados permitidos na base de dados P de modo a incluir outras entidades geométricas além de pontos. Neste caso, as formas dos objetos de dados permitidos em P constituem uma variável importante dos problemas de busca em subespaços, da mesma forma que os formatos dos subespaços de busca permitidos em Σ. Uma classificação possível para os métodos de solução de problemas de busca em subespaços, sob o ponto de vista dos tipos dos objetos permitidos em P, consiste em dividí-los em duas classes [Cox91]:

MAOP's: **Métodos de acesso a objetos pontuais** (*Point Access Methods*): Os objetos de dados armazenados são pontos num espaço multidimensional e têm *dimensão nula*.

MAON's: **Métodos de acesso a objetos NÃO pontuais** (*Spatial Access Methods*): Os objetos de dados são formas geométricas de *dimensão não nula* no espaço multidimensional.

Métodos de acesso eficientes para dados não pontuais podem ser de grande valia em aplicações a bancos de dados e projeto auxiliado por computador (*CAD*). No entanto, neste trabalho, restringimos o estudo somente a métodos de busca em subespaços considerando *dados pontuais* (*MAOP's*), a fim de facilitar a manipulação de generalizações em outros aspectos do problema, mantendo a complexidade em um nível aceitável. As razões expostas a seguir justificam esta restrição do ponto de vista da aplicabilidade das soluções estudadas:

1. Alguns dos *MAON's* conhecidos, para objetos espaciais simples (retângulos, intervalos, etc.), são baseadas em algum *MAOP* subjacente [KSSS], ou seja, alguns *MAOP's* podem ser aperfeiçoados e transformados em *MAON's*. Assim, quanto melhor o desempenho do *MAOP* subjacente melhor o *MAON* resultante, donde a importância de se pesquisar *MAOP's* eficientes.

   Existem também estratégias sistemáticas para transformação de objetos de dimensão não nula, em pontos num espaço de dimensão maior [Cox91]; obviamente a um certo custo de processamento.

   Assim, como é mencionado em [Cox91], não há uma distinção rigorosa entre *MAOP's* e *MAON's*, no sentido que métodos de acesso a pontos podem ser utilizados para objetos de dados não pontuais e vice-versa.

   Contudo, não se pode afirmar que todas as soluções de problemas relacionados com dados pontuais sejam extensíveis para problemas envolvendo objetos de dados não pontuais a um custo viável. Além disso, considerando objetos geométricos de dimensão não nula aparecem outros tipos de problemas (como, por exemplo, interseção e tangência).

2. A generalização dos formatos dos subespaços de busca (a qual podemos pesquisar mais profundamente, fixando o tipo dos objetos de dados na forma mais simples) apresenta-se atrativa no que diz respeito à sua potencial aplicação. Como mencionamos no final da seção 1.2.2, as consultas a bases de dados convencionais equivalem a buscas em subespaços de diversos formatos, quando mapeadas convenientemente numa representação em algum sistema de coordenadas.

1.4.3 Dinamização

Dinamização consiste em permitir modificações na composição do conjunto de dados armazenados P, suportando eficientemente uma seqüência arbitrária de operações de inserção e remoção intercaladas com operações de busca. Isso requer reorganizar a estrutura de dados durante as operações de atualização ou após um determinado número delas, a fim de evitar a degeneração da estrutura e a conseqüente degradação do desempenho. Estruturas dinâmicas envolvem outras medidas de desempenho (seção 1.2.2). Além das medidas usuais temos o tempo de inserção (I:), o tempo de remoção (R) e/ou o tempo de reorganização da estrutura (A:) após um certo número de atualizações. Muitas vezes não há pré-processamento propriamente dito, mas uma seqüência de inserções iniciais.

Há uma classificação das estruturas de dados quanto ao grau de dinamização. Uma estrutura de dados é dita ser **semi-dinâmica** se suporta somente inserções ou somente remoções e **totalmente dinâmica** se suporta ambas, inserções e remoções, intercaladas com consultas.

As árvores de busca balanceadas (árvores *AVL*, *B-Trees*, etc.) são exemplos de estruturas dinâmicas para busca unidimensional e permitem solucionar o problema de forma bastante satisfatória, com inserções, remoções e buscas em tempo ótimo[7] $O(\log N)$ e uso de memória $O(N)$. Entretanto, as estruturas para suportar buscas em subespaços são usualmente estáticas. Algumas destas estruturas permitem dinamização, muitas vezes mediante um fator de custo adicional nas consultas ou maior uso de memória. Soluções dinâmicas ou semi-dinâmicas são às vezes encontradas nos mesmos artigos que propõem as estruturas estáticas básicas.

Existem também técnicas gerais para dinamização de estruturas de dados, as quais são usualmente aplicáveis somente a problemas ou estruturas de dados que satisfaçam certas restrições. Técnicas e outras questões relacionadas com dinamização, além de muitas referências adicionais, podem ser encontradas em [Ove83, CT92].

1.4.4 Buscas no passado

Uma estrutura de dados dinâmica apresenta diversas configurações no decorrer do tempo. Isso abre a possibilidade de consultas relativas à configuração da estrutura em algum instante do passado. Uma estrutura de dados que permite realizar consultas como se estivesse em algum instante anterior é dita suportar buscas no passado (*search in history, search in the past*). Para isso é necessário manter informações a respeito de como a base de dados muda no decorrer do tempo sob efeito de atualizações (inserções e remoções de objetos de dados). Alguns algoritmos e estruturas de dados provenientes da geometria computacional levando a soluções para este problema podem ser encontrados em[8] [Ove81a, Ove81b, Ove83, DM85, Cha85a].

1.4.5 Armazenamento das estruturas em memória secundária

As estruturas de dados pré-processadas para suportar consultas eficientes, mesmo as estruturas de índices por uma única chave, da mesma forma que as próprias bases de dados, costumam ser grandes demais para caber inteiramente na memória principal e às vezes é adequado serem persistentes, para não terem de ser reconstruídas após cada interrupção de funcionamento do sistema de computação. Então, estas estruturas precisam ser armazenadas em memória secundária.

O armazenamento em memória secundária requer a divisão da estrutura de dados em blocos de transferência de informações entre a memória principal e a secundária. É importante que esta divisão se faça de modo a minimizar o fluxo de informações necessário para realizar as operações de busca e de atualização, pois a transferência de dados entre memória principal e secundária se torna o gargalo do sistema nestas operações.

[7] Neste trabalho, considere os logaritmos com base não especificada como sendo base 2.

[8] Por limitações de tempo e espaço nao vamos dar ênfase aos assuntos acima (dinamização e buscas no passado). O leitor interessado pode recorrer à bibliografia citada.

A literatura para problemas de busca em subespaços, sob o enfoque de projeto de algoritmos eficientes é pobre no que diz respeito à adequação das estruturas para possibilitar seu armazenamento em memória secundária. Os trabalhos abordando este assunto que conseguimos levantar são [Sil81, IKO88, OSdBvK90, SO90].

1.4.6 Implementação

A implementação é parte essencial da pesquisa para soluções de problemas computacionais. Entretanto, em problemas de busca em subespaços a complexidade das diversas soluções propostas e o desconhecimento de quais são as mais viáveis do ponto de vista tanto de eficiência computacional quanto facilidade de implementação, versatilidade e outros critérios, torna necessário um estudo mais detalhado anteriormente à implementação. Devido à quantidade de trabalho envolvido em contraposição ao tempo limitado do qual dispomos, optamos por elaborar primeiro um compêndio, o qual possa servir de apoio ao prosseguimento das pesquisas que levem a implementações experimentais no futuro.

1.5 Modelos para análise de complexidade e prova de cotas inferiores

Muitas estruturas engenhosas têm sido propostas para solucionar problemas de busca em subespaço. Todavia, para diversas delas não há muita análise de desempenho disponível na literatura, embora muitas tenham bom desempenho no caso médio (o que usualmente procura-se demonstrar por meio de razões heurísticas e experimentos). Neste trabalho, estamos particularmente interessados em analisar o comportamento das estruturas e vamos nos ater àquelas para as quais seja possível uma análise de desempenho dos algoritmos razoavelmente precisa[9].

A fim de fechar os problemas, do ponto de vista teórico, devemos determinar se as soluções concebidas para os mesmos são ótimas ou se é possível haver melhores resultados em termos assintóticos. Então, é necessário trabalhar em dois sentidos. De um lado temos a pesquisa de soluções eficientes e adequadas para os diversos tipos de aplicações, oferecendo diversas relações de compromisso entre as medidas de desempenho. De outro, é necessário demonstrar cotas inferiores cada vez mais precisas, para os valores das medidas de desempenho possíveis de serem alcançados.

As análises de desempenho das soluções e de complexidade dos problemas (os limites inferiores inerentes dos mesmos, para quaisquer soluções) podem ser efetuadas segundo diversos modelos computacionais, baseados em diferentes formas de encarar os problemas e suas soluções.

[9] A experiência de pesquisas na área de algoritmos demonstra que, em parte das situações, as soluções com melhores desempenhos assintóticos são também as mais adequadas em situações práticas. Todavia, isso não ocorre sempre, isto é, há situações em que soluções inferiores do ponto de vista assintótico apresentam bons resultados no caso médio. Este é o caso, por exemplo, da estrutura *KD-Tree* (seção 3.1.1), que apresenta bom desempenho no caso médio, segundo demonstram diversas análises e experimentos [BS75, LW77, Sil78, Ben79b].

Cada modelo permite um determinado conjunto de operações básicas, associa a estas custos (normalmente unitários) e leva em consideração somente determinados tipos de operações no cálculo da complexidade.

Entre os modelos usualmente empregados para a análise de problemas de busca encontram-se o modelo aritmético, os modelos de árvore de computação e os modelos de máquinas de ponteiros (*Pointer Machines*) e de acesso aleatório (*RAM - Random Access Machines*). Os modelos de máquinas de ponteiros e de acesso aleatório foram originalmente propostos para avaliar a capacidade computacional das máquinas, isto é, quais operações podem realizar e com que medidas de eficiência. Entretanto, eles se desenvolveram em modelos para análise de complexidade de algoritmos e há muitas análises de complexidade de soluções e provas de cotas inferiores para problemas de busca em subespaços efetuados nestes modelos.

Além dos modelos para análise de complexidades citados acima, existem outros modelos e maneiras distintas de efetuar estas análises. Uma técnica muito usada para análise da complexidade de operações realizadas repetidas vezes é o método de análise da complexidade amortizada (veja seção 1.5.4).

1.5.1 Modelo aritmético

No modelo aritmético [Vai89], consideram-se funções peso $w : P \rightarrow S$ mapeando os elementos de dados em um semi-grupo aditivo. As consultas são relativas à "soma" no semi-grupo dos pesos associados aos elementos de dados satisfazendo o critério de busca. Este modelo encara a estrutura de dados pré-processada como um conjunto de "somas" pré-computadas no semi-grupo e leva em consideração somente a quantidade de operações aritméticas necessárias para responder à consulta. O modelo aritmético considera custo unitário para cada operação aritmética e ignora custos de recuperação de memória. Assim, ele faz um mínimo de suposições a respeito dos custos reais de um algoritmo o que o torna mais simples e atrativo. O modelo aritmético é bastante geral e particularmente adequado para prova de cotas inferiores.

1.5.2 Árvores de decisão

Nos modelos de árvore de decisão [BO83, PS85, Vai89] cada algoritmo é representado por uma árvore de decisão, onde os nós intermediários correspondem aos passos do algoritmo e as folhas representam as respostas possíveis. Uma entrada para o algoritmo determina um único caminho na árvore de decisão, partindo da raiz e indo até a folha corresponde à resposta para esta entrada. Diz-se que uma árvore de decisão **resolve** um determinado problema se para qualquer entrada possível para este problema a resposta retornada pelo algoritmo representado pela árvore é correta.

Às operações correspondentes aos nós intermediários de uma árvore de decisão são associados custos computacionais. O custo total $C(q, T)$ para responder uma consulta q em uma árvore de decisão T é a soma dos custos relativos aos nós de T contidos no caminho da raiz até a folha correspondente à resposta de q. A complexidade $C(T)$ de um algoritmo representado pela árvore

de decisão T é o máximo de $C(q,T)$ para qualquer entrada q e o limite inferior de um problema é dado pelo mínimo de $C(T)$ para todas as árvores de decisão que resolvem o problema.

Existem diversas variedades de árvores de decisão, caracterizadas pelas operações permitidas em seus nós intermediários. Estas operações permitidas por cada variação do modelo determinam sua capacidade de representação. Diz-se que um modelo de computação é **mais poderoso** que outro se permite representar todos os algoritmos que podem ser representados com o outro modelo e ainda mais alguns que o outro não permite. Um modelo mais poderoso também pode possibilitar soluções mais eficientes, uma vez que considera custos constantes associados a operações primitivas mais sofisticadas que as dos modelos mais fracos. Variações de modelos de árvores de decisão podem ser encontradas em [Avi80, SY82, BO83, PS85, Vai89].

1.5.3 Máquinas de ponteiros e de acesso aleatório

Os modelos de máquinas de ponteiros e de acesso aleatório [AHU74, PS85, Cha88] foram originalmente concebidos com o objetivo de avaliar a capacidade computacional de diferentes arquiteturas de computadores, isto é, quais operações (algoritmos) essas arquiteturas podem executar e com qual desempenho. Todavia, pode-se também utilizá-los de um ponto de vista diferente: determinar até quais restrições os algoritmos podem ser executados e as medidas de desempenho correspondentes.

A principal característica dos modelos do tipo **máquinas de ponteiros** (*pointer machines*) é proibir quaisquer tipos de cálculos de endereços. Considera-se alocação dinâmica de memória em unidades denominadas **células**. Cada célula de memória tem um endereço associado, representado por um ponteiro. Um **ponteiro** é simplesmente um nome simbólico, isto é, um endereço cuja representação particular é transparente e para o qual nenhuma operação aritmética é definida. Novas células de memória são obtidas de uma lista de células livres e devolvidas juntamente com os ponteiros para as mesmas após o uso. Somente ponteiros fornecidos pela lista de livres podem ser usados. Uma máquina de ponteiros elementar é a variedade mais simples de máquinas de ponteiros e permite efetuar somente operações de comparação e soma, com os valores armazenados nas células de memória. Variedades de máquinas de ponteiros mais poderosas permitem realizar operações mais sofisticadas [Cha88].

Uma **máquina de acesso aleatório** (*RAM - Random Access Machine*) é munida de comparações, das operações aritméticas usuais de soma, subtração, multiplicação e divisão e permite acesso aleatório às células de memória por meio de cálculos de endereços. Pode-se também considerar máquinas de acesso aleatório munidas de funções trigonométricas, exponenciais e logaritmos. Este tipo de máquina de acesso aleatório reflete os programas típicos em linguagens seqüenciais de alto nível.

1.5.4 Análise de complexidade amortizada

Em problemas de busca com consultas em modo repetitivo, muitas vezes, estamos interessados no custo total de uma seqüência de operações, ao invés dos custos das operações individuais.

Uma análise de pior caso de cada uma das operações pode ser pessimista demais, porque ignora efeitos correlacionados das operações na estrutura de dados. Por outro lado, uma análise de custo médio da seqüência de operações pode ser imprecisa, uma vez que as suposições probabilísticas necessárias para realizar a análise podem ser falsas. Seja o tempo médio para cada operação igual ao tempo total de pior caso da seqüência dividido pelo número de operações. Uma análise amortizada que leve em consideração este tempo médio para cada operação pode ser mais realística e robusta.

A fim de tornar a idéia mais concreta, vamos considerar um exemplo. Seja a manipulação de uma pilha por uma seqüência de operações compostas de dois tipos de primitivas de tempo de execução unitário: empilha (*PUSH*), que adiciona um novo item no topo da pilha e desempilha (*POP*), que remove e devolve o item do topo da pilha. Deseja-se analisar o tempo de execução de uma seqüência de operações, cada qual composta de zero ou mais operações primitivas do tipo desempilha seguidas de uma empilha. Suponha que se inicie com a pilha vazia e se realize m operações. Uma análise de pior caso mostra que uma única operação na seqüência pode gastar $O(m)$ unidades de tempo (considere a operação composta de uma seqüência de primitivas desempilha após $m-1$ operações empilha). Multiplicando pelo número de operações temos $O(m^2)$ operações primitivas, no pior caso. Todavia, todas as m operações juntas podem envolver no máximo $2m$ operações primitivas, uma vez que há apenas m empilha's ao todo e cada desempilha deve corresponder a uma primitiva empilha.

1.6 Cotas inferiores para busca em subespaços de forma geral

O limites inferiores popularmente conhecidos para o problema de busca com uma única chave, $\Omega(N)$ para o uso de memória e $\Omega(\log N)$ para o tempo de busca se aplicam também a busca em subespaços dos formatos mencionados na seção 1.2.2. Limites inferiores não triviais para os diversos tipos de problemas podem ser encontrados nos capítulos dedicados a cada um (capítulos 3 a 6).

Fredman [Fre81b] propõe um modelo de computação, baseado no modelo aritmético (seção 1.5.1), juntamente com uma técnica poderosa para provar limites inferiores para a complexidade de estruturas de dados ótimas que permitam inserções, remoções e buscas.

Aplicando tal técnica ao problema de busca em subespaço ortogonal (capítulo 3), por exemplo, conseguiu-se estabelecer o limite inferior [Fre81a]:

$$\Omega(N \log^d N) \tag{1.1}$$

para a complexidade de pior caso inerente ao processamento de uma seqüência de N operações de busca, inserção e remoção intercaladas, donde se deduz um custo amortizado poli-logarítmico por operação.

Aplicando a técnica de Fredman aos problemas de busca em subespaços circulares, busca em semi-espaços ou busca em regiões com fronteira em forma de parábola, todos em duas dimensões, é possível demonstrar o limite inferior [Fre81b]:

$$\Omega(N^{4/3}) \tag{1.2}$$

para a complexidade de pior caso da seqüência de N operações de busca, inserção e remoção intercaladas. Este limite é significativo, pois o custo amortizado $\Omega(N^{1/3})$ por operação é muito maior que o limite superior poli-logarítmico já alcançado por algumas soluções para o caso ortogonal, donde se conclui que o problema de busca em subespaço ortogonal é inerentemente mais simples que estes outros no caso dinâmico.

Chazelle [Cha90a] observa que remoção (ou alguma operação de atualização relacionada) é parte essencial do trabalho de Fredman. Disso resulta que problemas estáticos ou problemas permitindo somente inserções e buscas têm suas cotas inferiores super-estimadas quando utilizado o método de Fredman para prová-las, uma vez que não é possível omitir remoções no método. Trabalhos em dinamização como o de Overmars [Ove83] mostram que a coexistência de inserções e remoções freqüentemente é difícil de tratar. Intuitivamente, o que torna remoções dispendiosas é que uma única operação pode invalidar grandes porções da estrutura de dados, pois o semi-grupo não tem operação inversa.

1.7 Notas Bibliográficas

Existem alguns livros publicados nas áreas de algoritmos e geometria computacional contendo seções ou capítulos dedicados ao problema de busca em subespaço (*range search*). Entre eles podemos citar as obras de Mehlhorn [Mel84], Preparata e Shamos [PS85] e Sedgewick [Sed88], nos quais pode-se encontrar muitos conceitos básicos relacionados com o problema de busca em subespaço, algumas cotas inferiores estabelecidas e a descrição de soluções obtidas.

Capítulo 2

Paradigmas de algoritmos na solução de problemas de busca em subespaços

''Vor lauter Bäumen sieht man den Wald nicht.''

(De tanta árvore, não se vê a floresta.)

Provérbio Alemão

Um dos objetivos centrais deste trabalho é determinar em quais aspectos se relacionam os diversos métodos encontrados na literatura para a solução de problemas de busca em subespaços. Com este propósito, muitas vezes é útil recorrer a abordagens mais abrangentes, em busca de visões unificadoras que tornem possível adotar soluções mais sistemáticas ou que pelo menos, possibilitem uma melhor compreensão dos problemas e das soluções propostas. No estudo das soluções para determinada classe de problemas, costuma ser útil ter em mente meios alternativos de relacionar os diversos problemas e suas soluções. Com o enfoque de paradigmas de algoritmos, pode-se relacionar as soluções de concepções semelhantes e observar características interessantes que os problemas têm em comum, às quais seja possível aplicar técnicas ou raciocínios semelhantes.

A fim de fundamentar este enfoque, são descritos a seguir alguns dos paradigmas de algoritmos empregados nas soluções de problemas de busca em subespaços (e mesmo de outros problemas) encontrados na literatura. O objetivo é fornecer ao leitor uma visão geral das técnicas envolvidas e prover uma base teórica que lhe permita identificar fundamentos em comum.

2.1 A abordagem do lugar geométrico

O paradigma do lugar geométrico (*locus approach*) [CY85] tem por fundamento abordar um problema de busca dividindo as consultas possíveis em grupos com a mesma resposta ou cujas respostas possam ser calculadas eficientemente uma vez conhecido o grupo. Esta abordagem costuma propiciar buscas eficientes (em tempo ótimo ou próximo do ótimo), a custo de elevada utilização de memória.

Descrição

Imagine solucionar um problema de busca com consultas em modo repetitivo, através da pré-computação de todas as respostas possíveis. Seja o problema de busca em subespaços. Computar e armazenar as respostas das consultas para cada subespaço $\sigma \in \Sigma$ é claramente inviável para Σ grande ou infinito. Entretanto, em muitas situações, mesmo com Σ contendo muitos elementos (até uma quantidade infinita de subespaços), pode-se decompor o conjunto de consultas possíveis (ou espaço de consultas) em um número suficientemente pequeno de grupos de consultas com a mesma resposta, de modo que seja viável armazenar a resposta para cada um desses grupos.

Os grupos de consultas com a mesma resposta (ou "regiões" do espaço de consulta onde a resposta é invariante) formam **classes de equivalência**. As respostas às consultas para as diversas classes de equivalência são pré-computadas e esta informação é organizada em alguma estrutura que permita acesso eficiente, tipicamente as chamadas **árvores de localização**, baseadas em localizações sucessivas da consulta em uma hierarquia de decomposições cada vez mais finas do conjunto de consultas possíveis. Exemplos deste tipo de estrutura podem ser encontrados em algumas soluções para busca em subespaços circulares (capítulo 4) e em soluções para busca em semi-espaços utilizando dualidade (seção 2.7).

A viabilidade de aplicação da abordagem do lugar geométrico depende do número de classes de equivalência e da quantidade de informações a serem processadas e mantidas para estas classes.

Exemplo: O problema de dominância

Um exemplo do emprego do paradigma do lugar geométrico é a solução descrita em [PS85] para busca em subespaços ortogonais no plano, em modo de contagem. Esta solução utiliza o conceito de **dominância** de um ponto em relação a um conjunto de pontos.

Dominância

Diz-se que um ponto q **domina** um ponto p se e somente se toda coordenada de q têm valor não inferior ao da coordenada correspondente em p:

$$q \text{ domina } p \iff q.x_i \geq p.x_i \ \forall \ 1 \leq i \leq d \tag{2.1}$$

A **dominância de q em relação a um conjunto de pontos** P, denotada por $\text{Dom}(q, P)$, é o subconjunto dos pontos de P dominados por q:

$$\text{Dom}(q, P) = \{p \in P \mid q \text{ domina } p\} \tag{2.2}$$

Considere o problema de dominância no plano. Os pontos de P dominados por um ponto q são aqueles contidos no quadrante sudoeste de q, como ilustra a figura 2.1.

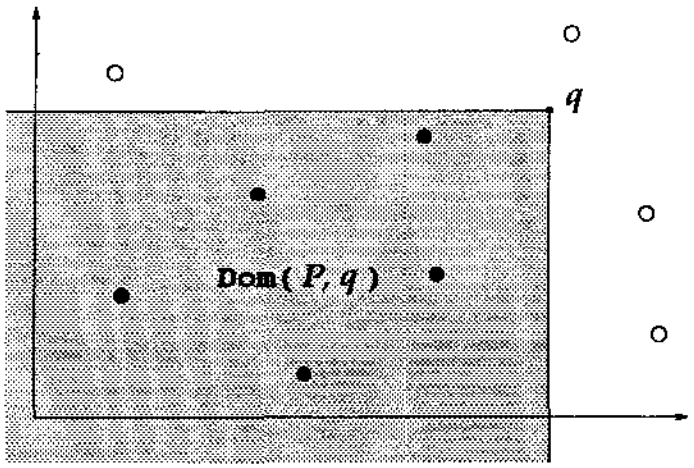


Figura 2.1: O problema de dominância no plano.

O número de pontos da base de dados P contidos em um retângulo de busca σ com vértices v_1, v_2, v_3 e v_4 pode ser obtido a partir da dominância desses vértices (por simplicidade de descrição, considere os valores de cada coordenada dos pontos de P todos distintos). A figura 2.2 ilustra a relação denotada pela equação 2.3:

$$|P \cap \sigma| = |\text{Dom}(v_1, P)| - |\text{Dom}(v_2, P)| - |\text{Dom}(v_3, P)| + |\text{Dom}(v_4, P)| \tag{2.3}$$

Desta forma, uma busca em subespaço ortogonal é reduzida a 4 consultas de dominância.

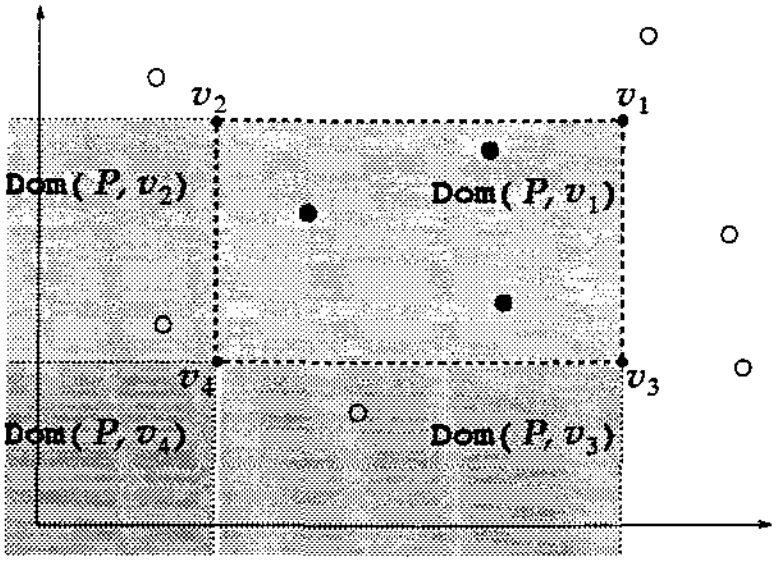


Figura 2.2: Mapeamento de busca em subespaço ortogonal no problema de dominância.

A vantagem desta redução é que o problema de dominância pode ser resolvido eficientemente utilizando o paradigma do lugar geométrico.

Seja o reticulado obtido traçando-se linhas verticais e horizontais passando por cada ponto de P como ilustra a figura 2.3. Quaisquer pontos no interior de um mesmo retângulo deste reticulado dominam os mesmos pontos de P. Estes retângulos constituem portanto as *classes de equivalência* para o problema de dominância. Tem-se $O(N^2)$ desses retângulos, sendo então necessário espaço de memória da ordem de $O(N^2)$ para armazenar todas as respostas pré-computadas. Pode-se organizar esta informação de modo que dada uma busca em subespaço retangular, a dominância de cada vértice seja determinada por meio de duas buscas binárias, uma para cada coordenada, resultando em tempo total $O(\log N)$ para responder uma consulta.

Aplicações e extensões

A abordagem do lugar geométrico na forma descrita acima é mais custosa para problemas de enumeração. Para tratar problemas de enumeração, ao invés de armazenar a *quantidade* de pontos de P relativos a cada classe de equivalência, é necessário armazenar os *próprios subconjuntos* de P relativos a cada uma destas classes, o que potencialmente pode acarretar um aumento de um fator $O(N)$ na demanda de memória.

Este inconveniente pode ser contornado afrouxando-se um pouco o critério de formação dos agrupamentos de consultas (já não formando classes de equivalência). Pode-se considerar grupos de consultas com respostas semelhantes mas não necessariamente iguais (ou "regiões" próximas no espaço de consultas) e concatenar as respostas relativas às consultas do grupo, ou propriedades que permitam calculá-las eficientemente, em um mesmo repositório de informações, de modo que a resposta para uma consulta em particular possa ser obtida eficientemente a partir

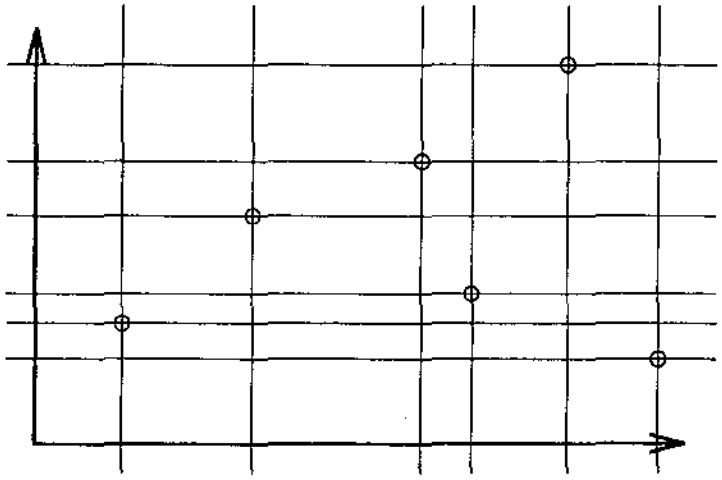


Figura 2.3: O reticulado correspondente à abordagem do lugar geométrico para o problema de dominância no plano.

das informações armazenadas para o grupo correspondente. Desta forma, a informação compartilhada é armazenada somente uma vez para cada grupo de consultas, resultando em economia de espaço. Para obter a resposta de uma consulta, a partir das informações compartilhadas do grupo, pode-se utilizar algum tipo de filtragem (veja paradigma de filtragens sucessivas na seção 2.2).

Exemplos do emprego desta abordagem para compactação de estruturas de dados (paradigma do lugar geométrico em conjunção com filtragens sucessivas) são freqüentemente encontrados na literatura. As soluções para busca em sub-espaços circulares utilizando diagramas de Voronoi para determinar as classes de equivalência (seção 4.1) constituem um exemplo. Outro exemplo típico desta abordagem é a técnica proposta por Cole [Col86] para efetuar buscas em diversas listas contendo elementos em comum.

2.2 Filtragens sucessivas

O paradigma de filtragens sucessivas (*filtering search*) [Cha86] é adequado para problemas de busca no modo de enumeração. O princípio básico deste paradigma é manter a complexidade do processamento das consultas inferior a um valor pré-estabelecido ou, no máximo, proporcional ao tempo necessário para enumerar a saída.

Descrição

Como foi discutido na seção 1.2.1, a complexidade de um algoritmo de busca no modo de enumeração possui duas componentes: $f(N)$, o tempo de busca em função do tamanho da entrada e k, o tempo de enumeração, que é proporcional ao tamanho da saída. O paradigma de

filtragens sucessivas tem por fundamento procurar relacionar $f(N)$ e k de modo que se minimize o tempo de busca quando se tem uma saída pequena. A idéia é a seguinte: para *enumerar* k pontos da resposta é necessário tempo $\Omega(k)$ de qualquer forma, então pode-se despender outro termo $O(k)$ para *calcular* a resposta, sem afetar a eficiência assintótica.

Primeiramente, obtém-se um conjunto de objetos candidatos, utilizando algum algoritmo eficiente. Este algoritmo testa uma condição *mais fraca*, no sentido que aqueles objetos satisfazendo à consulta satisfazem esta condição, mas não necessariamente a recíproca. Assim, este conjunto contém todos os elementos que satisfazem à consulta e, possivelmente, mais alguns que satisfaçam apenas à condição mais fraca utilizada, de modo que precisa ainda ser filtrado. O importante é que a cardinalidade deste conjunto intermediário seja uma função $g(k)$ do tamanho da saída, de preferência uma função de crescimento assintótico pequeno (o quão pequeno depende das operações a serem aplicadas a este conjunto de objetos candidatos).

Posteriormente, um novo passo é realizado sobre o conjunto de objetos candidatos, a fim de eliminar (*filter out*) os objetos estranhos. Pode-se ter inclusive filtragens com múltiplos estágios, aplicando repetidos refinamentos à coleção de objetos de dados candidatos. Imagine passar os objetos de dados candidatos por filtros cada vez mais finos (no sentido da propriedade a ser satisfeita).

Exemplos de aplicação do paradigma

O uso mais primitivo do paradigma de filtragens sucessivas consiste em desconsiderar no tempo de busca $f(N)$ quaisquer custos que sejam proporcionais ao tamanho da saída, na análise de complexidade dos algoritmos. É comum, em se tratando com estruturas de árvores, alcançar um nó v sabendo que todos os seus descendentes satisfazem à consulta e ainda assim, ser necessário descer até as folhas descendentes de v, onde são identificados os pontos (veja árvores de partição na seção 2.3). Este processo de propagação descendente para identificação obedece à equação de recorrência:

$$T(\overline{k}) = 2 \cdot T\left(\left\lceil \overline{k}/2 \right\rceil\right) + c \tag{2.4}$$

onde $\overline{k}$ é o número de folhas descendentes de v e c é o tempo necessário para processar qualquer nó na estrutura.

A cada um dos $\overline{k}$ pontos enumerados corresponde um número constante de folhas descendentes de v (um número menor ou igual a 1 quando não há replicação de pontos), donde se deduz que a solução da equação 2.4 é $T(N) = O(\overline{k})$. Portanto, o processo de propagação descendente a partir do nó v leva tempo linear no tamanho da saída relativa a v.

Um outro tipo de emprego, mais elaborado, do paradigma de filtragens sucessivas é também freqüente na literatura. Considere um problema de busca em modo de enumeração, com uma base de dados P, tal que se possa determinar uma coleção de subconjuntos aninhados de P, $C = \{C_1, C_2, C_4, \ldots, C_N\}$, onde C_i $(i = 1, 2, 4 \ldots, N)$ contém os pontos constituintes das respostas de todas as consultas com saída de tamanho menor ou igual a i. Suponha que dada uma

consulta qualquer (cujo tamanho da resposta não precisa ser necessariamente potência de 2) se possa determinar o menor C_i que contenha os pontos constituintes da resposta desta consulta, em tempo logarítmico. Então, para se obter os elementos de dados satisfazendo à consulta, basta filtrar os elementos de C_i, o que, no pior caso, leva tempo igual ao dobro do tamanho da resposta. Assim, o tempo total necessário para responder uma consulta é $O(\log N + k)$. Exemplos concretos desse tipo de aplicação do paradigma de filtragens sucessivas, em conjunção com o paradigma do lugar geométrico (seção 2.1) podem ser encontrados nas soluções para busca em subespaços circulares com partições determinadas por diagramas de Voronoi (seção 4.1).

Os ganhos obtidos nas soluções de problemas de busca em subespaços com o emprego do paradigma de filtragens sucessivas algumas vezes são surpreendentes. No que toca particularmente a busca em subespaço ortogonal, conseguiu-se um ganho de um fator multiplicativo $1/\log\log N$ na utilização de memória da estrutura *range tree* (veja seções 3.1.3 e 3.3).

2.3 Árvores de partição

As árvores de partição (*partition trees*) constituem um dos tipos de estruturas de dados mais empregados para solucionar problemas de busca em subespaços. O conceito de árvore de partição deriva do trabalho de Willard [Wil82b] sobre busca em sub-espaço poligonal no plano (capítulo 5), onde uma estrutura de dados deste tipo aparece pela primeira vez. Naquele trabalho não se menciona que a concepção básica da solução apresentada constitui uma abordagem genérica. Matoušek [Mat91c] formaliza o conceito de árvores de partição, aplicando-o na solução de busca em semi-espaços e busca em simplexos (capítulo 6). Formalizações adicionais e extensões do conceito podem ser encontradas em [AM92a, Mat93a].

Descrição

Uma árvore de partição é uma estrutura de dados cujo funcionamento baseia-se em informações topológicas: decomposições do espaço multidimensional induzindo partições do conjunto de pontos P (que aplicadas recursivamente sugerem uma estrutura de árvore) e interseção do subespaço de busca σ com as sub-regiões constituintes destas decomposições.

O esquema de partição (*partition scheme*) utilizado determina a conformação da árvore de partição. Um esquema de partição se aplica a todo o espaço de busca Γ ou a uma região $\mathcal{R}$ deste e permite obter uma coleção de sub-regiões (não necessariamente disjuntas ou cobrindo o espaço em questão), determinadas pelos pontos armazenados na base de dados P e induzindo uma partição de P (ou de $P \cap \mathcal{R}$) em subconjuntos associados a essas sub-regiões.

Cada sub-região proveniente da aplicação de um esquema de partição deve ser associada a uma determinada fração dos pontos de P, correspondente a um subconjunto dos pontos de P nela contidos e a fronteira $\partial\sigma$ de qualquer subespaço de busca $\sigma \in \Sigma$ (do tipo sendo considerado) *não* deve cortar pelo menos uma quantidade não constante dessas sub-regiões. Assim, dado um subespaço de busca σ, pode-se tratar diretamente os pontos contidos nas sub-regiões *não*

interceptadas por $\partial\sigma$: eles estão todos dentro ou todos fora do subespaço de busca. Para manipular os pontos contidos nas sub-regiões interceptadas por $\partial\sigma$, o esquema de partição pode ser aplicado recursivamente em cada sub-região da decomposição. Este processo de decomposição recursiva se repete até atingir subconjuntos induzidos de P suficientemente pequenos, para que possam ser tratados de maneira trivial por meio de busca seqüencial, sem afetar a complexidade assintótica da solução.

As **árvores de partição** são estruturas de dados baseadas na idéia de aplicação recursiva de esquemas de partição para solucionar problemas de busca em subespaços. A cada nó não folha v de uma árvore de partição é associada uma região $\mathcal{R}(v)$ do espaço, um conjunto de pontos $\mathcal{P}(v)$ ($P \cap \mathcal{R}(v)$ ou um subconjunto deste) e uma decomposição $\Phi(v)$ de $\mathcal{R}(v)$, determinada por $\mathcal{P}(v)$ (às vezes por todo o conjunto P) e induzindo uma partição deste. Para cada sub-região $\varphi \in \Phi(v)$ é atribuído um filho a v. Ao nó raiz é associado todo o espaço Γ e o conjunto de pontos P. Aos nós folha são associadas regiões com uma quantidade de pontos de P inferior a um valor constante pré-estabelecido.

Cada conjunto de pontos $\mathcal{P}(v)$, relativo à sub-região $\mathcal{R}(v)$ associada ao nó v de uma árvore de partição é chamado de **conjunto canônico**. Considerando-se consultas relativas a um semi-grupo aditivo, pode-se armazenar em cada nó v o peso acumulado dos pontos do conjunto canônico $\mathcal{P}(v)$, já pré-calculado. Uma árvore de partição permite calcular a resposta de uma busca em subespaço, somando-se os pesos acumulados dos conjuntos canônicos relativos a alguns dos seus nós, escolhidos de acordo com o subespaço de busca dado.

O processo de busca pelos pontos contidos em um subespaço σ começa pela raiz da árvore de partição. Em cada nó v visitado pelo processo de busca, são contabilizados os pesos acumulados dos pontos de P associados às sub-regiões da decomposição $\Phi(v)$ internas a σ e procede-se recursivamente na árvore para as sub-regiões interceptadas por $\partial\sigma$.

A fim de dar uma noção mais concreta do funcionamento de uma árvore de partição, é descrita a seguir a estrutura proposta por Willard.

Exemplo: a árvore de Willard

A árvore de Willard [Wil82b] foi originalmente proposta para possibilitar a solução de buscas em regiões poligonais no plano, não necessariamente conexas ou limitadas. Entretanto, ela pode ser mais facilmente compreendida se analisada como um método de busca em *semi-espaços* no plano, conforme descrição apresentada por Matoušek em [Mat93a].

Seja P um conjunto de N pontos no plano. Por simplicidade, assuma os pontos de P em posição geral, isto é, que nenhum subconjunto de P com três pontos é colinear. Seja r_1 uma reta de direção arbitrária dividindo o conjunto de pontos P ao meio. Pelo teorema do *ham-sandwich cut* [Ede87] existe uma reta r_2, tal que r_1 e r_2 dividem o plano em 4 regiões, $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$ e $\mathcal{R}_4$, cada qual contendo 1/4 dos pontos de P (por simplicidade de descrição, considere $N = 4^c$, onde c é um inteiro maior que 1), como ilustra a figura 2.4-a.

Seja $\sigma(h)$ um semi-plano limitado por uma reta h. É fácil ver que a reta h intercepta o

interior de no máximo 3 regiões $\mathcal{R}_i$ $(1 \leq i \leq 4)$, ou seja uma das regiões fica totalmente dentro ou totalmente fora do semi-plano $\sigma(h)$, como ilustra a figura 2.4-b. No processo de busca pelos pontos contidos em $\sigma(h)$, esta região pode ser tratada diretamente. Para as demais regiões é necessário processamento adicional para determinar quais pontos contidos no interior das mesmas pertencem ao semi-plano de busca $\sigma(h)$.

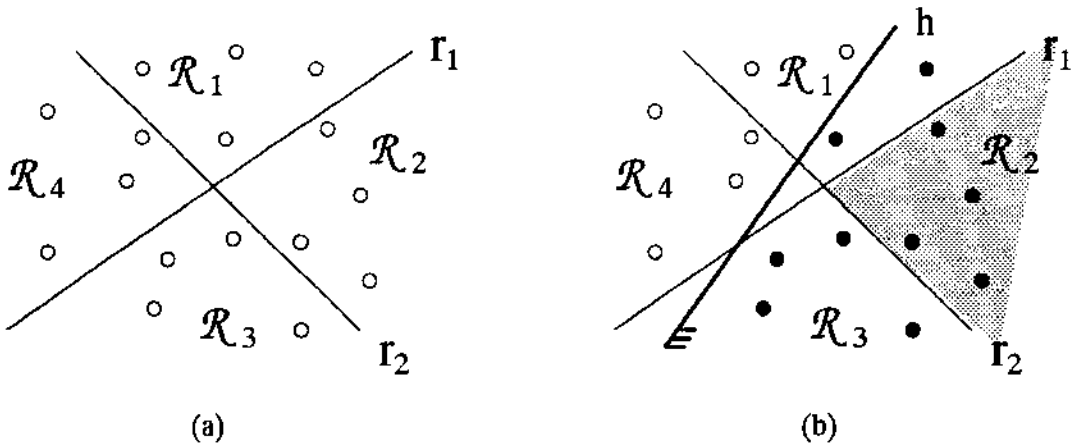


Figura 2.4: A partição de um conjunto de pontos no plano em 4 partes iguais (a) e uma região totalmente contida em um semi-espaço (b).

Esta técnica de decomposição do problema permite uma economia de 25% do processamento de uma consulta e aplicando-a recursivamente nas regiões decompostas, tem-se a seguinte equação de recorrência para o tempo de consulta de pior caso:

$$T(N) = 3T(N/4) + \overline{k} \in O(N^{\log_4 3} + k) \subset O(N^{0,7925} + k) \tag{2.5}$$

onde $\overline{k}$ é o número máximo de pontos contidos em uma região de uma decomposição e k é o tempo de enumeração da resposta.

Este método sugere uma estrutura de dados pré-processada T em forma de árvore, armazenando a informação das partições aplicadas recursivamente, de modo a dar suporte ao processamento das consultas.

O processo de busca começa pela raiz de T e descende recursivamente nos nós relativos às regiões interceptadas por h. Para problemas de contagem, pode-se armazenar em cada nó v o número de pontos contidos em $\mathcal{R}(v)$. No caso de problemas de enumeração os pontos de P ficam armazenados nas folhas de T associadas com as regiões que os contêm. Para enumerar os pontos contidos em uma região $\mathcal{R}(v)$ é necessário descer até as folhas descendentes de v, mas o tempo despendido neste processo é proporcional ao tamanho da saída, de modo que este processamento adicional não aumenta o tempo de busca assintótico[1].

Willard originalmente propôs uma partição determinada por duas retas mais um número configurável de semi-retas. Uma descrição da estrutura proposta por Willard, no contexto de

[1] Observa-se aqui uma ocorrência sutil do paradigma de filtragens sucessivas (seção 2.2).

busca em regiões poligonais e segundo as peculiaridades do esquema de partição empregado pode ser encontrada na seção 5.1.1.

Aplicações e extensões

O teorema do *ham-sandwich cut* [Ede87] e teoremas similares podem ser usados para provar a existência de esquemas de partição. Yao e Yao [YY85] provam a existência de esquemas de partição para toda dimensão fixa, tais que qualquer hiperplano cruza uma quantidade sub-linear de sub-regiões, o que tem motivado a pesquisa de soluções baseadas em árvores de partição para várias dimensões. Atualmente, há uma ampla bibliografia com soluções para busca em subespaços baseadas no paradigma de árvores de partição, propondo diversos esquemas de partição distintos.

A eficiência de uma solução utilizando um determinado esquema de partição é determinada pelo número máximo de sub-regiões que podem ser cortadas pela fronteira de um subespaço de busca, assim como pela distribuição dos pontos da base de dados entre essas sub-regiões.

Observa-se que a abordagem de árvores de partição é dominante na quantidade de trabalhos publicados e nos desempenhos obtidos para busca em semi-espaços e simplexos (capítulo 6). As soluções baseadas neste paradigma estão entre as mais eficientes para o caso geral de busca em semi-espaços e buscas simpliciais com uso de memória quase linear[2]. Entretanto, elas são superadas por outras soluções de concepções distintas, em casos específicos como enumeração dos pontos contidos em semi-espaços e simplexos, para dimensões pequenas.

Cabe observar ainda que o conceito de árvore de partição é bastante genérico, podendo ser aplicado a uma ampla classe de problemas de busca em subespaços. Recentemente, árvores de partição têm sido empregadas também para busca em subespaços semi-algébricos em geral, mas os resultados ainda são escassos [AM92a, Mat93a]. Dentre as questões abertas está a determinação de esquemas de partição que possibilitem soluções eficientes para esses tipos de subespaços.

2.4 Divisão e conquista multidimensional e estruturas de dados multinível

O paradigma de divisão e conquista multidimensional foi originalmente formalizado por Bentley [Ben80] e sua generalização resulta no conceito de estruturas de dados multinível, um tipo de solução muito empregado para problemas de busca em subespaços dados pela interseção de diversas regiões mais simples (ou, de maneira equivalente, problemas de busca dados por conjunções de restrições). Formalizações do conceito de estruturas de dados multinível, no contexto de árvores de partição, são realizadas por Matoušek em [Mat92, Mat93a]. A seguir, são sintetizados alguns conceitos e comentários encontrados nestes trabalhos de Bentley e Matoušek.

[2] Mais formalmente, "quase linear" significa de ordem $O(N^{1+\epsilon})$, onde o valor de ϵ depende da relação de compromisso estabelecida entre o uso de memória e o tempo de busca.

A formulação de Bentley

Em [Ben80], são considerados problemas de busca envolvendo algum tipo de ortogonalidade, que permita reduzi-los a sub-problemas em um espaço de dimensão menor. A idéia formulada por Bentley é a seguinte:

Divisão e conquista multidimensional

Para resolver um problema de N pontos no espaço d-dimensional primeiro se resolve, recursivamente, dois problemas de aproximadamente $N/2$ pontos no espaço d-dimensional (as duas metades do conjunto de pontos, particionado por um hiperplano perpendicular a uma das dimensões) e então se resolve, também recursivamente, um problema de N pontos no espaço $(d-1)$-dimensional (considera-se os N pontos projetados perpendicularmente no hiperplano que divide o conjunto de pontos). Somando-se[3] as respostas das chamadas recursivas obtém-se a solução desejada.

A descrição acima pode ser vista como uma especificação funcional de uma estrutura de dados (seção 2.6). A partir dela, pode-se construir estruturas de dados para auxiliar na solução de problemas de busca. Um exemplo de tais estruturas é a *Range-Tree*, descrita a seguir.

Exemplo: a *Range-Tree*

O método de solução de Bentley sugere uma estrutura de dados para suportar consultas em modo repetitivo, composta de árvores cujos nós apontam para outras árvores (não sub-árvores descendentes), definidas recursivamente para solucionar sub-problemas com uma dimensão a menos. A *Range-Tree* é a estrutura de dados para suportar buscas em subespaços ortogonais, baseada na idéia de Bentley.

Na *Range-Tree*, uma árvore primária induz uma decomposição do espaço em faixas perpendiculares à primeira dimensão, possibilitando a decomposição do problema em sub-problemas relativos a intervalos da primeira coordenada. Os subconjuntos de P contidos nas faixas determinadas por esta decomposição constituem os chamados conjuntos canônicos e esta estrutura primária constitui um tipo de árvore de partição (seção 2.3).

Cada nó v da árvore primária é associado a um conjunto canônico e aponta para uma estrutura secundária, de forma análoga à da estrutura primária, para tratar o sub-problema com uma dimensão a menos relativo ao conjunto canônico associado a v. Este processo de tratar os sub-problemas por meio de estruturas definidas recursivamente se aplica por um número de níveis igual ao número de dimensões[4]. A figura 2.5 ilustra uma *Range-Tree* em dois níveis, onde

[3] Entenda-se aqui a operação genérica de soma em semi-grupo. Esta operação pode ser uma simples soma, no caso de um problema de busca no modo de contagem ou uma união de conjuntos, no caso de enumeração.

[4] Observe que o termo *nível* aqui não diz respeito aos *níveis* de descendência em uma árvore (raiz, filhos da

os nós circulares e arestas com linhas de contorno mais grossas denotam a estrutura de árvore primária e os triângulos conectados aos nós da estrutura primária, com linhas de contorno mais finas, denotam as estruturas do segundo nível.

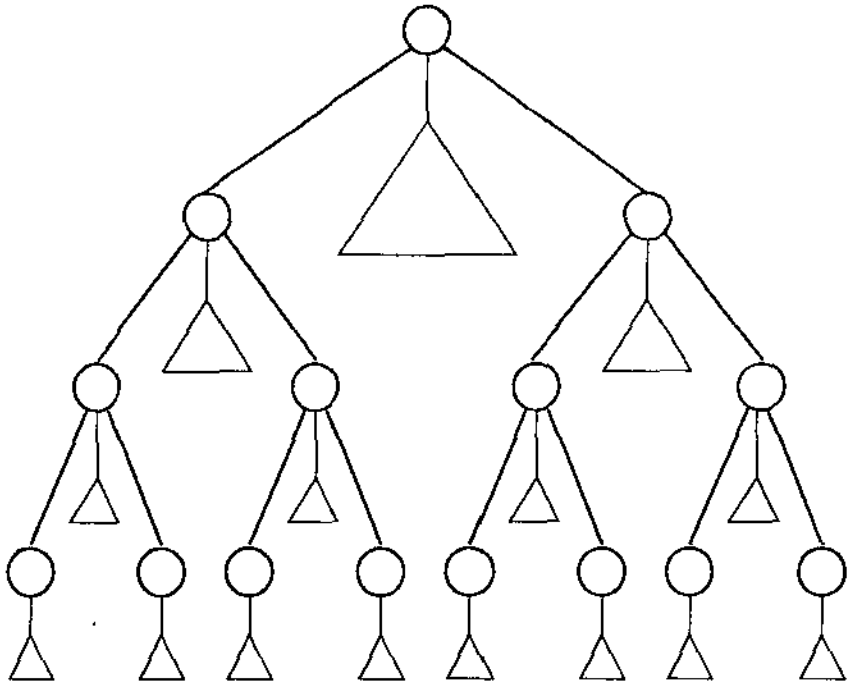


Figura 2.5: Uma *Range-Tree* em dois níveis.

Uma busca em subespaço ortogonal é decomposta pelo primeiro nível de uma *Range-Tree* em $O(\log N)$ sub-consultas em conjuntos canônicos, os quais incluem todos os pontos contidos em uma faixa larga perpendicular a primeira dimensão e contendo o hiper-retângulo de busca. Cada conjunto canônico é processado utilizando a estrutura do próximo nível, a fim de determinar os pontos contidos nos intervalos relativos às demais dimensões (veja descrição detalhada da *Range-Tree* na seção 3.1.3).

Aplicações e extensões

A idéia de Bentley pode ser estendida, isolando o seu princípio básico: decompor o problema em sub-problemas mais simples, resolvê-los *recursivamente* e então calcular a resposta do primeiro a partir das respostas dos sub-problemas. Este princípio é bastante universal e especialmente aplicável a problemas de busca em que as consultas são denotadas por uma conjunção de condições, tais que para cada condição já exista uma estrutura de dados eficiente para tratar o problema considerando somente ela.

Considere problemas de busca em subespaços expressos por conjunções de condições ou,

raiz, etc.), mas aos níveis de árvores secundárias (não sub-árvores) conectadas a partir da estrutura primária.

geometricamente, pela interseção de diversas regiões. Por exemplo, uma busca em subespaço ortogonal pode ser vista como a interseção de d intervalos $[a_i, b_i] \subset \gamma_i$ (faixas no espaço multidimensional) e uma busca em um simplexo (ou politopo) como a conjunção de buscas em *semi-espaços*. Existem soluções para os problemas envolvendo os termos dessas conjunções (faixas e semi-espaços, respectivamente), tipicamente baseadas em alguma forma de árvore de partição (seção 2.3).

Os problemas envolvendo conjunções de condições podem ser tratados por meio de estruturas de dados multinível, compostas de estruturas para solucionar os problemas relativos a cada um dos termos das conjunções, dispostas em níveis.

Considere árvores de partição em uma estrutura multinível. Seja T a árvore de partição para tratar o primeiro termo da conjunção de condições. Para cada nó v de T e para cada sub-região φ da decomposição $\Phi(v)$, ao invés de se armazenar simplesmente o peso acumulado dos pontos de P associados a φ, é pendurada em v uma estrutura secundária análoga à estrutura primária, mas com um nível (de estruturas encadeadas) a menos, para tratar as demais condições. Este processo de definição recursiva da estrutura se repete por um número de níveis de estruturas igual ao número de condições a serem tratadas.

Dado um subespaço de busca σ, denotado por uma conjunção de condições, utiliza-se a estrutura primária T para determinar os maiores conjuntos canônicos de P totalmente contidos em um subespaço determinado pela primeira condição da conjunção. A união desses conjuntos canônicos inclui todos os pontos de $P \cap \sigma$. Para cada um desses conjuntos canônicos satisfazendo à primeira condição, recorre-se à estrutura secundária correspondente, para determinar os pontos satisfazendo também às demais condições relativas a σ.

Em árvores de partição, os conjuntos canônicos são os pontos associados às regiões resultantes da aplicação do esquema de partição. Entretanto, o conceito de divisão e conquista multidimensional se aplica em muitos outros contextos. Para se construir uma estrutura de dados multinível para solucionar um problema de busca, é necessária uma regra que permita obter uma partição do conjunto de objetos de dados em conjuntos canônicos e expressar uma consulta como uma coleção de consultas (mais simples) nestes conjuntos canônicos. Não importa como são obtidos esses conjuntos canônicos, mas é importante que eles possam ser calculados eficientemente.

O uso de memória de uma estrutura multinível é determinado pelo tamanho total dos conjuntos canônicos a serem armazenados (os quais podem não ser disjuntos) e o tempo de consulta é determinado pelo número máximo de conjuntos canônicos resultantes da decomposição de uma consulta.

À primeira vista, pode parecer que estruturas de dados multinível sejam ineficientes assintoticamente, no uso de memória e no tempo de busca. Entretanto, esse não é geralmente o caso, porque não é comum muitos subconjuntos canônicos grandes (para uma árvore binária, por exemplo, há apenas um conjunto canônico de tamanho N, dois de tamanho $N/2$, quatro de tamanho $N/4$ e assim por diante). Em algumas estruturas multinível o fator de sobrecarga é de $O(N^\varepsilon)$ (onde $\varepsilon < 1$ é uma constante positiva) por nível e em outras é de apenas $O(\log N)$ por nível, tanto no uso de memória quanto no tempo de consulta. Matoušek [Mat92] descreve

esquemas de partição possibilitando estruturas de dados multinível com uma sobrecarga poli-logarítmica por nível.

Por outro lado, parece que a eficiência prática de uma estrutura multinível, a despeito das medidas assintóticas, deve se deteriorar rapidamente com o aumento do número de níveis (não há ainda experiência prática para substanciar esta afirmação). Uma estrutura tratando a conjunção de condições diretamente (como é possível encontrar para simplexos) pode ser mais apropriada em muitos casos.

2.5 Buscas por propagação

A técnica de buscas por propagação (*fractional cascading*) [CG86a, CG86b, MN90] é destinada a realizar **busca iterativa**, isto é, buscas pelo mesmo valor de chave em uma seqüência de m listas ordenadas separadas, com valores de chave em um mesmo domínio γ, de forma mais eficiente do que uma cadeia de m buscas binárias independentes, que requer tempo $O(m \log N)$, onde N é o tamanho máximo de qualquer uma das listas. Diversas soluções de problemas geométricos recaem em busca iterativa, tais como interseção de um caminho poligonal com uma reta e alguns tipos de buscas em subespaços como busca em subespaço ortogonal, busca em subespaço em forma de trapézio e busca em semi-espaço.

Descrição

As listas ordenadas nas quais se deseja realizar as buscas são colocadas em correspondência um a um com os vértices de um grafo $G(V, E)$, denominado **grafo de adjacências**, de modo que o processo de busca iterativa sempre segue pelas arestas de $G(V, E)$, que é uma abstração para indicar os possíveis roteiros da seqüência de buscas nas listas, isto é, em qual lista se pode efetuar a primeira busca e dela para qual lista passar e assim por diante, até ter efetuado as buscas em todas as listas desejadas.

O conhecimento da posição de um valor de chave em uma lista relativa a um vértice $v \in V$ pode ser utilizado para agilizar a busca pelo mesmo valor de chave em uma lista adjacente a v no grafo $G(V, E)$. Utilizando ponteiros conectando valores subseqüentes armazenados em listas adjacentes a intervalos regulares, de modo a formar uma malha de ponteiros entre as listas, a posição de um valor de chave de busca pode ser propagada de uma lista para outra lista adjacente em $G(V, E)$ em tempo limitado por uma constante (dependente da densidade da malha). Uma malha de ponteiros entre listas de valores armazenados é ilustrada na figura 2.6-a. Cada coleção de ponteiros conectando listas correspondentes a vértices adjacentes em $G(V, E)$ corresponde a uma aresta $e \in E$. A figura 2.6-b mostra o grafo de adjacências correspondente à malha de ponteiros entre listas da figura 2.6-a.

A técnica de buscas por propagação permite realizar m buscas pela mesma chave, em uma seqüência de listas relativas aos nós de um caminho no grafo $G(V, E)$ utilizado, com um custo extra $\log g$ por busca (isto é, por lista considerada), onde g é um parâmetro denominado **grau**

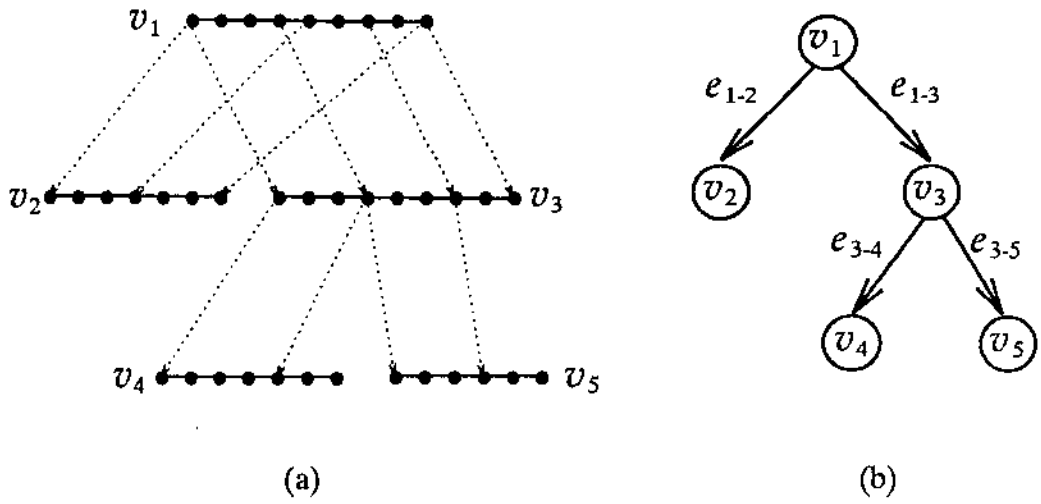


Figura 2.6: A técnica de buscas por propagação: uma malha de ponteiros entre listas (a) e o grafo de adjacências correspondente (b).

localmente limitado dos nós de $G(V, E)$, determinando a densidade da malha de ponteiros.

A primeira busca é realizada como uma busca binária comum, a um custo $O(\log n_0) \subset O(\log s)$, onde n_0 é o tamanho da primeira lista e s é a somatória dos tamanhos de todas as listas. A partir daí, utiliza-se a malha de ponteiros para alcançar os elementos com o mesmo valor de chave ou valores vizinhos em listas adjacentes.

Fazendo um balanceamento adequado desses ponteiros e mantendo o grau de cada vértice de $G(V, E)$ limitado por g, consegue-se garantir um custo extra para cada busca adicional (cada lista considerada, além da primeira onde é realmente efetuada a busca) não superior a $O(\log g)$. Assim, uma seqüência de m buscas pode ser realizada em tempo $O(\log s + m \cdot \log g)$. É demonstrado, por meio de uma análise de complexidade amortizada [Tar85] que, mantendo-se o valor do parâmetro g inferior a $(6d - 1)$, onde d é o número de dimensões, o tempo de pré-processamento e o uso de memória mantêm-se proporcionais a $O(d \cdot s)$ e $O(s)$, respectivamente [CG86a].

Exemplo: a *Range-Tree* com buscas por propagação

Como foi visto na seção 2.4, a árvore primária de uma *Range-Tree* permite decompor uma busca em subespaço ortogonal em $O(\log N)$ buscas em conjuntos canônicos, que são subconjuntos de P contidos em faixas perpendiculares à primeira dimensão, as quais interceptam o hiper-retângulo de busca. Para determinar os pontos destes conjuntos contidos no intervalo de busca relativo à próxima dimensão é necessário efetuar, em cada um deles, uma busca pelo valor da chave correspondente ao início deste outro intervalo. Tem-se então um problema de busca iterativa, onde os conjuntos canônicos com chaves relativas à segunda dimensão correspondem às listas ordenadas.

Considere uma *Range-Tree* para pontos no plano (2 níveis). Os conjuntos canônicos resul-

tantes da partição pelo primeiro nível de uma *Range-Tree* são dispostos em forma de árvore binária, onde os elementos de cada nó são separados em duas metades, cada qual replicada em um dos filhos. Considere cada conjunto canônico ordenado em um vetor pelas chaves da segunda dimensão. Para agilizar as buscas nestes conjuntos canônicos, pode-se conectar ponteiros partindo de determinadas posições do conjunto canônico relativo a cada nó pai para as posições correspondentes nos subconjuntos relativos aos filhos, a intervalos regulares de valores de chave. Estes ponteiros agem como pontes, permitindo que a posição de uma chave pela qual é efetuada uma busca no conjunto canônico relativo ao nó raiz seja propagada para os subconjuntos relativos aos nós inferiores da estrutura.

Esta técnica possibilita uma agilização de um fator $O(\log N)$ do tempo de busca em subespaço ortogonal utilizando *Range-Tree* (veja detalhes em [LW82, Wil85, PS85]). O uso de memória assintótico não é afetado, pois os ponteiros adicionais representam um custo extra de um fator constante apenas.

Aplicações e extensões

A técnica de buscas por propagação pode ser estendida para possibilitar a propagação de posições em listas encadeadas, onde não são explicitadas as chaves de busca. Chazelle, Guibas e Lee [CGL85], utilizando buscas por propagação, conseguiram uma solução ótima para o problema de *enumeração* dos pontos contidos em um *semi-espaço* no plano: tempo de consulta $O(\log N + k)$, onde k é o tamanho da saída e uso de memória $O(N)$. A solução deles, descrita na seção 6.1.2, baseia-se nas camadas convexas dos pontos de P. Dado o semi-espaço de busca, calcula-se a interseção de seu hiperplano limitante com um casco das camadas convexas, de modo a determinar um ponto de P contido no semi-espaço de busca. Então, segue-se os ponteiros conectando os pontos do casco convexo em questão e deste para os outros cascos, enumerando os pontos contidos no semi-espaço de busca. Uma solução muito semelhante foi proposta em [CG86b] para enumeração dos pontos contidos em um trapézio no plano.

Há também estudos sobre dinamização de buscas por propagação, contando com alguns resultados aplicáveis na prática. Porém, alguns dos esquemas propostos têm apenas interesse teórico devido às enormes constantes multiplicativas envolvidas nas medidas de desempenho e/ou por serem complexos demais para se implementar. Mehlhorn e Näher [MN90] propuseram um esquema que permite efetuar inserções e remoções a um custo proporcional a $O(\log\log s)$, aumentando o tempo de m buscas para $O(\log s + m \cdot \log g + m \cdot \log\log s)$. Se apenas inserções ou apenas remoções tiverem de ser suportadas o fator $O(\log\log s)$ dos tempos de busca e de atualização (inserção ou remoção) reduz-se para $O(1)$.

2.6 Abordagem funcional para estruturas de dados

Uma abordagem funcional para estruturas de dados foi proposta por Chazelle [Cha88] e consiste em estudar o funcionamento das estruturas do ponto de vista do cliente, procurando determinar a funcionalidade mínima necessária para atender às suas necessidades. A partir deste estudo,

muitas vezes se consegue aperfeiçoar as estruturas de dados, obtendo estruturas mais arrojadas, compactas e voltadas para tarefas específicas.

Descrição

Uma das formas de se encarar uma estrutura de dados é vê-la como um grafo, com os dados armazenados nos nós. Pode-se adicionar regras semânticas para especificar o funcionamento da estrutura e como ela pode ser modificada (por exemplo, em uma árvore binária de busca os filhos à esquerda de um nó v têm valor de chave menor que o de v) e pode-se adicionar também restrições para garantir certos critérios de forma (como balanceamento e restrições de grau).

A abordagem funcional estende esta noção de estrutura de dados, associando com cada nó v do grafo correspondente à estrutura não apenas uma peça de dados como é usual, mas uma função f_v (ou diversas se necessário) assim como uma coleção de dados $\mathcal{P}(v)$, de maneira que avaliar f_v seja suficiente, mas não necessário, para ter disponível $\mathcal{P}(v)$.

Cada função f_v determina uma estrutura de dados chamada **estrutura de dados funcional** (*functional data structure*), denominação abreviada por *EDF*. A própria estrutura de grafo primária também pode ser vista como uma *EDF* determinada por alguma função. O processo de expandir as funções associadas aos nós de uma *EDF* em outras *EDFs*, produzindo estruturas físicas correspondendo à descrição funcional abstrata é chamado **refinamento**. Uma característica essencial de uma *EDF* é permitir refinamento passo a passo, a medida que surge a necessidade de acessar os dados.

O benefício desta abordagem é aliviar a demanda de memória. A associação do conjunto de dados $\mathcal{P}(v)$ ao nó v é virtual e assim $\mathcal{P}(v)$ não precisa estar armazenado fisicamente para cada nó. O único requisito é que os valores de dados estejam disponíveis de alguma forma (mediante algum processamento) quando forem necessários. Desta forma, tem-se no uso de memória um efeito análogo ao que avaliação por demanda (*call by need, lasy evaluation*) tem no tempo de execução: apenas a quantidade mínima de informação necessária para avaliar a função em determinada quantidade de tempo fica armazenada e só é expandida quando a informação detalhada for requisitada. Isso possibilita também estabelecer relações de compromisso entre o uso de memória e o tempo de processamento.

Para dar uma noção mais concreta dos conceitos discutidos acima, é apresentada a seguir a aplicação da abordagem funcional à estrutura *Range-Tree* (seção 3.1.3), utilizada para solucionar busca em subespaços ortogonais.

Exemplo: A *Range-Tree* como estrutura funcional

Vejamos uma definição funcional para a *Range-Tree*. Por simplicidade de descrição, considere os valores de cada coordenada dos pontos de P todos distintos. Seja f a função de busca mapeando a sub-região ortogonal de busca $\sigma = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ e o conjunto de pontos P em um subconjunto $\mathcal{P} = f(P, \sigma) \subseteq P$, relativo aos pontos de P contidos em σ. Se o

conjunto de pontos P for vazio o problema é trivial: nenhum ponto de P está em σ. Senão, se P estiver totalmente contido no intervalo de busca relativo à primeira dimensão, use a função g para resolver o problema na próxima dimensão com o conjunto de dados $P' = \{(x_2, \ldots, x_d) \mid (x_1, x_2, \ldots, x_d) \in P\}$ e o subespaço de busca $\sigma' = [a_2, b_2] \times \ldots \times [a_d, b_d]$, isto é, P e σ descartando a primeira dimensão; senão divida P em duas metades met_esq(P) e $P-$met_esq(P), através da mediana das coordenadas x_1 dos pontos de P e recorra para cada um dos lados da faixa no espaço multidimensional, dividida na mediana de x_1, que tenha interseção não vazia com σ. A equação abaixo sintetiza esta descrição funcional.

$f(P, \sigma) \equiv$ **if** $P \neq \emptyset$ **then**
if $\forall\, p \in P,\ a_1 \leq p.x_1 \leq b_1$ **then** $g(P', \sigma')$
else
Seja med(P) a mediana das coordenadas x_1 de P;
Seja met_esq(P) = $\{p \in P \mid p.x_1 \leq \text{med}(P)\}$;
(limitado a $|\text{met\_esq}(P)| \leq N/2$)
if med(P) $\geq a_1$ **then** $f(\text{met\_esq}(P), \sigma)$;
if med(P) $\leq b_1$ **then** $f(P-\text{met\_esq}(P), \sigma)$;

A equação funcional $f(P, \sigma)$ sugere uma estrutura de árvore $F(P)$ (*EDF*) para solucionar buscas em subespaços ortogonais dado o conjunto de pontos P. A cada nó v de $F(P)$ é associado (implicitamente) um subconjunto canônico $\mathcal{P}(v) \in P$, obtido da aplicação a P de composições das funções met_esq(P) e $P-$met_esq(P). Ao nó raiz está associado o conjunto P. Ao filho à esquerda de um nó v é associado o conjunto met_esq($\mathcal{P}(v)$) e ao filho à direita $\mathcal{P}(v)-$met_esq($\mathcal{P}(v)$).

A cada nó v de $F(P)$ é associada uma função $g(\mathcal{P}(v)', \sigma')$, para solucionar o problema na próxima dimensão com os pontos de $\mathcal{P}(v)$. A função $g(\mathcal{P}', \sigma')$ pode ter uma definição similar à de $f(P, \sigma)$:

$g(\mathcal{P}', \sigma') \equiv$ **if** $\mathcal{P}' \neq \emptyset$ **then**
if $\forall\, p \in \mathcal{P}',\ a_2 \leq p.x_2 \leq b_2$ **then** $h(P'', \sigma'')$
else
Seja med($\mathcal{P}'$) a mediana das coordenadas x_2 de $\mathcal{P}'$;
Seja met_esq($\mathcal{P}'$) = $\{p \in \mathcal{P}' \mid p.x_2 \leq \text{med}(\mathcal{P}')\}$;
(limitado a $|\text{met\_esq}(P')| \leq N/2$)
if med(P') $\geq a_2$ **then** $f(\text{met\_esq}(\mathcal{P}'), \sigma')$;
if med(P') $\leq b_2$ **then** $f(\mathcal{P}'-\text{met\_esq}(\mathcal{P}'), \sigma')$;

Esta equação tem suas operações, inclusive as funções med e met_esq, diferindo das operações de f apenas pela coordenada manipulada e determina *EDF's* análogas a $F(P)$, denotadas por

$G(\mathcal{P}')$. Associa-se uma estrutura $G(\mathcal{P}(v)')$ a cada nó v de $F(P)$, como é determinado pela própria equação $f(P,\sigma)$. O processo de avaliar as $G(\mathcal{P}(v)')$, expandindo a estrutura de dados física, constitui o refinamento.

A figura 2.7-a ilustra a *EDF* determinada pelas equações acima. Note a semelhança de forma da estrutura de cada árvore $G(\mathcal{P}(v)')$ associada a um nó v com a sub-árvore de $F(P)$ com raiz em v. Cada um desses pares é constituído de estruturas idênticas em sua topologia (abstraindo as informações associadas aos nós), de modo que se pode efetuar na estrutura a transformação denominada **sobreposição** (*folding*), pela qual cada estrutura $G(\mathcal{P}(v)')$ é dobrada sobre a sub-árvore de $F(P)$ com raiz em v. Obtém-se assim uma única árvore "sobrecarregada", isto é, com diversos níveis de estruturas de dados funcionais sobrepostos nela, conforme ilustra a figura 2.7-b. Codificando toda a informação necessária para as buscas dos diversos níveis nesta estrutura sobrecarregada, tem-se uma economia de memória, tipicamente, de um fator multiplicativo $O(\log N)$ por nível sobreposto.

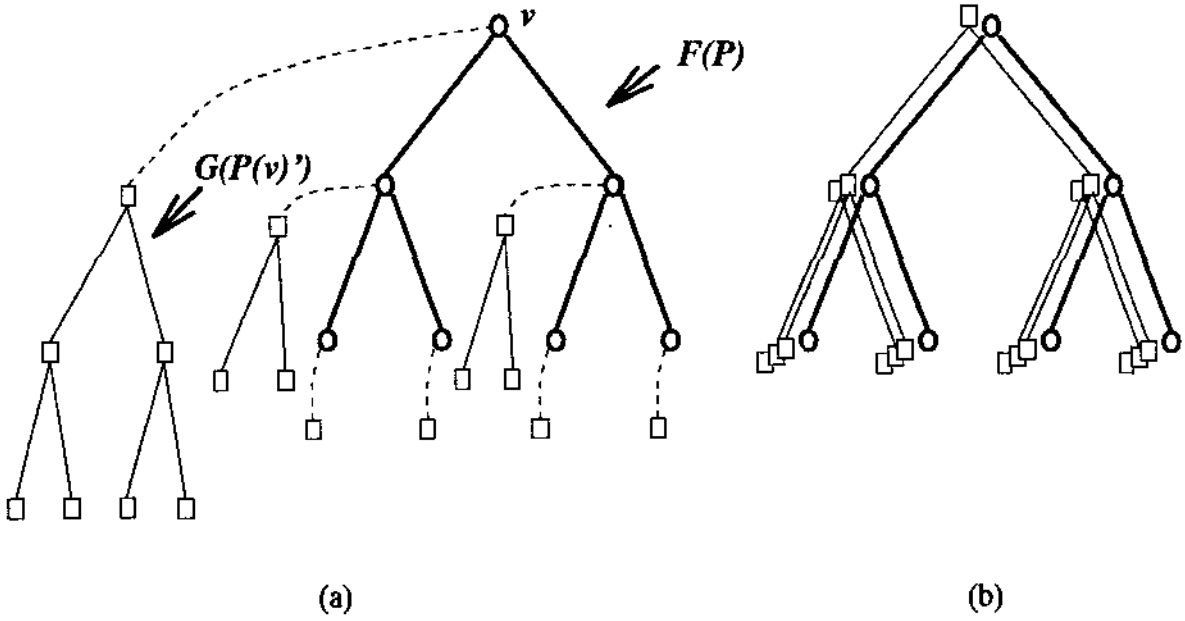


Figura 2.7: A estrutura de dados correspondente à especificação funcional da *Range-Tree* (a) e a transformação de sobreposição (b).

A transformação de sobreposição requer estruturas topologicamente iguais, como ilustra a figura 2.8-a. A atribuição dos dados aos nós das estruturas não é o aspecto importante, mas sim a disposição dos nós e arestas. Entretanto, quando as estruturas tiverem formas diferentes, como ilustra a figura 2.8-b, ainda assim pode ser possível trabalhar suas definições funcionais de modo a torná-las compatíveis. Esta adequação pode acarretar um aumento da complexidade e do tempo de execução dos algoritmos e é importante manter esta complexidade sob controle a fim de viabilizar o método.

O benefício da abordagem funcional para estruturas de dados existentes é permitir visualizar

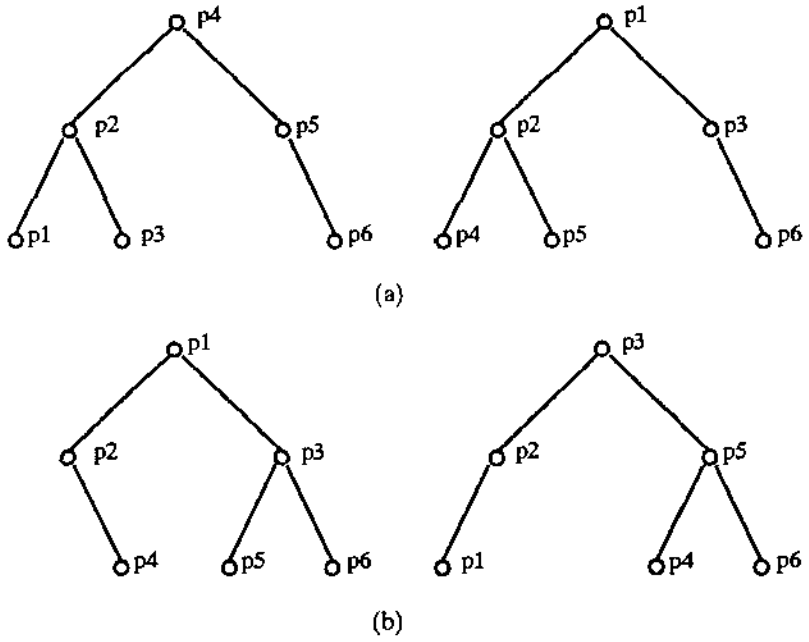


Figura 2.8: Semelhança de estruturas de dados funcionais: (a) estruturas passíveis de sobreposição; (b) estruturas incompatíveis.

mais facilmente possíveis aperfeiçoamentos das mesmas. As semelhanças entre as porções das estruturas funcionais podem ser utilizadas para compactá-las, através de sobreposição e, se as componentes forem diferentes impossibilitando a sobreposição, pode-se tentar modificar sua definição funcional de modo a torná-las compatíveis.

Na próxima seção, são esboçados alguns detalhes da implementação de sobreposição em uma *Range-Tree* para busca em subespaços ortogonais no plano, a fim de oferecer ao leitor interessado uma visão mais concreta da transformação efetuada na estrutura. O leitor interessado apenas nos conceitos gerais pode desconsiderá-la e passar para a seção seguinte.

Um exemplo numérico de *Range-Tree* com sobreposição

Seja o conjunto de pontos no plano:

$$P = \{(34,3),(12,1),(28,23),(64,15),(2,35),(6,17),(52,43),(22,13)\} \tag{2.6}$$

A estrutura da *Range-Tree* para os pontos de P é ilustrada na figura 2.9. Veja que as folhas da árvore principal $F(P)$, na ordem da esquerda para a direita, correspondem a uma ordenação ascendente dos pontos de P segundo a coordenada x, enquanto as folhas das árvores $G(\mathcal{P}(v)')$, penduradas em cada nó v de $F(P)$ para suportar a solução na próxima dimensão, correspondem aos pontos de $\mathcal{P}(v)$, ordenados segundo a coordenada y. Note que cada sub-árvore de $F(P)$ com

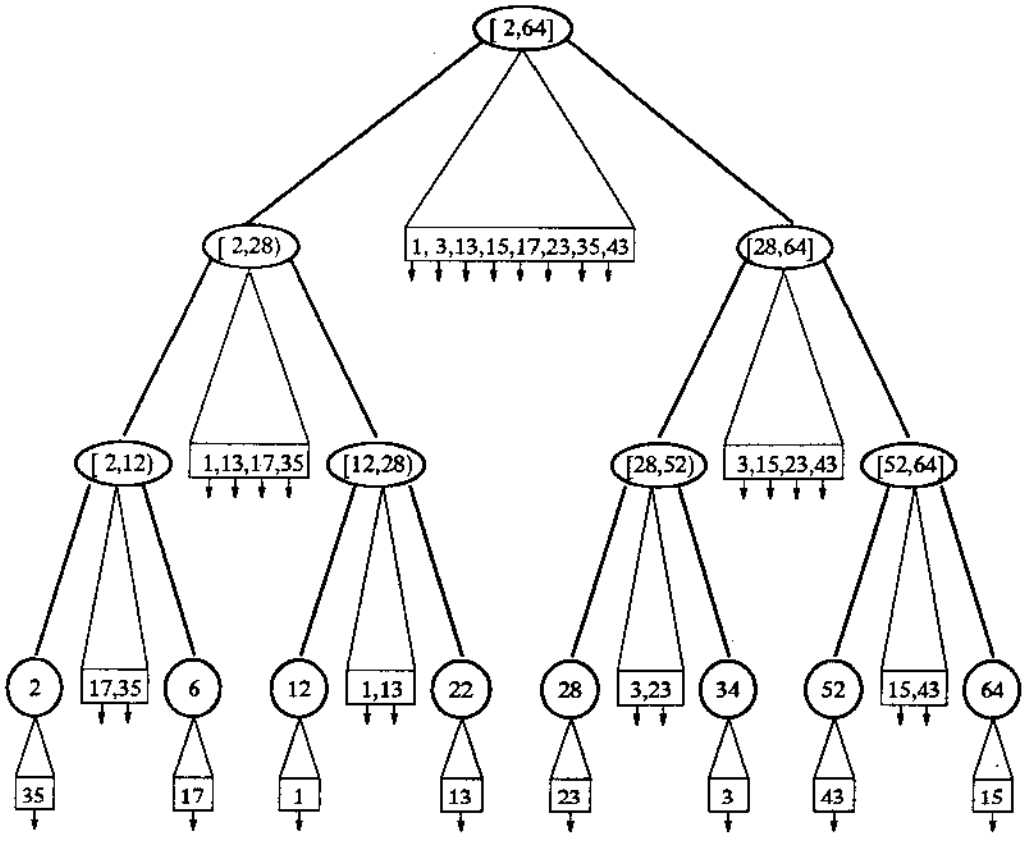


Figura 2.9: *Range-Tree* correspondente ao conjunto de pontos P.

raiz em um nó v pode ter a mesma forma da estrutura $G(\mathcal{P}(v)')$ pendurada a v, apesar das duas estruturas dizerem respeito a coordenadas distintas.

A transformação de sobreposição é efetuada codificando a informação necessária a cada $G(\mathcal{P}(v)')$ na própria sub-árvore de $F(P)$ com raiz em v. Utiliza-se um campo de bits $B(v) = [b_0, b_1, \ldots, b_{|\mathcal{P}(v)|-1}]$ em cada nó v, onde $b_i = 0$ ($b_i = 1$) se o elemento na posição i da lista ordenada dos pontos de $\mathcal{P}(v)$ provém do filho à esquerda (direita) de v. Os $B(v)'s$ representam movimentos de ponteiros e têm semelhanças com os ponteiros da técnica de busca por propagação da seção 2.5, no sentido que permitem propagar a posição de uma chave de busca de um nó para outro.

Com esses campos de bits é possível propagar uma posição i correspondente a uma chave $y_i(v)$ do nó v para a posição j correspondente à mesma chave no filho apropriado (filho à esquerda se $b_i(v) = 0$ e à direita se $b_i(v) = 1$). O índice j é igual ao número de $0's$ ($1's$) à esquerda de $b_i(v)$ se $b_i(v) = 0$ ($b_i(v) = 1$). Desta maneira, pode-se propagar uma busca pela coordenada y efetuada no nó raiz para nós inferiores da árvore.

A figura 2.10 ilustra estes campos de bits na sua disposição em árvore para a *Range-Tree* relativa ao conjunto de pontos P da equação 2.6, com uma estrutura contendo os valores das

coordenadas y dos pontos, em ordem, conectada ao nó raiz da estrutura.

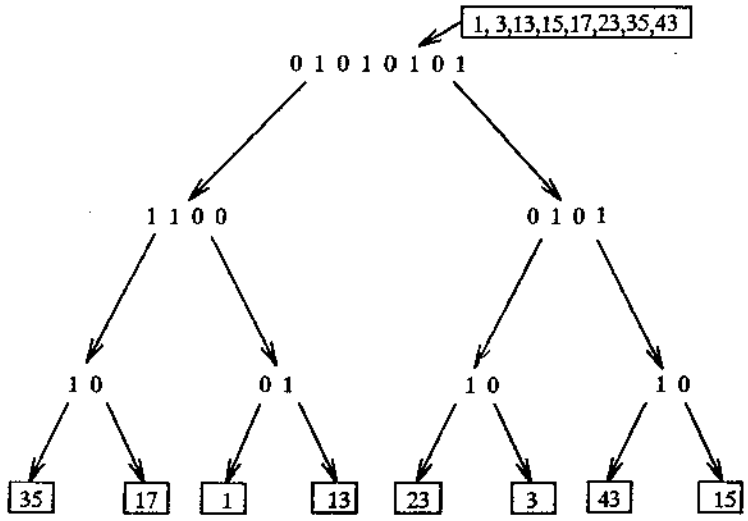


Figura 2.10: Campos de bits para as coordenadas y dos pontos de P.

O tempo de cada transição de posição de chave de um nó v para um de seus filhos é o tempo necessário para contar o número de $0's$ e $1's$ à esquerda de $b_i(v)$ em $B(v)$. Seja $n = |B(v)|$. Para realizar o cálculo da quantidade de $0's$ e $1's$ eficientemente, divide-se o vetor de bits $B(v)$ em uma seqüência de palavras $\beta_0, \beta_1, \ldots, \beta_{m-1}$, cada qual com $\lfloor \log n \rfloor$ bits. No modelo de máquinas de ponteiros (seção 1.5.3) tais palavras são organizadas em árvores que permitem obter a resposta em tempo $O(\log n)$, sem alterar o uso de memória por um fator superior a uma constante. No modelo de máquina de acesso aleatório o cálculo pode ser feito em tempo constante, utilizando uma estrutura linear em n. Detalhes podem ser encontrados em [Cha88].

Resultados obtidos

A estrutura *Range-Tree* com sobreposição, esboçada acima, permite solucionar o problema de busca em subespaço ortogonal no plano, modo de contagem, em tempo e espaço ótimos: $O(\log N)$ e $O(N)$, respectivamente. O tempo de busca permanece o mesmo da *Range-Tree* original para pontos no plano. O argumento pelo qual a estrutura é linear é fácil de ser explicado. Sabe-se que $F(P)$ ocupa espaço linear no tamanho de P. Então, basta provar que o espaço ocupado pelos campos de bits é linear no tamanho da entrada. Tem-se que cada nível da árvore requer $O(N)$ bits para armazenar os $B(v)'s$ correspondentes aos seus nós. Como a árvore tem $O(\log N)$ níveis, são necessários $O(N \log N)$ bits, que é justamente o mínimo necessário para armazenar a entrada, uma vez que cada coordenada requer uma palavra de $\Omega(\log N)$ bits (podem haver N coordenadas distintas). Assim, conclui-se que o espaço total ocupado pela *Range-Tree* para pontos no plano, com os níveis sobrepostos é linear no tamanho da entrada.

Chazelle [Cha88] discute também a adaptação desta solução a diversos tipos de máquinas de ponteiros e fornece variações da solução permitindo obter diferentes relações entre as medidas de

desempenho (tempo de busca e uso de memória), além de oferecer algumas soluções dinâmicas em tipos específicos de máquinas de ponteiros.

2.7 Dualidade

O princípio da dualidade [CGL85, PY86, Ede87, Sto91] consiste em estabelecer uma correspondência dos objetos do espaço primal (*primal space*) com outros objetos de um espaço dual, de tal forma que um problema no espaço primal corresponde a um outro problema no espaço dual e que as soluções para os dois problemas sejam intercambiáveis, através da função de dualidade. Utilizando dualidade consegue-se o dobro dos resultados com praticamente o mesmo esforço, além disso, muitos problemas são mais fáceis de serem tratados em sua formulação dual.

Definição

Para se determinar uma dualidade estipula-se alguma função de mapeamento, $\mathcal{D}$, pela qual entidades do espaço primal são transformadas em outras entidades de um espaço dual. Por um mapeamento geométrico clássico, um ponto p no plano corresponde a uma reta $\mathcal{D}(p)$ no plano dual e uma reta ℓ a um ponto $\mathcal{D}(\ell)$ (usamos a mesma designação "$\mathcal{D}$" tanto para o mapeamento quanto para o seu inverso).

Geralmente, utilizam-se mapeamentos preservando **relações de incidência**, para se poder efetuar a correspondência entre os problemas de forma fiel. Por exemplo, um mapeamento de pontos em retas e vice-versa, preservando incidência, deve ser tal que se um ponto p está *sobre* uma reta ℓ no espaço primal, então a reta $\mathcal{D}(p)$ no espaço dual passa *sobre* o ponto $\mathcal{D}(\ell)$. Desta forma, determinar a reta passando por determinados pontos equivale, no espaço dual, a determinar o ponto onde as retas duais aos pontos do espaço primal se interceptam.

Outro tipo de relação desejável de se preservar em uma dualização são as **relações de orientação**. Por exemplo, um ponto p localizado *acima* de uma reta ℓ (considere uma reta não vertical) deve corresponder a uma reta $\mathcal{D}(p)$ *acima* do ponto $\mathcal{D}(\ell)$ ou, pelo menos, o mapeamento das relações *acima* e *abaixo* deve ser consistente (ou sempre inverte ou sempre mantém a relação de orientação).

Muitos enunciados de problemas têm seus duais e muitas demonstrações e soluções de problemas estabelecem automaticamente dois resultados distintos. Por uma dualidade do tipo ponto em reta e vice-versa, cada teorema tem seu dual, obtido trocando-se a palavra **ponto por reta** e vice-versa. Por exemplo, a expressão "um ponto caminhando sobre uma reta" dualiza para "uma reta girando em torno de um ponto", como ilustra a figura 2.11.

A dualidade constitui também uma ferramenta poderosa em projeto, análise e descrição de algoritmos. Particularmente para busca em subespaços, pode-se explorar o fato da dualidade permitir reduzir o problema a localização de pontos, intersecção de segmentos ou algum outro problema, para o qual já existam soluções eficientes. Dualidade também costuma ser empregada em determinadas soluções sofisticadas, para reduzir algum sub-problema componente de um

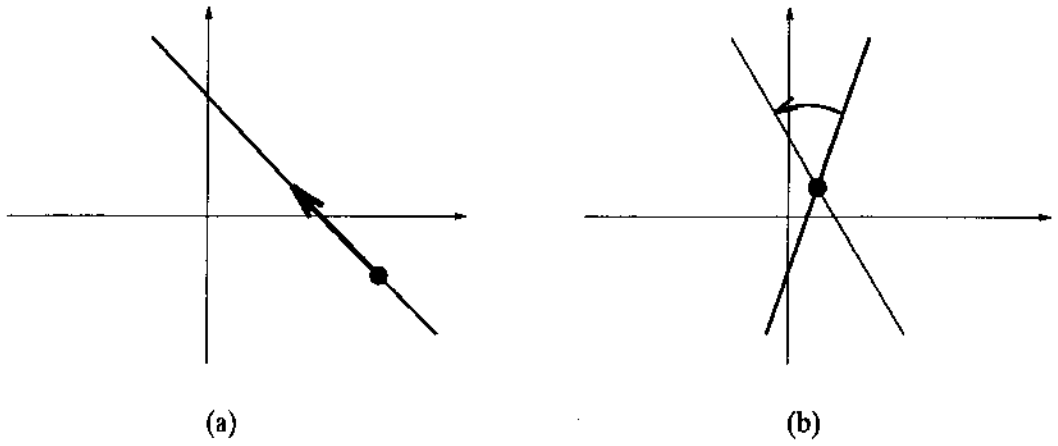


Figura 2.11: Um ponto caminhando sobre uma reta (a) corresponde a uma reta girando em torno de um ponto no espaço dual (b).

problema maior e solucioná-lo eficientemente por meio de alguma sub-rotina.

O princípio da dualidade é portanto uma ferramenta extremamente útil na teoria e na prática. A figura 2.12 ilustra algumas configurações de entidades geométricas no plano primal e as configurações correspondentes no plano dual, para uma função de mapeamento típica.

Exemplo: o dual de busca em semi-espaço

Seja o problema de determinar os pontos de P contidos em um *semi-espaço* $\sigma(h)$, limitado por um hiperplano h. Considere a transformação de dualidade $\mathcal{D}$, pela qual cada ponto $p = (a_1, a_2, \ldots, a_d) \in \mathbf{R}^d$ corresponde a um hiperplano $\mathcal{D}(p)$, dado pela equação $a_1x_1 + a_2x_2 + \ldots + a_dx_d + 1 = 0$ no espaço dual e analogamente, cada hiperplano $h \in \mathbf{R}^d$ corresponde a um ponto $\mathcal{D}(h)$.

Pode-se demonstrar que os pontos de P contidos no semi-espaço de busca $\sigma(h)$ (considere h não vertical) são exatamente aqueles cujos hiperplanos duais são interceptados por uma semi reta vertical partindo do ponto $\mathcal{D}(h)$, com orientação dependendo da orientação de $\sigma(h)$ em relação a h.

Um arranjo de hiperplanos divide o espaço $\mathbf{R}^d$ em um conjunto de regiões, tais que os hiperplanos interceptados pela semi-reta partindo de $\mathcal{D}(h)$, localizado em qualquer posição de uma mesma região são sempre os mesmos (veja paradigma do lugar geométrico na seção 2.1). Então, basta localizar o ponto $\mathcal{D}(h)$ neste arranjo para determinar os hiperplanos interceptados pela semi-reta e conseqüentemente, os pontos de P contidos em $\sigma(h)$. Esta correspondência é ilustrada na figura 2.13.

Esta correspondência de busca em *semi-espaço* com localização de ponto num arranjo de hiperplanos no espaço dual constitui o princípio básico de muitos algoritmos com tempo de busca poli-logarítmico propostos na literatura e é analisada em mais detalhes na seção 6.1.1.

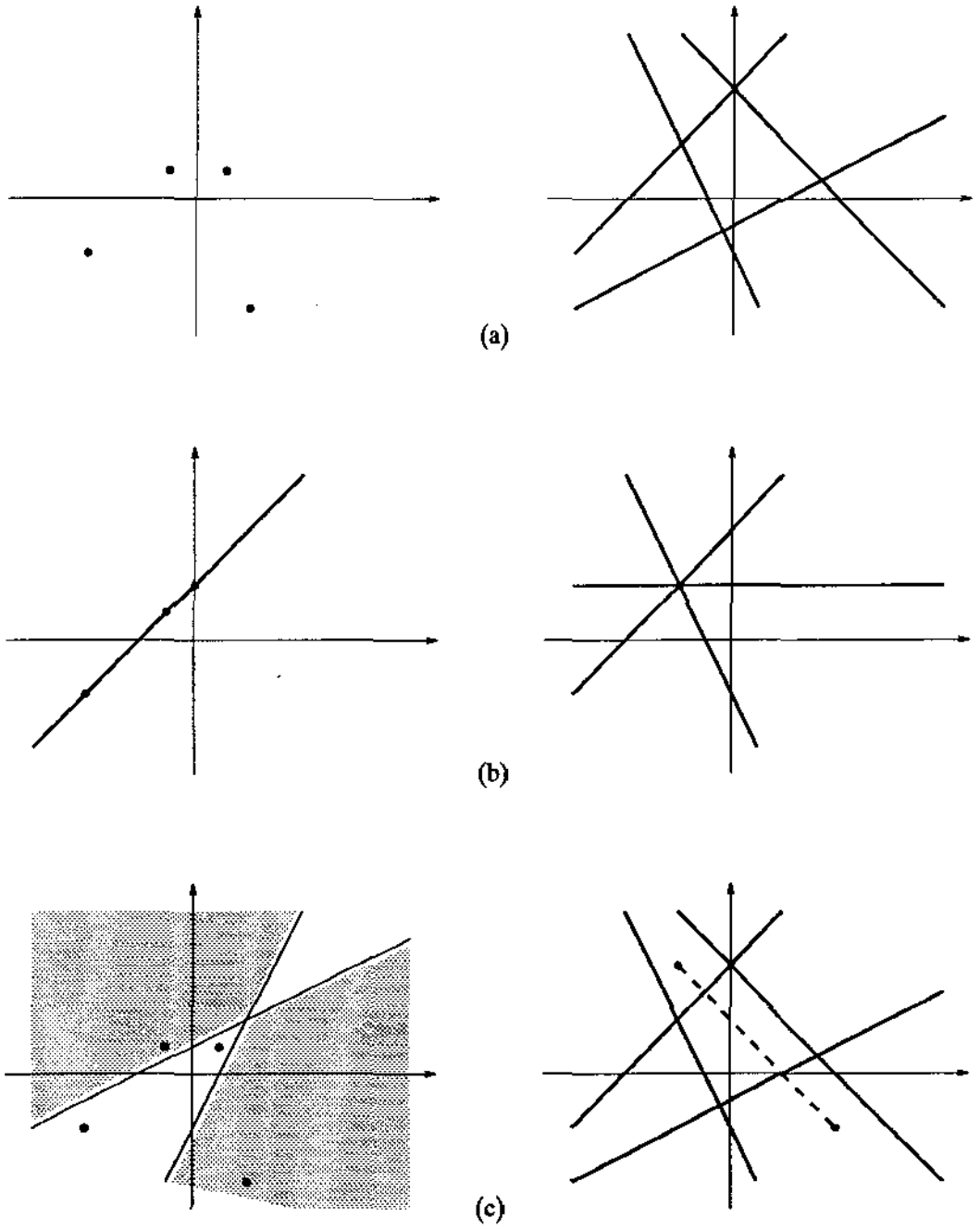
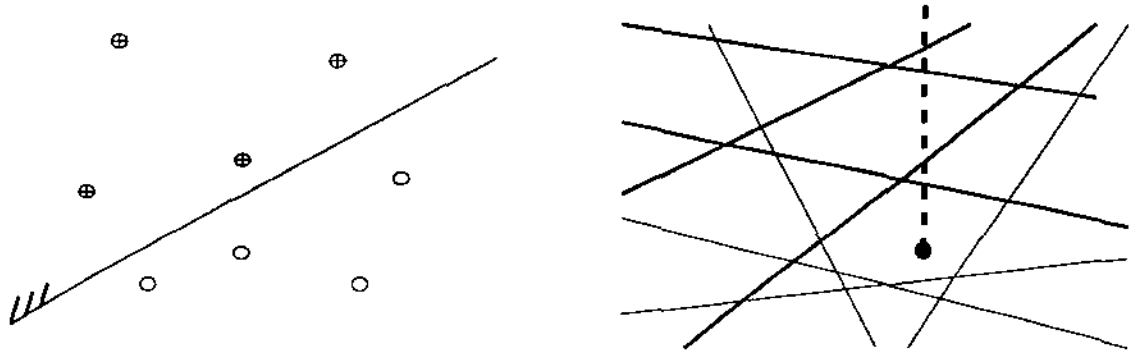


Figura 2.12: O princípio da dualidade: (a) a cada ponto corresponde uma reta; (b) pontos sobre uma reta correspondem a retas passando pelo ponto correspondente àquela reta; (c) pontos em uma cunha dupla correspondem a retas interceptadas pelo segmento de reta correspondente à cunha.



(a) (b)

Figura 2.13: Busca em semi-espaço (a) e o problema correspondente no espaço dual (b).

Partindo desta correspondência, pode-se construir uma estrutura de árvore de localização (seção 2.1), na qual cada nó v é associado a uma região $\mathcal{R}(v)$ do espaço dual (e portanto, a um subconjunto das buscas possíveis). Seja $\mathcal{H}(v)$ o subconjunto dos hiperplanos de $\mathcal{D}(P)$ interceptando $\mathcal{R}(v)$. Em cada nó v considera-se uma partição por alguns dos hiperplanos de $\mathcal{H}(v)$ e atribui-se um filho a v para cada sub-região proveniente desta partição. Ao nó raiz da árvore de localização é associado todo o espaço Γ. Com esta estrutura, pode-se efetuar uma localização gradativa da classe de equivalência, partindo da raiz e localizando $\mathcal{D}(h)$ na partição relativa a cada nó visitado para determinar em que filho descer. Uma tal árvore de localização e o caminho nela percorrido para localizar a região contendo um determinado ponto são ilustrados na figura 2.14.

O processo de decomposição hierárquica do espaço de consulta através dos níveis da árvore cessa quando se atingem as classes de equivalência ou conjuntos de hiperplanos suficientemente pequenos para serem testados um a um, a fim de determinar quais são interceptados por uma semi-reta vertical partindo do ponto $\mathcal{D}(h)$. No segundo caso, não há respostas pré-computadas para classes de equivalência nas folhas, sendo necessário armazenar em cada nó v o conjunto de hiperplanos fora de $\mathcal{R}(v)$ e interceptando qualquer semi-reta vertical partindo de um ponto de $\mathcal{R}(v)$ no sentido sendo considerado (para baixo ou para cima, dependendo da orientação de $\sigma(h)$), de modo que a resposta possa ser acumulada a medida que se desce na estrutura.

Extensões: o espaço projetivo orientado

No plano euclideano só se pode construir dualidades imperfeitas, as quais não se aplicam em certos caso particulares (retas passando pela origem de dualidade, por exemplo). Um meio de contornar tal inconveniente é utilizar geometria projetiva[5] [CGL85, PY86, Sto91], expressando

[5]Pode-se representar o plano projetivo como o plano euclideano acrescido de pontos no infinito (duais a retas passando pela origem) e de uma reta no infinito (dual ao ponto origem).

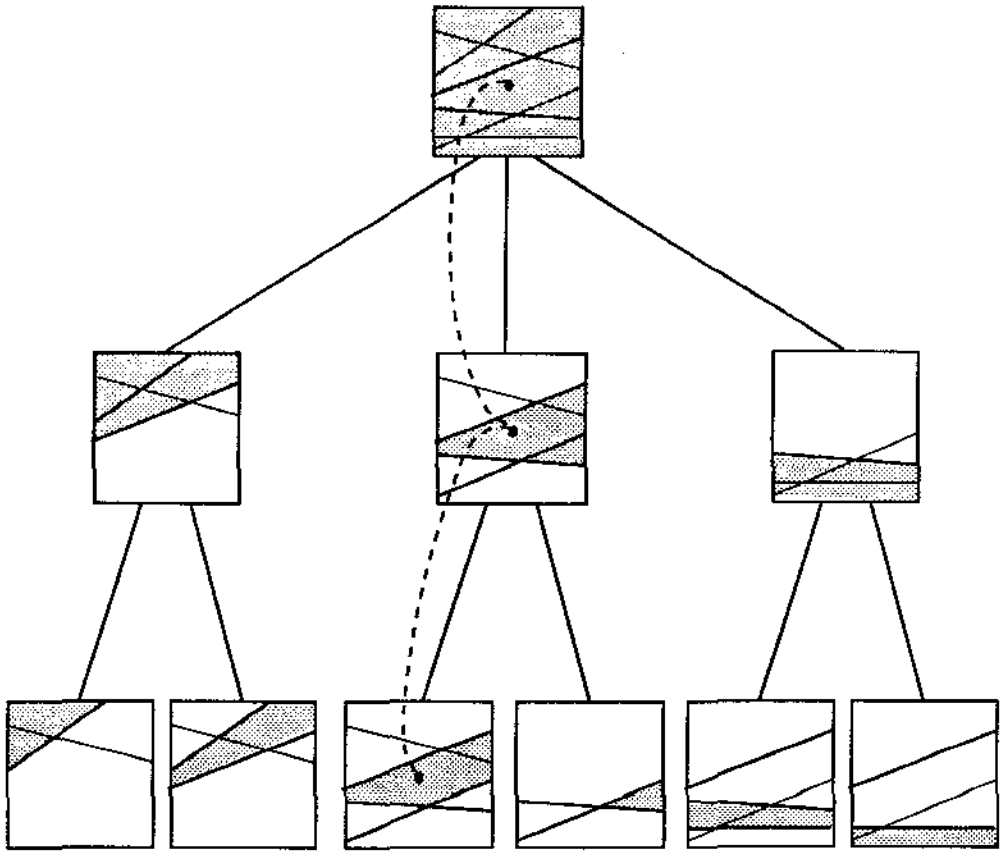


Figura 2.14: Uma árvore de localização para o dual de busca em subespaços e o processo de localização gradativa descendente.

as entidades geométricas em **coordenadas homogêneas**[6].

Todavia, a falta de orientação dos espaços projetivos inviabiliza a definição consistente de diversos conceitos fundamentais da geometria euclideana como segmentos de reta, semi-espaços e convexidade. Para contornar tal dificuldade, adiciona-se orientação às entidades geométricas no espaço projetivo, resultando no **espaço projetivo orientado** [Sto91].

Uma correspondência tipicamente utilizada para dualização no plano projetivo mapeia um ponto $[w, a, b]$ (coordenadas homogêneas) do espaço primal na reta $ax + by + w = 0$ do espaço dual. No plano projetivo orientado faz-se uma distinção entre os pontos do lado de cima do plano ($w > 0$) e aqueles do lado de baixo do mesmo ($w < 0$). Como na geometria projetiva tradicional, $w = 0$ indica ponto no infinito, na direção denotada pelas coordenadas $[0, x, y]$, onde $(x, y) \neq (0, 0)$.

A geometria projetiva orientada proporciona as vantagens da geometria projetiva, preser-

[6]Por definição, o ponto do plano projetivo cujas coordenadas homogêneas (modelo analítico da geometria projetiva) são $[w, x, y]$ é o mesmo cujas coordenadas cartesianas são $(x/w, y/w)$, com $w \neq 0$. Assim, $[cw, cx, cy] = [w, x, y]$ para qualquer $c \neq 0$ ($c \in \mathbf{R}$).

vando não só as relações de incidência como também as relações de orientação, de maneira consistente.

2.8 Linearização de subespaços de busca

O paradigma de linearização [AM92a, Mat93a] consiste em estabelecer uma correspondência entre um problema de busca em subespaços de algum formato arbitrário em d dimensões e o problema de busca em *semi-espaços* num espaço de dimensão $D > d$. A dimensão D do espaço para onde é mapeado o problema é chamada de **dimensão da linearização**. O termo linearização provém do fato da fronteira de um semi-espaço ser um hiperplano, dado por uma equação linear.

Aplicando este paradigma é possível tratar problemas de busca em subespaços onde a fronteira do subespaço de busca é constituída de trechos de variedades algébricas dadas por equações polinomiais de grau limitado (os subespaços semi-algébricos definidos no final da seção 1.2.2). Estes subespaços de busca são usualmente determinados por conjunções e/ou disjunções de inequações polinomiais do tipo[7] $f(x_1, x_2, \ldots, x_d) \geq 0$, onde $x_1, x_2, \ldots, x_d$ são as coordenadas de um ponto $x \in \Gamma$ e f é um polinômio de grau limitado nestas coordenadas.

Descrição

Considere um problema de busca em subespaço polinomial em $\mathbf{R}^d$. Para empregar o paradigma de linearização é necessário determinar uma função mapeando cada ponto $(x_1, \ldots, x_d) \in \mathbf{R}^d$ em algum ponto $(t_1, \ldots, t_D) \in \mathbf{R}^D$, de tal forma que buscas em subespaços semi-algébricos em $\mathbf{R}^d$ correspondam a buscas em *semi-espaços* em $\mathbf{R}^D$.

Função de linearização:

Uma função $\zeta : \mathbf{R}^d \to \mathbf{R}^D$, com $D > d$ é chamada função de linearização se:

1. ζ é contínua;
2. dado um subespaço polinomial $\sigma_f \in \Sigma_f$, o subespaço afim gerado pela imagem de sua fronteira, $\langle \zeta(\partial\sigma_f) \rangle$ é um hiperplano de dimensão $D - 1$.

Decorre da continuidade de ζ que a imagem $\zeta(\sigma_f)$ de um subespaço de busca polinomial $\sigma_f \in \Sigma_f$ está inteiramente contida em um dos *semi-espaços* de $\mathbf{R}^D$ delimitados por $\langle \zeta(\partial\sigma_f) \rangle$, donde se pode estabelecer a correspondência do problema de busca em subespaços original em

[7] As inequações também podem ser estritas ($f(x_1, x_2, \ldots, x_d) > 0$). Os subespaços por elas determinados recebem o mesmo tratamento.

$\mathbf{R}^d$ com busca em *semi-espaços* em $\mathbf{R}^D$, isto é:

$$x \in \sigma_f \iff \zeta(x) \in \zeta(\sigma_f) \ \forall x \in \mathbf{R}^d \tag{2.7}$$

Desta forma, soluções originalmente propostas para busca em semi-espaços (capítulo 6), tipicamente soluções envolvendo algum tipo de árvore de partição (seção 2.3), podem ser utilizadas (às vezes mediante algumas adequações) para solucionar problemas de busca em subespaços polinomiais.

As soluções para busca em subespaços semi-algébricos podem ser obtidas diretamente das soluções para subespaços polinomiais, utilizando esquemas de partição produzindo soluções eficientes para diversas inequações polinomiais simultaneamente ou então, compondo diversas estruturas para busca em subespaços polinomiais em uma estrutura de dados multinível (seção 2.4), de modo que cada nível trate uma das restrições polinomiais.

Exemplo: linearização de busca em subespaço circular

Um disco fechado $\sigma(o, r)$ em $\mathbf{R}^2$, de centro em $o = (o_x, o_y)$ e raio r é o conjunto de pontos (x, y) satisfazendo a inequação polinomial $(x - o_x)^2 + (y - o_y)^2 \leq r^2$. Re-arranjando os termos desta inequação, temos:

$$\sigma(o, r) = \{(x, y) \in \mathbf{R}^2 \mid c + 2o_x x + 2o_y y - x^2 - y^2 \geq 0\} \tag{2.8}$$

onde $c = r^2 - {o_x}^2 - {o_y}^2$.

Considere o hiperplano:

$$h = \{(t_1, t_2, t_3, t_4) \in \mathbf{R}^4 \mid c + t_1 + t_2 + t_3 + t_4 = 0\}.$$

Mapeando cada ponto $(x, y) \in \mathbf{R}^2$ num ponto $(t_1, t_2, t_3, t_4) \in \mathbf{R}^4$ com $t_1 = 2o_x x$, $t_2 = 2o_y y$, $t_3 = -x^2$ e $t_4 = -y^2$ temos que:

$$(x, y) \in \sigma(o, r) \iff (t_1, t_2, t_3, t_4) \in \sigma(h) \tag{2.9}$$

onde:

$$\sigma(h) = \{(t_1, t_2, t_3, t_4) \in \mathbf{R}^4 \mid c + t_1 + t_2 + t_3 + t_4 \geq 0\} \tag{2.10}$$

O problema de busca em subespaço circular no plano é portanto reduzido a busca em semi-espaço em 4 dimensões. Porém, cabe observar que a linearização descrita acima *não* é a de menor dimensão para discos em $\mathbf{R}^2$.

A transformação conhecida como elevação ao parabolóide possibilita uma linearização de dimensão 3, mapeando os pontos de $\mathbf{R}^2$ em pontos do parabolóide $z = x^2 + y^2$. Seja $\zeta : \mathbf{R}^2 \to \mathbf{R}^3$ a transformação que leva o ponto (x, y) no ponto $\zeta(x, y) = (x, y, x^2 + y^2)$ de $\mathbf{R}^3$. O leitor pode

verificar que pontos co-circulares são mapeados por ζ em pontos coplanares e que a imagem do interior do disco $\sigma(o, r)$ em $\mathbf{R}^2$ é a intersecção do parabolóide com um dos semi-espaços abertos determinados pelo plano que contém a imagem da fronteira de $\sigma(o, r)$. Portanto, determinar os pontos de P contidos no disco $\sigma(o, r)$ equivale a determinar os pontos de $\zeta(P)$ contidos nesse semi-espaço.

A transformação de elevação ao parabolóide é ilustrada na figura 2.15.

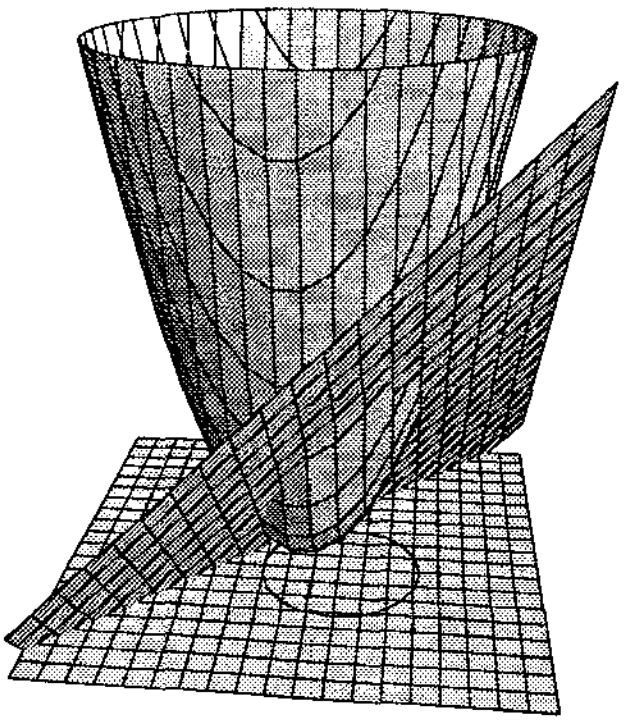


Figura 2.15: A transformação de elevação ao parabolóide.

A procura de linearizações adequadas

Agarwal e Matoušek [AM92a, Mat93a] concluem a partir do trabalho de Yao e Yao [YY85], que todo polinômio é linearizável, isto é, admite uma função de linearização.

Um método geral para se obter uma linearização é fazer corresponder uma coordenada t_i de $\mathbf{R}^D$ a cada monômio nas coordenadas de $\mathbf{R}^d$ aparecendo no polinômio f. Porém, este método produz linearizações de alta dimensão (exponencial no grau de f), tornando tais soluções muito dispendiosas.

Na procura de uma função de linearização, pode-se agrupar os monômios de f de diversas maneiras distintas, para formar as funções coordenadas de $\zeta(x) = (\zeta_1(x), \ldots, \zeta_D(x))$. O fun-

damental para se ter uma linearização é que a fronteira de σ_f, determinada pelo polinômio f, quando mapeada por ζ em $\mathbf{R}^D$ determine um hiperplano denotado por uma equação da forma:

$$\sum_{i=1}^{D} \alpha_i \zeta_i(x) = 0 \tag{2.11}$$

onde cada $\zeta_i(x)$ (cada qual corresponde a uma coordenada de $\mathbf{R}^D$) é um polinômio em x obtido de somas de monômios de f e os α_i's são coeficientes determinados a partir dos coeficientes de f.

Um problema em aberto é conseguir abordagens algorítmicas para se obter linearizações da menor dimensão possível (quem sabe de dimensão polinomial no grau de f).

Aplicações e extensões

A aplicação direta do paradigma de linearização possibilita as melhores soluções conhecidas para busca em alguns tipos de subespaços determinados por polinômios. Para outros tipos de subespaços, entretanto, são obtidos algoritmos mais eficientes generalizando métodos de busca em subespaços de fronteira linear (tipicamente semi-espaços e simplexos) para o caso não linear. Por exemplo, para busca em subespaço circular no plano, a aplicação direta de linearização em 3 dimensões resulta em um algoritmo com tempo de busca $O(N^{2/3})$, utilizando uma quantidade linear de memória. Todavia, substituindo semi-espaços por círculos em algumas soluções para busca em semi-espaços (por exemplo [Mat91c]) e realizando as generalizações necessárias, pode-se obter tempo de busca $O(\sqrt{N})$, com memória linear.

Algoritmos inspirados em linearização e baseados em generalização de soluções propostas para busca em semi-espaços e para buscas simpliciais são abordados por Agarwal e Matoušek [AM92a, Mat93a]. Eles propõem uma estrutura para busca em subespaços determinados por polinômios com uso de memória e tempo de pré-processamento linear, permitindo efetuar buscas em tempo $O(N^{1-1/b+\varepsilon})$, onde b é uma constante tal que $d \leq b \leq 2d-3$ e $\varepsilon > 0$ é uma constante arbitrária.

Agarwal e Matoušek conjeturam que nas soluções para busca em subespaços polinomiais que utilizam memória linear, o tempo de busca depende do número de dimensões d e que nas soluções com tempo de busca poli-logarítmico, o uso de memória depende do número de parâmetros K do vetor de parâmetros $a = (a_1, \ldots, a_K)$, relativo ao polinômio $f(x, a)$ (seção 1.2.2). Baseados nas medidas de desempenho obtidas para busca em subespaço circular no plano, eles levantam a suposição que o tempo mínimo de busca com memória linear seja $O(N^{1-1/d})$ e que a quantidade de memória necessária para efetuar busca em tempo poli-logarítmico seja $O(N^{K+1})$.

Entretanto, a prova de tal suposição para $d \geq 3$ é um problema em aberto. Esta questão esbarra no problema de decompor um arranjo de variedades algébricas ("hiper-superfícies" no espaço d-dimensional, determinadas por polinômios de grau limitado) em um conjunto de células simples (com complexidade de descrição limitada), um dos maiores problemas em aberto em geometria computacional atualmente.

Eles concluem que se um arranjo de m variedades algébricas em $\mathbf{R}^d$ puder ser decomposto em $O(m^b)$ células simples, onde $b > 0$ é uma constante, então será possível obter tempo de busca $O(N^{1-1/b+\epsilon})$ com memória linear ou então tempo de busca poli-logarítmico com utilização de memória proporcional a $O(N^{b+\epsilon})$.

Capítulo 3

Busca em subespaço ortogonal

O problema de busca em subespaço ortogonal, como vimos na seção 1.2.2, consiste em determinar (ou contar) os pontos da base de dados P contidos em um hiper-retângulo isotético σ, dado pelo produto cartesiano dos intervalos de busca $\sigma_i = [a_i, b_i]$ das várias dimensões ($1 \leq i \leq d$). A figura 3.1-a ilustra o problema de busca em subespaço ortogonal no plano e a figura 3.1-b mostra as faixas no plano correspondentes aos intervalos nas várias dimensões.

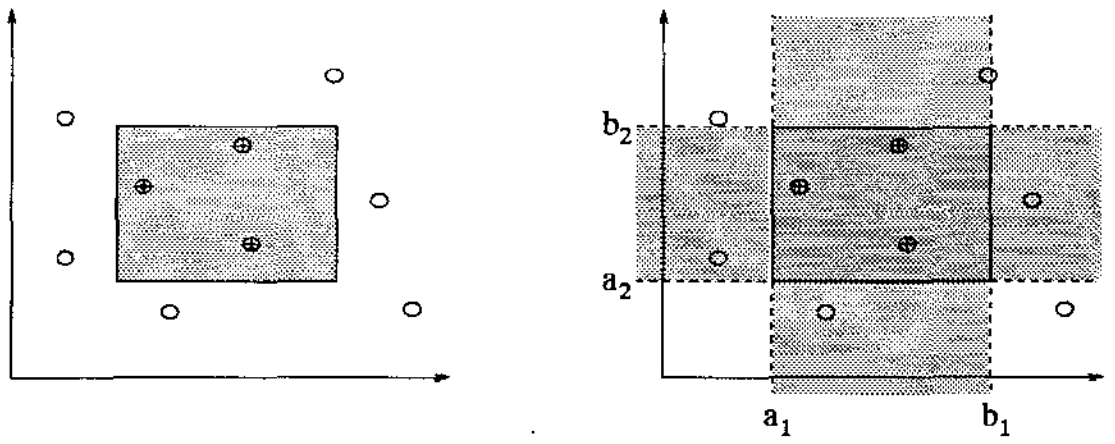


Figura 3.1: Busca em um subespaço ortogonal no plano (a) e o mesmo subespaço visto como a interseção de faixas correspondentes aos intervalos nas duas dimensões (b).

O fato do subespaço de busca σ ser um hiper-retângulo isotético permite calcular interseções de σ com sub-regiões do espaço multidimensional mais facilmente do que se fossem considerados subespaços de busca de formatos arbitrários. As operações de teste envolvidas nas soluções de busca em subespaço ortogonal freqüentemente são bem mais simples do que aquelas requeridas por outros tipos de subespaços de busca. A consulta de contenção de um ponto em um hiper-retângulo d dimensional, por exemplo, requer apenas $2d$ comparações, relativas aos extremos dos

intervalos $\sigma_i = [a_i, b_i]$. Esta característica já é suficiente para tornar o problema de busca em subespaço ortogonal mais fácil de ser tratado, pois torna os algoritmos mais simples e minimiza o tamanho das constantes multiplicativas embutidas nas equações de complexidade das soluções. A *KD-Tree* (seção 3.1.1) e *Quad-Tree* (seção 3.1.2) são exemplos de estruturas que, embora possam ser utilizadas para outros tipos de consultas (especificamente busca em subespaços de outros formatos), têm algoritmos mais eficientes para busca em subespaço ortogonal do que para esses outros tipos de consultas.

Além disso, no caso ortogonal cada intervalo de busca σ_i identifica uma faixa em Γ contendo o subespaço de busca (veja figura 3.1-b). Utilizando esta propriedade é possível transformar o problema d-dimensional em um certo número (preferencialmente pequeno) de problemas $(d-1)$-dimensionais. Tal técnica aplicada recursivamente constitui o paradigma de divisão e conquista multidimensional (seção 2.4), o qual provê a concepção algorítmica básica da estrutura *Range-Tree* (seção 3.1.3), uma das mais eficientes para efetuar busca em subespaço ortogonal. Aplicando-se os paradigmas de buscas por propagação (seção 2.5) e abordagem funcional para estruturas de dados (seção 2.6) é possível alcançar a solução ótima, para alguns casos de busca em subespaço ortogonal.

3.1 Algumas soluções

Dentre as estruturas mais difundidas e eficientes para solucionar o problema de busca em subespaço ortogonal estão a *KD-Tree*, a *Quad-Tree* e a *Range-Tree*. Existem diversas variações de cada uma destas estruturas na literatura (veja seção 3.3), há variações com diferentes relações de compromisso entre as medidas de desempenho e podem ser aplicadas diversas técnicas sobre estas estruturas para melhorar o desempenho (muitas vezes sob determinadas restrições do problema). Neste trabalho, entretanto, estas estruturas são descritas apenas em sua forma mais típica.

A *KD-Tree*, a *Quad-Tree* e a *Range-Tree* baseiam-se todas em estruturas de árvores, onde cada nó v é associado a uma região $\mathcal{R}(v)$ do espaço multidimensional Γ e a um conjunto de pontos $\mathcal{P}(v) = P \cap \mathcal{R}(v)$. Cada região $\mathcal{R}(v)$ correspondente a uma face ou conjunto de faces de uma partição de Γ.

A *KD-Tree* e a *Quad-Tree* são bastante semelhantes. As partições do plano efetuadas por uma *KD-Tree* e uma *Quad-Tree* são ilustradas nas figuras 3.2-a e 3.2-b, respectivamente. As divisões do espaço Γ se alternam entre as as dimensões na *KD-Tree* e são efetuadas em todas as direções em cada nó da árvore, no caso da *Quad-Tree*; mas em ambas a partição do espaço é representada em uma única estrutura de árvore. As duas estruturas ocupam memória linear e podem efetuar buscas em tempo $O(N^{1-1/d})$ no pior caso e $O(\log N)$ no caso médio.

A *Range-Tree*, por sua vez, efetua a partição do espaço multidimensional Γ independentemente em cada uma das dimensões, de acordo com o paradigma de divisão e conquista multidimensional (seção 2.4). Há uma árvore primária correspondente a uma partição na primeira dimensão, tal que cada nó aponta para uma estrutura que irá auxiliar a solucionar o problema a partir da próxima dimensão, recursivamente e considerando somente os pontos de $\mathcal{P}(v)$. Uma

partição do plano efetuada por uma *Range-Tree* é ilustrada na figura 3.2-c. A *Range-Tree* ocupa memória $O(N \log^{d-1} N)$ e permite efetuar buscas em tempo $O(\log^{d-1} N)$.

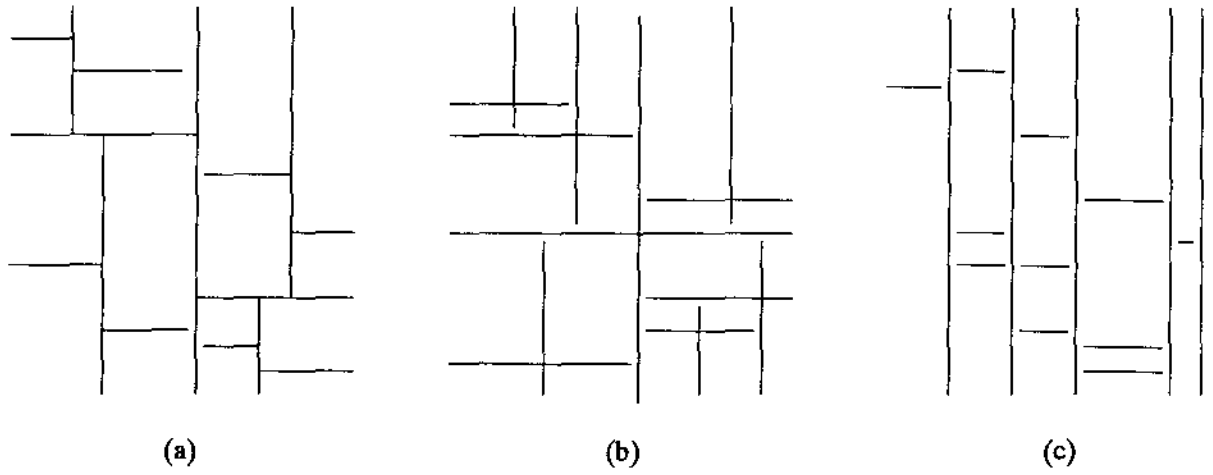


Figura 3.2: Partições do plano efetuadas por algumas estruturas voltadas para busca em subespaço ortogonal: (a) *KD-Tree*, (b) *Quad-Tree* e (c) *Range-Tree*.

Embora a *Range-Tree* apresente melhor desempenho assintótico de pior caso nas buscas, muitas vezes as estruturas mais adequadas em aplicações práticas são a *KD-Tree* e a *Quad-Tree*, pois podem alcançar melhor desempenho no caso médio [BS75] e são mais versáteis, permitindo busca em subespaços de outros formatos (por exemplo, subespaços circulares e semi espaços) com medidas de desempenho análogas (salvo pelas constantes multiplicativas) [Ben90]. Além disso, o uso de memória de uma *Range-Tree* torna-se elevado para dimensões mais altas[1].

A seguir, essas três estruturas são descritas mais detalhadamente.

3.1.1 *KD-Tree*

A estrutura *KD-Tree* [PS85, Ben75] é também conhecida por árvore binária de busca multidimensional (*multidimensional binary search tree*) e foi uma das primeiras estruturas empregadas para solucionar buscas em espaços multidimensionais. O K (maiúsculo) de *KD-Tree* diz respeito ao número de dimensões do espaço Γ. Ficamos com o nome *KD-Tree* a fim de manter a nomenclatura usualmente empregada na literatura. Assim, considere nesta seção $K = d$. O tamanho da saída continua sendo denotado por k (minúsculo).

Uma vantagem significativa da *KD-Tree* na solução de problemas de busca em subespaços é sua versatilidade, permitindo o processamento eficiente de vários tipos de consultas. Além de busca em subespaço ortogonal, a *KD-Tree* pode também ser empregada na solução de consultas de vizinhança e busca em subespaço circular (seção 4), por exemplo.

[1] Embora seja $o(N^2)$ para qualquer d constante.

Descrição da estrutura

A *KD-Tree* é baseada em divisões sucessivas do domínio de busca por meio de hiperplanos perpendiculares aos eixos coordenados, de modo que cada lado de uma divisão de uma região por um hiperplano contenha (aproximadamente) metade dos pontos daquela região.

Considere os valores de cada coordenada todos distintos para os pontos da base de dados P, para efeito de simplificação da descrição a seguir[2]. A cada nó v da *KD-Tree* é associada uma região $\mathcal{R}(v)$ do espaço multidimensional Γ e o conjunto de pontos $\mathcal{P}(v) = P \cap \mathcal{R}(v)$, isto é, o subconjunto dos pontos de P contidos no interior de $\mathcal{R}(v)$. Toma-se a mediana dos pontos de $\mathcal{P}(v)$ segundo uma das coordenadas e o ponto correspondente à mediana é armazenado no nó v. O valor da mediana define um hiperplano, perpendicular à coordenada em questão, que divide $\mathcal{R}(v)$ em duas partes disjuntas $\mathcal{R}_{inf}(v)$ e $\mathcal{R}_{sup}(v)$, cada qual contendo aproximadamente metade dos pontos de $\mathcal{P}(v)$. $\mathcal{R}_{inf}(v)$ ($\mathcal{R}_{sup}(v)$) contém os pontos com valores da coordenada em questão menores (maiores) que o valor da mediana. Cada uma dessas duas regiões será associada a um dos filhos de v.

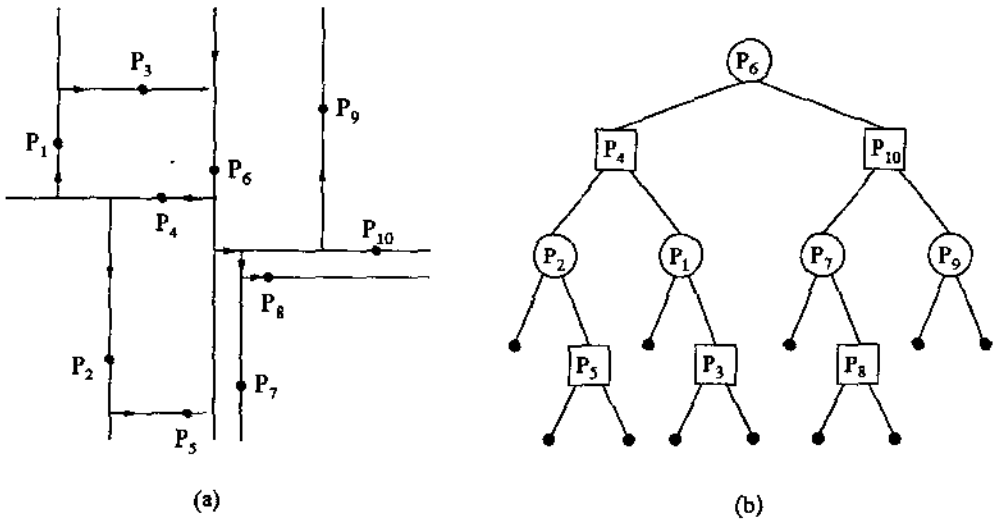


Figura 3.3: Uma *KD-Tree* para $P = \{p_1, \ldots, p_{10}\}$. A parte (a) mostra uma subdivisão do espaço e a parte (b) mostra a *KD-Tree* correspondente.

À raiz da *KD-Tree* é associado todo o espaço de busca Γ e a divisão do espaço é efetuada perpendicularmente à primeira coordenada, da forma mencionada acima. Procede-se recursivamente nos filhos, alternado a coordenada de corte a cada nível da árvore, ciclicamente. A subdivisão cessa quando são obtidas regiões com nenhum ponto da base de dados P e essas regiões são associadas às folhas da *KD-Tree*. A partição do espaço bidimensional para um conjunto de dez pontos e a *KD-Tree* correspondente são ilustrados na figura 3.3, na qual nós circulares representam cortes verticais e nós quadrados cortes horizontais.

[2]Formas de tratamento de situações patológicas podem ser encontradas em [OvL82] e [Ben90].

O algoritmo de busca utilizando a *KD-Tree*

Vejamos agora o uso da *KD-Tree* para efetuar busca em subespaço ortogonal. Considere a interação de uma região $\mathcal{R}(v)$, associada ao nó v da *KD-Tree* com o subespaço de busca σ, de modo que se tenha uma interseção não vazia (isso certamente acontece na raiz, considerando $\sigma \neq \emptyset$). A região $\mathcal{R}(v)$ é dividida em duas partes $\mathcal{R}_{inf}(v)$ e $\mathcal{R}_{sup}(v)$, cada qual associada a um dos filhos de v e o ponto de mediana na direção sendo considerada, $p(v)$, no qual é efetuada a divisão de $\mathcal{R}(v)$ fica armazenado em v. Então, basta testar se o ponto $p(v)$ associado ao nó v está em σ (neste caso ele faz parte da saída) e prosseguir a busca somente no(s) filho(s) cujas regiões associadas tenham interseção não vazia com o subespaço de busca $(\mathcal{R}_i(v) \mid \mathcal{R}_i(v) \cap \sigma \neq \emptyset;\ i \in \{inf, sup\})$.

Um algoritmo simples para busca em uma *KD-Tree* é especificado abaixo. Denota-se por $v.esq$ ($v.dir$) o filho à esquerda (direita) do nó v, $p(v)$ o ponto de mediana associado ao nó v, $p(v).val[i]$ o valor da i-ésima coordenada do ponto $p(v)$ e $\sigma.a[i]$ ($\sigma.b[i]$) o extremo inicial (final) do intervalo correspondente à i-ésima coordenada no subespaço de busca $\sigma = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$. Uma busca na *KD-Tree* T, pelos pontos contidos em σ, partindo da primeira coordenada é especificada por Busca(raiz(T),σ,1), utilizando a função abaixo.

```
function Busca(v,σ,i)
begin
   if v não é folha then
      if (p(v) ∈ σ) then p(v) está no subespaço;
      if (σ.a[i] ≤ p(v).val[i]) then Busca(v.esq,σ,(i mod d) + 1);
      if (σ.b[i] ≥ p(v).val[i]) then Busca(v.dir,σ,(i mod d) + 1);
      ((i mod d) + 1 denota a próxima coordenada)
end;
```

O tempo de busca é demonstrado ser sub-linear [LW77], quando $k \in o(N)$. A figura 3.4 ilustra uma busca na *KD-Tree* da figura 3.3, indicando os nós visitados pelo procedimento de busca.

Complexidades

C: $O(N^{1-1/d} + k)$ $\qquad (d \geq 2)$
M:$O(N)$
P: $O(N \log N)$

O comportamento da *KD-Tree* é ótimo em termos do uso de memória. Entretanto, o mau desempenho de pior caso do procedimento de busca motivou o desenvolvimento de outros métodos

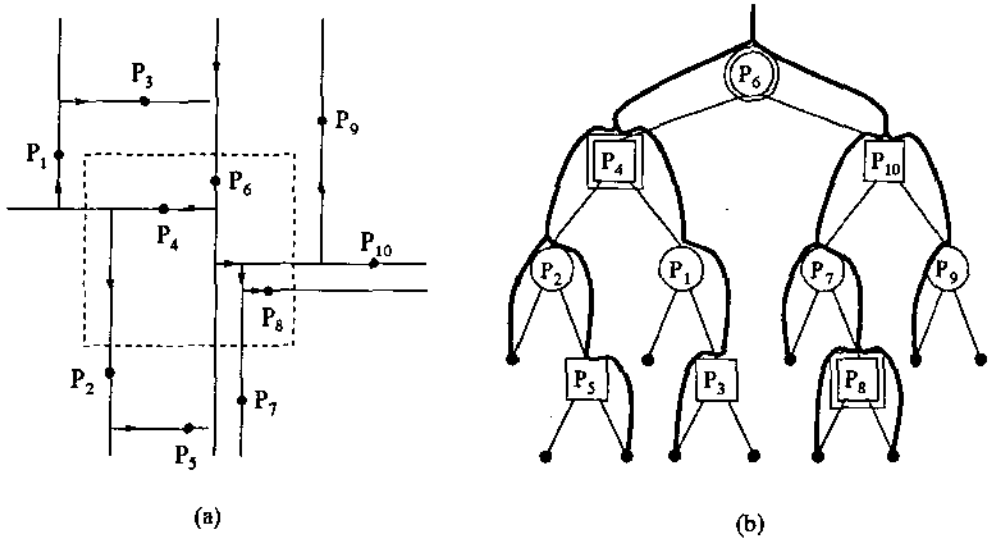


Figura 3.4: Procedimento de busca em uma *KD-Tree*. Na parte (a) vê-se o subespaço de busca e na parte (b) o caminho percorrido na árvore.

permitindo buscas mais eficientes, ainda que às custas de maior uso de memória.

3.1.2 *Quad-Tree*

Na *Quad-Tree* de pontos (*point Quad-Tree*)[3] [FB74, BS75], da mesma forma que na *KD-Tree*, cada nó v corresponde a uma região $\mathcal{R}(v)$ do espaço Γ e ao conjunto de pontos $\mathcal{P}(v) = P \cap \mathcal{R}(v)$, contidos no interior da mesma. Em cada nó v de uma *Quad-Tree* também se considera uma partição de $\mathcal{R}(v)$, no ponto médio de $\mathcal{P}(v)$ (médio segundo uma de suas coordenadas) que é armazenado em v, porém, a *cada nó* v considera-se *quebras de* $\mathcal{R}(v)$ *em todas as dimensões*, nas posições dadas pelas coordenadas do ponto médio. Assim, um nó interno de uma *Quad-Tree* em $\mathbf{R}^2$ possui sempre 4 filhos, correspondendo aos quadrantes. Uma "*Quad-Tree*" em $\mathbf{R}^3$ pode ser chamada de *Octal-Tree*, uma vez que cada nó interno possui 8 filhos (relativos aos 8 octantes). Generalizando, em d dimensões cada nó interno tem 2^d filhos. Uma partição do espaço bidimensional por uma *Quad-Tree* e a estrutura correspondente são ilustradas na figura 3.5.

As medidas de complexidade da *Quad-Tree* são as mesmas da *Range-Tree*.

[3]Sempre que nos referimos a *Quad-Trees* estamos falando de *Quad-Trees* de pontos. As *Quad-Trees* no sentido geral são também utilizadas para representação de objetos geométricos, aproximando a forma destes objetos por uma hierarquia de quadrados em $\mathbf{R}^2$ ou cubos em $\mathbf{R}^3$. Neste último caso, as estruturas são chamadas *Octal-Trees* e são generalizações de *KD-Trees* para 3 dimensões.

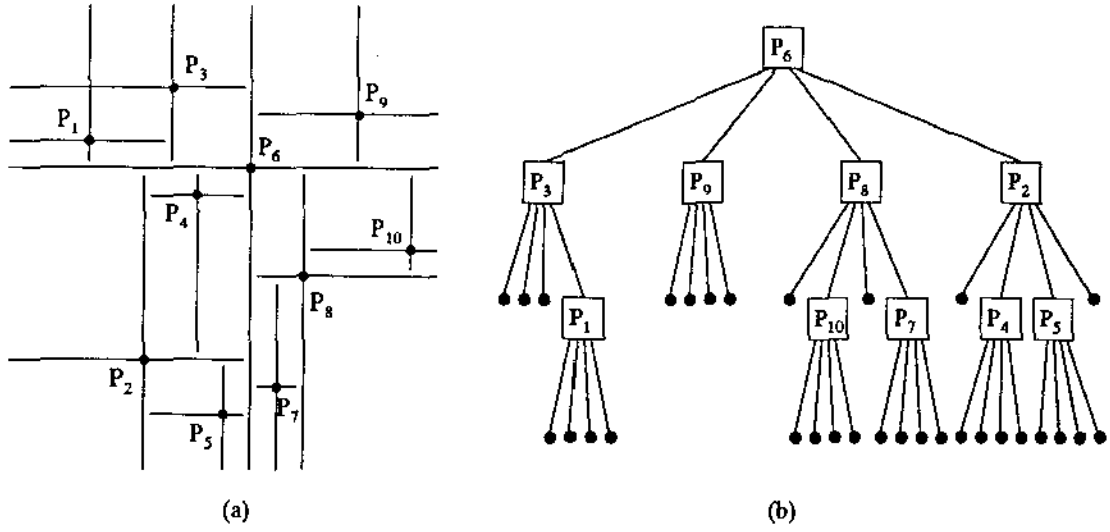


Figura 3.5: A *Quad-Tree* para $P = \{p_1, \ldots, p_{10}\}$. A parte (a) mostra uma subdivisão do espaço e a parte (b) mostra a *Quad-Tree* correspondente.

Complexidades

C: $O(N^{1-1/d} + k)$ $(d \geq 2)$
M: $O(N)$
P: $O(N \log N)$

3.1.3 *Range-Tree*

Para explicar a estrutura da *Range-Tree* [PS85, Ben80, Wil82a, Lue79, Wil78, Lue78], vamos primeiramente definir a estrutura denominada árvore de segmentos, na qual ela se baseia.

A Árvore de segmentos

Uma árvore de segmentos (*segment tree*) subdivide um domínio linear γ em uma hierarquia de partições em segmentos, denominados **segmentos canônicos**. Cada partição do domínio corresponde a um nível[4] da árvore de segmentos, isto é, a união dos segmentos canônicos associados a um mesmo nível da árvore é igual ao domínio γ e a interseção de qualquer par dos mesmos é nula. As partições componentes desta hierarquia são cada vez mais finas a medida que se desce nos níveis da árvore e são realizadas de acordo com um conjunto P de elementos de dados armazenados, tomados em γ.

Seja uma árvore de segmentos para um conjunto P de N valores tomados em γ. A cada

[4] Nível aqui se refere à profundidade na árvore.

nó v desta árvore corresponde um segmento $\gamma(v)$ e o conjunto de elementos de P contidos em $\gamma(v)$, denotado por $\mathcal{P}(v)$. A raiz da árvore de segmentos corresponde a todo o domínio γ e portanto, a todos os elementos de P. A divisão em cada nó v é realizada no valor da mediana dos valores de $\mathcal{P}(v)$, sendo o filho à esquerda (direita) associado à porção do segmento $\gamma(v)$ na parte inferior (superior) da mediana, denotada por $\gamma_{inf}(v)$ ($\gamma_{sup}(v)$), e os pontos nela contidos $P_{inf}(v)$ ($P_{sup}(v)$). Esse processo é aplicado recursivamente na descendência dos nós formados, até as folhas que ficam em correspondência um a um com os intervalos entre valores adjacentes de P.

Um intervalo arbitrário em γ é subdividido por uma árvore de segmentos em, no máximo, $(2 \cdot \lceil \log N \rceil - 2)$ intervalos canônicos, como ilustrado na figura 3.6, na qual se pode ver uma árvore de segmentos armazenando os valores inteiros de 4 a 15 e a partição do intervalo fechado [5, 14] em cinco segmentos canônicos, correspondentes aos nós realçados.

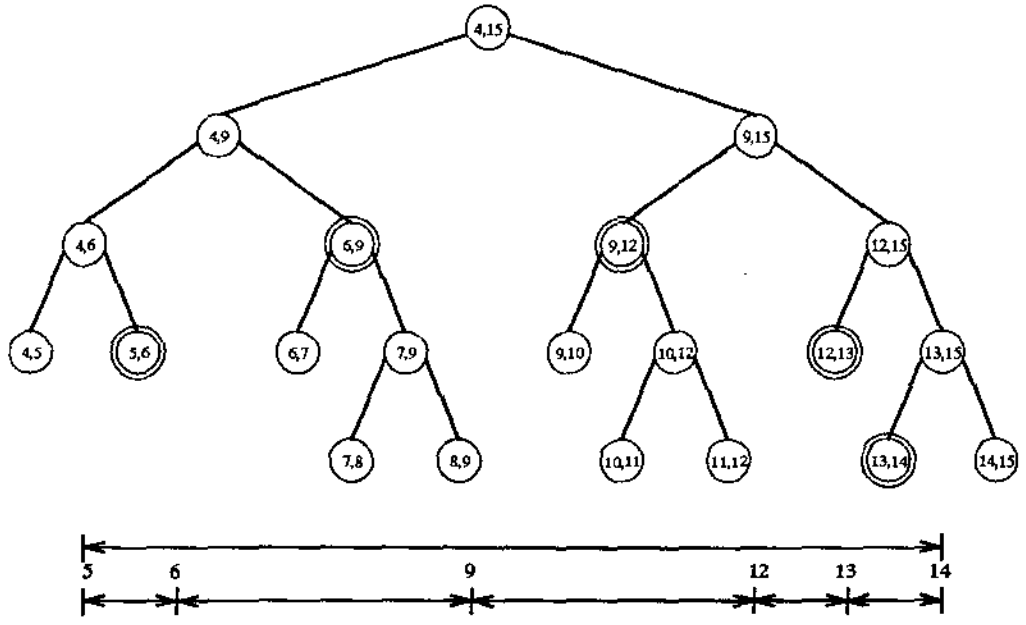


Figura 3.6: Uma árvore de segmentos e a partição de um intervalo em uma seqüência de intervalos canônicos.

A *Range-Tree*

A estrutura *Range-Tree* é um exemplo típico de aplicação do paradigma de divisão e conquista multidimensional (veja seção 2.4). Ela utiliza árvores de segmentos para efetuar buscas nas várias dimensões e há uma única árvore de segmentos para a primeira dimensão (γ_1), denominada **árvore de segmentos primária**. Cada nó v da árvore de segmentos primária corresponde a um intervalo canônico dentro do domínio γ_1, sendo cada qual *ligado a uma estrutura secundária*, que pode ser vista como uma *Range-Tree* definida recursivamente para a próxima dimensão, considerando os elementos de $\mathcal{P}(v)$ a partir da segunda coordenada ($P(v)' = \{p' = (x_2, \ldots, x_d); p =$

$(x_1, x_2, \ldots, x_d) \in \mathcal{P}(v)\}$). Na última dimensão, tem-se vetores ordenados ou árvores de busca para os conjuntos de pontos considerados.

Busca em subespaço ortogonal utilizando a *Range-Tree*

Para efetuar busca em subespaço ortogonal utilizando uma *Range-Tree*, busca-se os valores correspondentes aos extremos a_1 e b_1 do intervalo de busca $\sigma_1 = [a_1, b_1]$ da primeira dimensão na árvore de segmentos primária, identificando-se os nós correspondentes aos intervalos canônicos que compõem σ_1. Então, procede-se recursivamente nas estruturas apontadas por esses nós, considerando (no máximo $(2 \cdot \lceil \log N \rceil - 2)$) problemas análogos decrescidos em uma dimensão e levando em conta somente os pontos contidos em cada intervalo canônico. A complexidade resultante para o tempo de busca é $O(\log^d N)$, onde N é o tamanho da base de dados e d é o número de dimensões. A figura 3.7 ilustra uma *Range-Tree* e o procedimento de busca.

O processo de busca pode ser agilizado aplicando-se buscas por propagação (seção 2.5) na última dimensão da *Range-Tree*, onde se tem vetores ordenados pelos valores de chave da última dimensão. Adiciona-se ponteiros entre vetores correspondentes a nós adjacentes nas árvores de segmentos da dimensão anterior, a intervalos regulares, de forma a permitir a propagação de valores de chave de cada nó para os seus filhos. Isso permite que uma busca, realizada no vetor correspondente à raiz de uma árvore de segmentos na dimensão anterior, possa ser propagada para os vetores relativos aos nós descendentes. Com este processo de propagação, evita-se repetir as buscas binárias nos vetores correspondentes a nós não raiz na dimensão anterior, resultando na economia de um fator $O(\log N)$ do tempo de busca. A *Range-Tree* em que as estruturas de busca na última dimensão são acrescidas destes ponteiros auxiliares é denominada *Range-Tree* com pontes (*bridged range tree*) [LW82, Wil85, PS85].

Complexidades

C: $O(\log^{d-1} N + k)$ $\quad (d \geq 2)$
M: $O(N \log^{d-1} N)$
P: $O(N \log^{d-1} N)$

Aplicando-se a transformação de sobreposição à *Range-Tree* para duas dimensões analisada com a abordagem funcional (seção 2.6), consegue-se reduzir o uso de memória para linear, sem afetar o tempo de busca em problemas de contagem e aumentando-o para $O\left(k\left(\log \frac{2N}{k}\right)^{\varepsilon} + \log N\right)$ em problemas de enumeração, onde k é o tamanho da saída e $\varepsilon > 0$ é uma constante [Cha88].

3.2 Cotas inferiores

Os trabalhos mais antigos a respeito de limites inferiores para busca em subespaço ortogonal que encontramos são [Lue78, Fre81a, Yao85, Vai89] e os limites inferiores obtidos mais recentemente

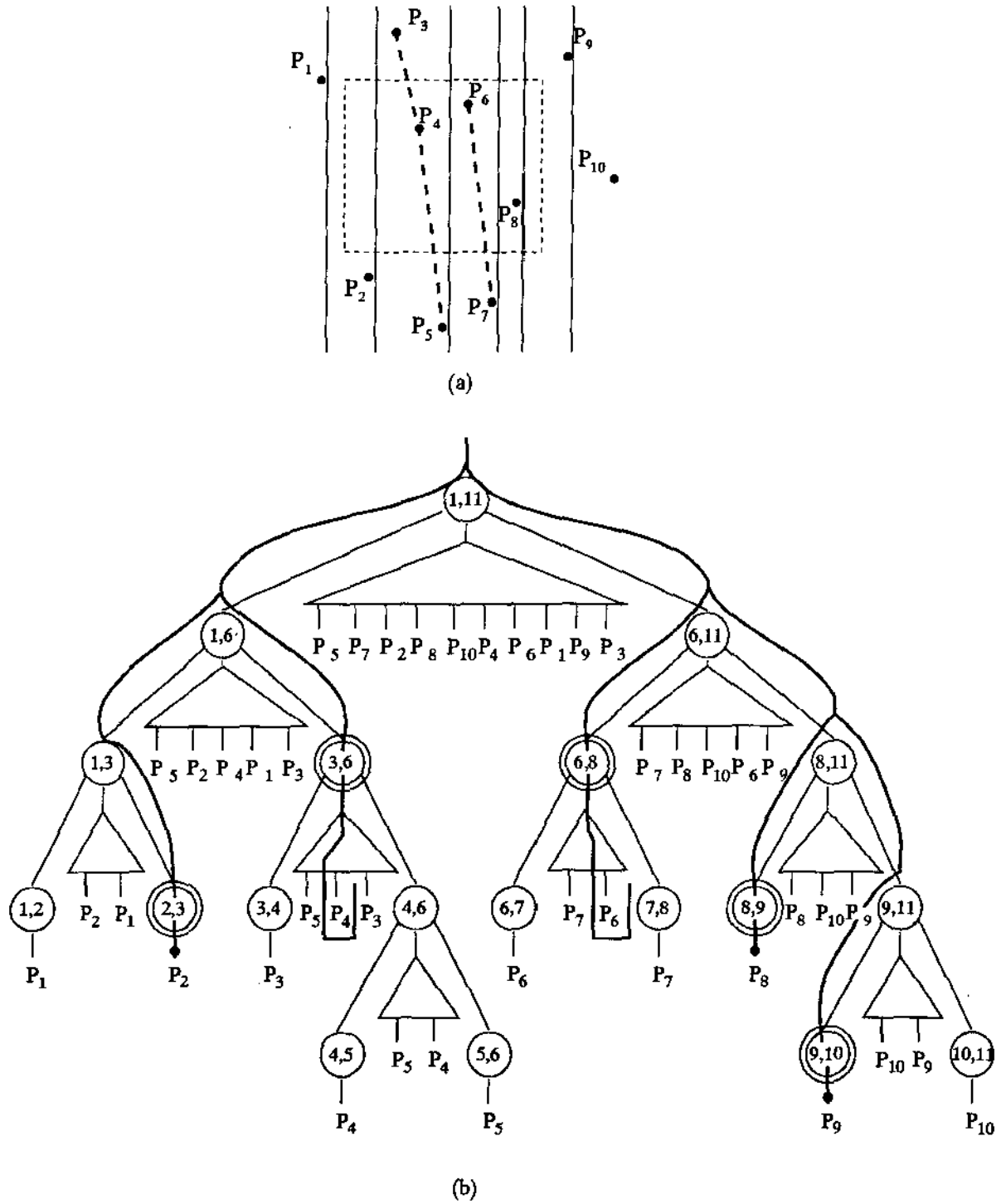


Figura 3.7: Busca em uma *Range-Tree* para o conjunto de pontos $P = \{p_1, \ldots, p_{10}\}$ no plano. A parte (a) mostra a distribuição dos pontos no plano, o subespaço de busca e os segmentos canônicos da primeira dimensão utilizados na busca. A parte (b) mostra a *Range-Tree* correspondente e o percurso de busca nela efetuado.

são descritos a seguir.

Willard [Wil89] demonstra que em d dimensões $\Omega(\log^d N)$ é um limite inferior para o tempo de execução de operações de busca, inserção e remoção, relativas ao caso totalmente dinâmico do problema de busca em subespaço ortogonal, no modelo aritmético (seção 1.5.1).

Chazelle publicou dois artigos recentes com limites inferiores para busca em subespaço ortogonal no caso estático [Cha90a, Cha90b]. No primeiro artigo [Cha90a] é demonstrado que, no modelo de máquinas de ponteiros (seção 1.5), um tempo de consulta em modo de *enumeração* $O(\log^c N + k)$, com $c > 0$ constante só pode ser atingido às custas de memória:

$$M : \Omega\left(N \cdot \left(\frac{\log N}{\log\log N}\right)^{d-1}\right) \tag{3.1}$$

onde d é o número de dimensões.

No segundo artigo [Cha90b], Chazelle demonstra que com uso de memória $O(m)$ o tempo necessário para *contagem* dos pontos contidos em um subespaço ortogonal, tanto no pior caso quanto no caso médio é:

$$C : \Omega\left(\left(\frac{\log N}{\log(2m/N)}\right)^{d-1}\right) \tag{3.2}$$

no modelo aritmético.

Note que estes limites (equações 3.1 e 3.2) estão bastante próximos das cotas superiores obtidas com o uso da *Range-Tree* (seção 3.1.3), permitindo concluir que busca em subespaço ortogonal é um problema essencialmente resolvido, do ponto de vista das medidas assintóticas de complexidade.

3.3 Notas bibliográficas

As resenhas de Bentley e Friedman [BF79] e de Bentley e Maurer [BM80] fornecem indicações dos primeiros resultados obtidos para busca em subespaço ortogonal. As obras de Mehlhorn [Mel84], Preparata e Shamos [PS85] e Sedgewick [Sed88] contêm descrições de algumas soluções para busca em subespaço ortogonal, comparações entre as mesmas e alguns dos primeiros limites inferiores estabelecidos para o problema, nas seções relativas a busca em subespaços.

Diversos trabalhos têm sido publicados ao longo das duas últimas décadas a respeito de soluções para busca em subespaço ortogonal. Análises de custo de operações de busca em *KD-Trees* e *Quad-Trees* podem ser encontradas em [FB74, BS75, LW77, FGPR91], suporte a operações de atualização em [FB74, Ben75, Sam80, OvL82, Ben90] e tratamento de entradas patológicas (pontos com alguma coordenada com o mesmo valor, por exemplo) em [OvL82, Ben90]. Aplicações de *KD-Trees* a outros problemas de busca encontram-se em [Ben75, Ben90] e variações de *KD-Trees* são propostas em [FBF77, Zol78, Spr91]. Silva Filho [Sil81] descreve alguns experimentos a respeito da otimização de *KD-Trees* em memória secundária.

Com respeito às operações de busca e atualização em *Range-Trees* e variações da estrutura, pode-se citar os trabalhos de Bentley, Willard e Lueker [Wil78, Lue78, Lue79, Ben80, Wil82a, Wil85, WL85].

Aplicando o paradigma de filtragens sucessivas (seção 2.2) à solução de busca em subespaço ortogonal baseada em *Range-Tree*, Chazelle [Cha86] conseguiu economizar um fator multiplicativo $1/\log\log N$ no uso de memória da estrutura de dados (seção 3.1.3). Ele conseguiu efetuar consultas em d-dimensões em tempo $O((\log N)^{d-1})$, com utilização de memória $O(N(\log N)^{d-1}/\log\log N)$, provando que o limite superior $O(N(\log N)^{d-1})$ não era uma cota inferior e alcançando o limite superior demonstrado por ele posteriormente (seção 3.2). Além disso, o algoritmo é conceitualmente bastante simples, uma vez que é constituído de blocos de construção independentes e pode ser parametrizado, oferecendo várias possibilidades de relações entre tempo de busca e espaço ocupado pela estrutura pré-processada.

Em [OSdBvK90, SO90] é tratado o problema de armazenamento de *Range-Trees* em memória secundária. No primeiro artigo [OSdBvK90], são apresentadas várias formas de dividir a *Range-Tree* em blocos a serem armazenados na memória secundária, proporcionando diversas relações entre o número de acessos a disco e a quantidade de memória transportada em cada acesso efetuado para realizar consultas ou atualizações. No segundo artigo [SO90], são demonstrados limites inferiores para as divisões da *Range-Tree* em blocos, provando que muitas das divisões empregadas no primeiro artigo são ótimas. Estes são alguns dos poucos trabalhos que encontramos a respeito do armazenamento em memória secundária, de estruturas voltadas para solucionar problemas de busca em subespaços em várias dimensões (veja seção 1.4.5).

Capítulo 4

Busca em subespaço circular

O problema de busca em subespaço circular consiste em determinar os pontos da base de dados P contidos em um "hiper-disco" $\sigma(o, r)$ de dimensão d, com centro $o \in \Gamma$ e raio $r \in \mathbf{R}$. Uma busca em subespaço circular é ilustrada na figura 4.1-a.

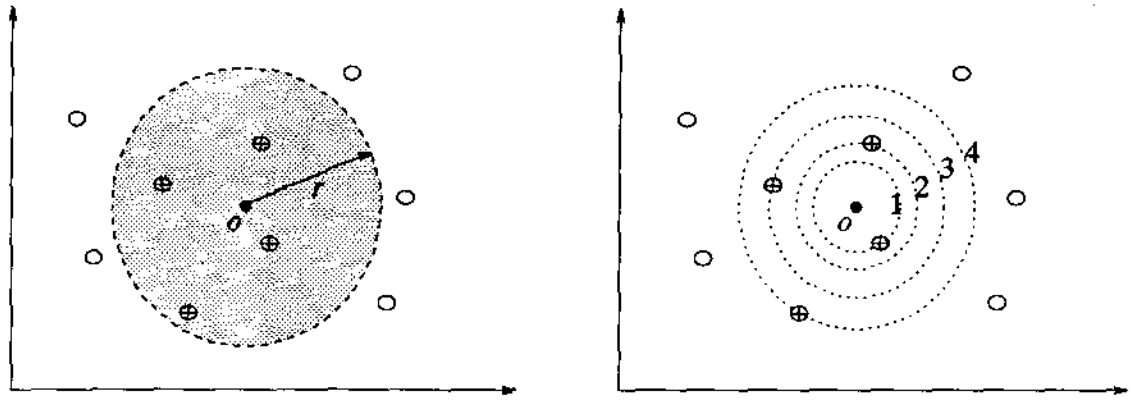
Este problema está intimamente relacionado com o problema de determinar os K vizinhos mais próximos[1] de um ponto de consulta $o \in \Gamma$ [PS85, Ede87]. Ambos dizem respeito à proximidade dos pontos de P em relação ao ponto de consulta o. No problema de busca em subespaço circular há uma distância máxima r (o raio do disco) para os pontos satisfazendo à consulta (figura 4.1-a), ao passo que no problema dos K vizinhos mais próximos limita-se o número de pontos a serem retornados (figura 4.1-b).

Uma vez que busca em subespaço circular e K vizinhos mais próximos envolvem proximidade, pode-se utilizar diagramas de Voronoi como estruturas básicas para determinar as decomposições do espaço Γ que vão orientar a concepção de estruturas pré-processadas para auxiliar as buscas.

Os trabalhos encontrados na literatura a respeito do problema de busca em subespaço circular, até o presente momento, são voltados principalmente para o problema restrito a duas dimensões. Assim, também as estruturas consideradas neste capítulo são adequadas especificamente para busca em discos no plano. Aplicando o paradigma de linearização (seção 2.8), pode-se utilizar algumas soluções propostas para o problema de busca em semi-espaços (capítulo 6), para obter soluções para busca em subespaço circular em várias dimensões.

As soluções descritas neste capítulo baseiam-se em diagramas de Voronoi de K-ésima ordem (seção 4.1.1) e diagramas de Voronoi de K-ésimo vizinho mais próximo (seção 4.1.2), os quais permitem particionar o espaço Γ em classes de equivalência, sugerindo o paradigma do lugar geométrico (seção 2.1) para solucionar o problema dos K vizinhos mais próximos com K fixo. Aplicando-se o paradigma de filtragens sucessivas (seção 2.2) é possível aperfeiçoar a técnica,

[1] Neste capítulo, utilizaremos K (maiúsculo) para denotar o *grau de vizinhança* de um ponto $p \in P$ em relação a um ponto de consulta $q \in \Gamma$ (veja definição de diagrama de K-ésimo vizinho mais próximo na seção 4.1.2) ou o *escopo* (ordem) de um diagrama de Voronoi (veja definição de diagrama de Voronoi de K-ésima ordem na seção 4.1.1). O k (minúsculo) continua denotando o tamanho da saída.



(a) (b)

Figura 4.1: O problema de busca em subespaço circular (a) e sua relação com o problema de determinar os K vizinhos mais próximos (b).

levando a soluções para o problema dos K vizinhos mais próximos com K variável e para busca em subespaço circular.

As soluções utilizando diagramas de Voronoi são adequadas, isto é, garantem o tempo de busca de pior caso limitado, somente para problemas de enumeração, uma vez que utilizam o paradigma de filtragens sucessivas.

A seguir, são descritas algumas destas soluções, partindo da mais simples para a mais sofisticada.

4.1 Algumas soluções

4.1.1 Solução direta por diagramas de Voronoi

Preliminares

A idéia de de utilizar diagramas de Voronoi para solucionar consultas de vizinhança se deve a Bentley e Maurer [BM79]. Uma possibilidade é utilizar o diagrama de Voronoi de K-ésima ordem (*Kth order Voronoi diagram*) [PS85]:

Diagrama de Voronoi de K-ésima ordem:

Seja $P \subset \Gamma$ um conjunto de N pontos no espaço multidimensional Γ.

Seja $\mathcal{P}$ um subconjunto não vazio de P.

Seja $vor(\mathcal{P})$ a região do espaço Γ tal que um ponto arbitrário no interior da mesma é mais próximo de qualquer ponto de $\mathcal{P}$ do que de qualquer outro ponto em $P - \mathcal{P}$.

O diagrama de Voronoi de K-ésima ordem ($0 < K < N$) para P, denotado por $Vor_K(P)$ é a partição do espaço Γ na coleção de regiões $vor(\mathcal{P})$, onde $\mathcal{P} \in 2^P$ e $|\mathcal{P}| = K$:

$$Vor_K(P) = \{vor(\mathcal{P}) \mid \mathcal{P} \in 2^P, |\mathcal{P}| = K\} \tag{4.1}$$

A ordem K do diagrama $Vor_K(P)$ é também denominada escopo do diagrama.

Dois pontos quaisquer contidos em uma mesma região $vor(\mathcal{P})$ de $Vor_K(P)$ possuem o mesmo conjunto de K vizinhos mais próximos $\mathcal{P}$. A figura 4.2 ilustra um diagrama de Voronoi de ordem 3 para o conjunto de cinco pontos $P = \{p_1, p_2, p_3, p_4, p_5\}$ no plano euclideano. Cada região $vor(p_i, p_j, p_k)$ de $Vor_3(P)$ tem como conjunto de 3 pontos mais próximos $\{p_i, p_j, p_k\}$, com $1 \leq i, j, k \leq 5$.

O leitor pode verificar que $Vor_K(P)$, para um conjunto de pontos P no plano euclideano[2] é uma partição do plano (uma subdivisão planar) cujas células são polígonos convexos possivelmente ilimitados. Para alguns subconjuntos $\mathcal{P} \subset P$ a região $vor(\mathcal{P})$ pode ser vazia. É demonstrado inclusive que somente $O(N^3)$ das $\sum_{K=1}^{N-1} \binom{N}{K} = 2^N - 2$ possibilidades para $\mathcal{P} \subset P$ com $0 < |\mathcal{P}| < N$ correspondem a alguma região $vor(\mathcal{P})$ não vazia [Lee82].

Duas propriedades dos diagramas de Voronoi de K-ésima ordem em $\mathbf{R}^2$ relevantes na solução de consultas de vizinhança são:

1. $Vor_K(P)$ pode ser visto como um grafo planar, cujos vértices têm grau maior ou igual a 3 e com $O(K(N-K))$ vértices, arestas e faces.

2. Uma vez que $Vor_K(P)$ constitui uma subdivisão planar, pode-se realizar localização de pontos neste diagrama em tempo $O(\log N)$, utilizando uma estrutura de dados pré-processada de tamanho linear no número de células (vértices, arestas e faces) do diagrama, $O(K(N-K))$ [LT80, Kir83].

[2]Daqui para frente, considere $\Gamma = \mathbf{R}^2$, a não ser que seja especificado de forma diferente.

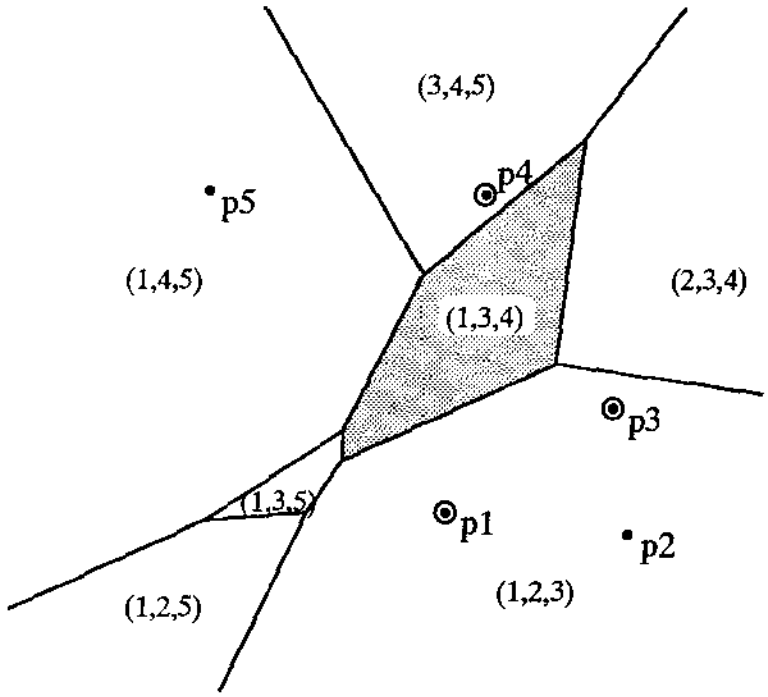


Figura 4.2: Um diagrama de Voronoi de ordem 3, onde a região hachurada tem como 3 vizinhos mais próximos os pontos realçados com círculos.

Descrição

A definição de diagrama de Voronoi de K-ésima ordem sugere o paradigma do lugar geométrico (seção 2.1) para solucionar o problema de K vizinhos mais próximos, pois efetua a divisão do espaço de consultas em classes de equivalência (as faces onde o conjunto de K vizinhos mais próximos é constante). Para determinar os K vizinhos mais próximos de um ponto de consulta $q \in \Gamma$ basta efetuar uma localização deste ponto em $Vor_K(P)$. O inconveniente desta abordagem é que só se consegue efetuar consultas com K fixo.

A solução aqui apresentada baseia-se na descrição encontrada em [CCPY86] e se aplica tanto ao problema dos k vizinhos mais próximos com k variável, quanto a busca em subespaço circular e é uma aplicação da idéia básica de localização em diagramas de Voronoi de K-ésima ordem em conjunção com o paradigma de filtragens sucessivas (seção 2.2). Ela utiliza uma seqüência de diagramas de Voronoi de várias ordens:

$$\{Vor_{2^i}(P)\,;\, i = 0, 1, \ldots, \lceil \log N \rceil - 1\} \tag{4.2}$$

A cada face $vor(\mathcal{P})$ desses diagramas é conectada a lista de vizinhos mais próximos relativa a $\mathcal{P}$, denominada lista de vizinhança (*neighbor list*). Dada uma busca em um subespaço circular $\sigma(o, r)$, onde o é o centro e r o raio do disco de busca, o ponto o é sucessivamente localizado em

cada um dos diagramas $Vor_{2^i}(P)$, $i = 0, 1, 2, \ldots$ e a lista de vizinhança correspondente à face contendo o ponto o de cada diagrama é examinada.

Este processo de localização nos diagramas e verificação das listas de vizinhança termina quando for encontrado um ponto p numa lista de vizinhança tal que $\mathrm{dist}(o,p) > r$. Isso significa que a lista de vizinhança em questão contém os pontos que satisfazem à consulta e possivelmente mais alguns. O tamanho da lista de vizinhança em questão é no máximo o dobro da quantidade k de pontos satisfazendo à consulta e desta forma ela pode ser percorrida em tempo $O(k)$ para filtrar os elementos desejados.

Se nenhuma das $2^{\lceil \log N \rceil}$ listas de vizinhança que podem ser analisadas contiver um tal elemento p, então pelo menos a metade dos pontos de P estão contidos em $\sigma(o,r)$ de modo que se pode efetuar uma busca seqüencial em P, sem afetar o tempo de busca assintótico da solução.

A memória ocupada pela estrutura pré-processada necessária para efetuar as buscas em modo de enumeração é a soma dos espaços ocupados pela seqüência de diagramas de Voronoi, juntamente com as listas de vizinhança de suas faces:

$$m = \sum_{i=0}^{\lceil \log N \rceil - 1} 2^{2i} \cdot (N - 2^i) < N \cdot \left(\frac{(2^2)^{\lceil \log N \rceil} - 1}{2^2 - 1} \right) < N \cdot \frac{N^2 + 3}{2^2 - 1} = O(N^3) \qquad (4.3)$$

O tempo de consulta depende do tamanho total $t(k)$ das listas de vizinhança examinadas, onde k denota o tamanho da saída:

$$t(k) = 1 + 2 + 4 + \ldots + 2^{\lfloor \log k \rfloor + 1} \leq 4k = O(k) \qquad (4.4)$$

Se $k < \log N \cdot \log\log N$ o tempo de busca (localizações de pontos nos $O(\log k)$ diagramas de Voronoi) domina o tamanho do conjunto recuperado, ao passo que a enumeração da saída é o tempo dominante para $k \geq \log N \cdot \log\log N$. Assim, o tempo de consulta é $O(\log N \cdot \log\log N + k)$.

O tempo de pré-processamento é o necessário para construir os diagramas de Voronoi de ordem mais alta com as listas de vizinhança das faces. Pode-se utilizar o algoritmo de Lee [Lee82] para construir cada diagrama $Vor_K(P)$ em tempo $O(K^2 N \log N)$. Este algoritmo retorna juntamente com o diagrama os conjuntos de K vizinhos das faces, sem custo adicional. Mais recentemente, Edelsbrunner e Seidel [ES86] propuseram um método ótimo para a construção de um diagrama de Voronoi de ordem K qualquer, isto é, em tempo $O(N^3)$ para qualquer ordem K, mas portanto menos eficiente que o método de Lee para $K \in o(N/\sqrt{\log N})$.

As medidas de complexidade da solução para busca em subespaço circular descrita acima são sintetizadas a seguir.

Complexidades

C: $O(\log N \cdot \log\log N + k)$
M:$O(N^3)$
P: $O(N^3)$

Note que o uso de memória tem um fator $O(N)$ a mais que os limites inferiores estabelecidos (seção 4.2), o que justifica a pesquisa de algoritmos mais eficientes. As duas soluções apresentadas a seguir mostram como se pode efetuar compactação das listas de vizinhança e utilizar diagramas com escopos limitados para diminuir esta demanda de memória.

4.1.2 Solução de Chazelle, Cole, Preparata e Yap

Preliminares

Esta solução [CCPY86] é um aperfeiçoamento da solução anterior (seção 4.1.1), com ganhos tanto no uso de memória quanto no tempo de consulta. Ela utiliza o conceito de **diagrama de K-ésimos vizinhos mais próximos** (*Kth nearest neighbors diagram*).

Conceitos relacionados com vizinhança:

Seja $P \subset \Gamma$ um conjunto de N pontos no espaço multidimensional Γ.

O grau de vizinhança de um ponto $p \in P$ em relação a um ponto de consulta $q \in \Gamma$ é a posição de $\text{dist}(p, q)$ na seqüência ascendente $\{\text{dist}(p_i, q) \mid p_i \in P, 1 \leq i \leq N\}$.

O K-ésimo vizinho de q em P é o ponto $p \in P$ cujo grau de vizinhança em relação a q é igual a K.

O diagrama de K-ésimos vizinhos mais próximos para P, denotado por $Viz_K(P)$ é a partição do espaço Γ em um conjunto de regiões, tal que quaisquer dois pontos no interior de uma mesma região têm os seus conjuntos de K-ésimos vizinhos idênticos.

O diagrama de K-ésimos vizinhos mais próximos $Viz_K(P)$ para um conjunto de pontos P no plano euclideano, quando visto como um grafo tem a seguinte relação com o diagrama de Voronoi de K-ésima ordem $Vor_K(P)$ (seção 4.1.1)[3]:

$$Viz_K(P) = Vor_{K-1}(P) \cup Vor_K(P) \tag{4.5}$$

[3] Na equação que se segue, a operação de união diz respeito aos conjuntos de vértices e de arestas do grafo.

A figura 4.3 ilustra a obtenção do diagrama de terceiros vizinhos mais próximos $Viz_3(P)$ para o conjunto de pontos $P = \{p_1, p_2, p_3, p_4, p_5\}$ no plano euclideano, através da união dos diagramas de Voronoi de ordem mais alta $Vor_2(P)$ e $Vor_3(P)$.

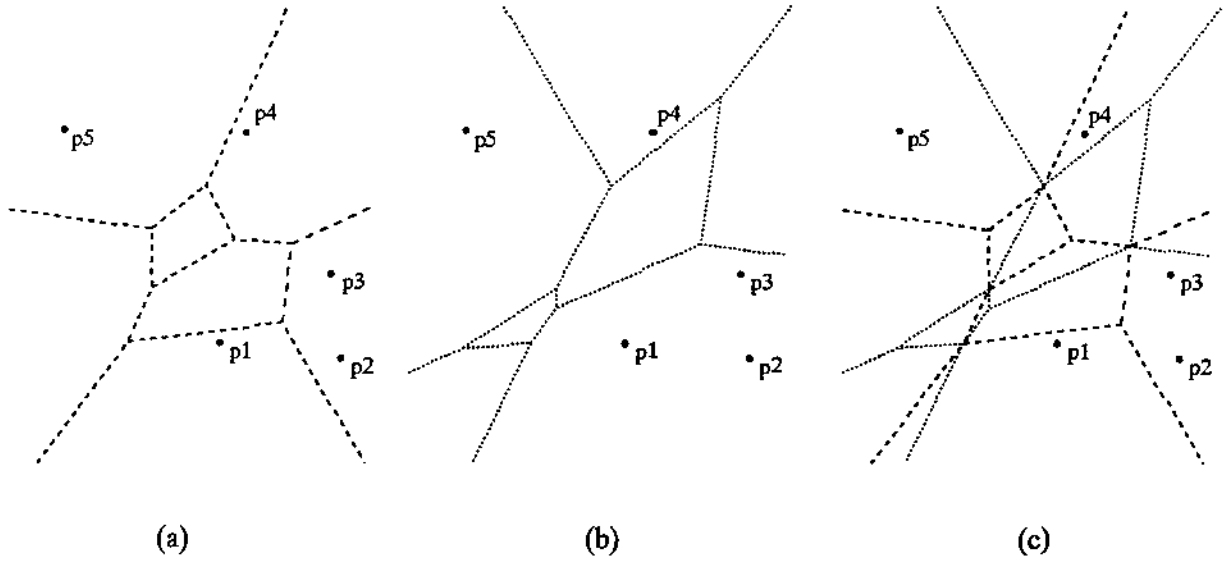


Figura 4.3: $Viz_3(P)$ como a união de $Vor_2(P)$ e $Vor_3(P)$.

A partir da relação dada pela equação 4.5, pode-se derivar para diagramas de K-ésimos vizinhos mais próximos em $\mathbf{R}^2$ relações análogas às destacadas na seção 4.1.1 para diagramas de Voronoi de K-ésima ordem, considerando os pontos de P em posição geral e que nenhum subconjunto de P contenha três pontos co-circulares:

1. $Viz_K(P)$ pode ser visto como um grafo planar, cujos vértices têm grau 3 ou 6 e com $O(K(N-K))$ vértices, arestas e faces.

2. Pode-se efetuar localização de pontos em $Viz_K(P)$ em tempo $O(\log N)$, utilizando uma estrutura de dados pré-processada, armazenada em espaço $O(K(N-K))$ [LT80, Kir83].

Seja $viz(p_i)$ a região do diagrama de K-ésimos vizinhos mais próximos $Viz_K(P)$ que tem como terceiro vizinho mais próximo o ponto p_i, com $1 \leq i \leq K$. Considere os pontos de $P \subset \mathbf{R}^2$ em posição geral e tais que não haja três pontos co-circulares em P. O leitor pode verificar que cada região $viz(p_i)$ é constituída de uma cadeia de faces de $Viz_K(P)$, tal que duas faces consecutivas de $viz(p_i)$ compartilham um vértice de grau 6 de $Viz_K(P)$. A figura 4.4 ilustra o diagrama de Voronoi de terceiros vizinhos mais próximos $Viz_3(P)$ para o conjunto de pontos $P = \{p_1, p_2, p_3, p_4, p_5\}$ no plano euclideano, onde a região hachurada tem como terceiro vizinho mais próximo o ponto p_1.

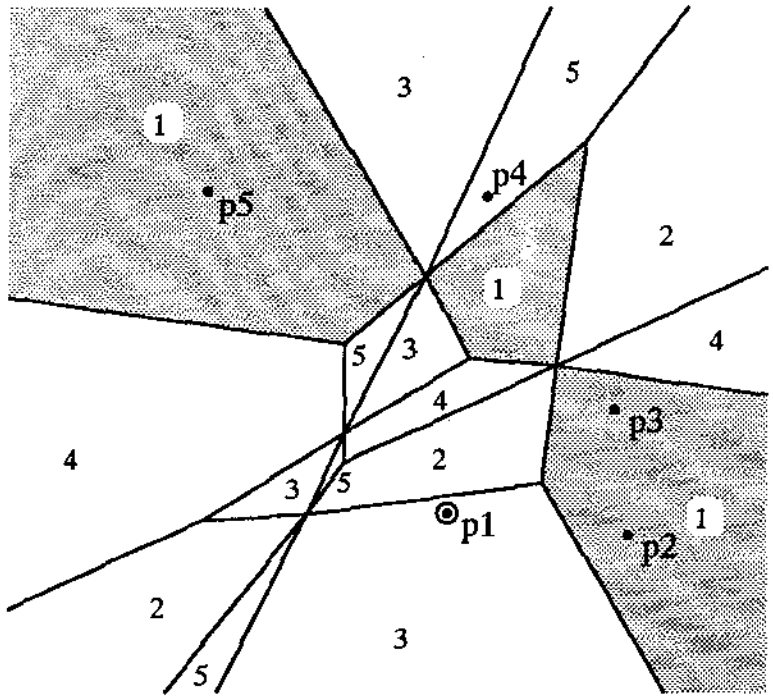


Figura 4.4: O diagrama $Viz_3(P)$, onde é realçada a região que tem como terceiro vizinho mais próximo o ponto p_1.

Uma estrutura compactada para consultas de K vizinhos mais próximos

Seja uma estrutura de dados pré-processada representando o diagrama de K-ésimos vizinhos mais próximos $Viz_K(P)$, onde cada face $\varphi \in Viz_K(P)$ aponta para a lista dos K vizinhos mais próximos correspondente. Então, para determinar os K vizinhos mais próximos de um ponto de consulta o (com K fixo) basta localizar o ponto o nesta estrutura. O inconveniente desta abordagem é que a estrutura pré-processada ocupa memória $O(K^2(N-K))$.

Chazelle, Cole, Preparata e Yap observaram que faces adjacentes em $Viz_K(P)$ têm suas listas de vizinhança diferindo em no máximo 1 elemento. Tal propriedade permite uma compactação das listas de vizinhança utilizando o paradigma de filtragens sucessivas (seção 2.2).

As faces de $Viz_K(P)$ são agrupadas em um sistema de grupos de faces $\Upsilon = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_\tau\}$, onde cada grupo $\mathcal{F}_i$ $(1 \leq i \leq \tau)$ é uma coleção conexa de faces[4]. Seja $\mathcal{P}(\varphi)$ a lista de vizinhança correspondente a uma face φ de $Viz_K(P)$. As listas de vizinhança das faces de cada grupo $\mathcal{F}_i$ são unidas em um conjunto de vizinhos $\mathcal{P}(\mathcal{F}_i) = \bigcup_{\varphi \in \mathcal{F}_i} \mathcal{P}(\varphi)$. Desta forma, consegue-se diminuir o número de replicações de pontos a armazenar. O método de agrupamento das faces garante que cada conjunto de vizinhos $\mathcal{P}(\mathcal{F}_i)$ contenha $O(K)$ pontos, o que permite a filtragem dos K

[4] Na verdade, uma face $\varphi \in Viz_K(P)$ pode ser associada a mais de um grupo de Υ, mas é garantido que o número de replicações de faces entre os grupos é linear no número de faces. A forma como é efetuada a divisão em grupos pode ser encontrada em detalhes em [CCPY86].

pontos correspondentes aos vizinhos mais próximos de qualquer face φ em tempo $O(K)$, uma vez localizado o grupo de faces correspondente e conhecendo-se o K-ésimo vizinho. Esta filtragem é efetuada comparando cada ponto de $\mathcal{P}(\mathcal{F}_i)$ com o K-ésimo vizinho do ponto de consulta. A figura 4.5 ilustra os grupamentos de faces para o diagrama de terceiros vizinhos mais próximos $Viz_3(P)$, relativo ao conjunto de pontos $P = \{p_1, p_2, p_3, p_4, p_5\}$.

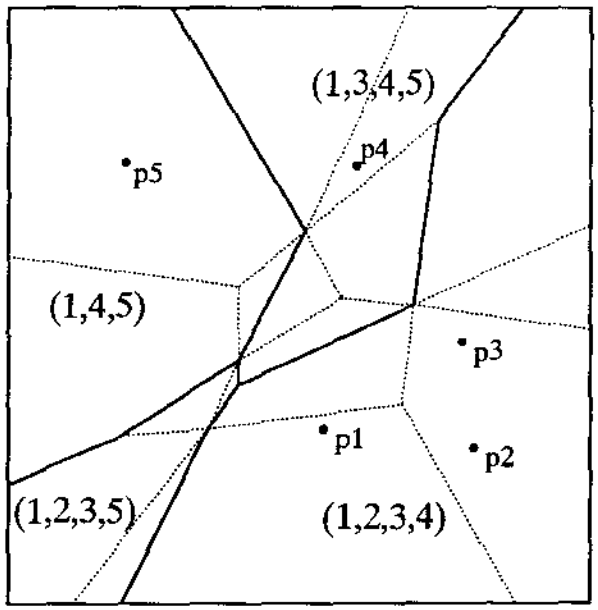


Figura 4.5: Grupamentos de faces para $Viz_3(P)$ com os conjuntos de vizinhos.

A memória total ocupada pelos conjuntos de vizinhos de ordem K relativos aos grupos de faces de $Viz_K(P)$ é[5]:

$$m(K) = \sum_{\mathcal{F}_i \in \Upsilon} |\mathcal{P}(\mathcal{F}_i)| = O(K(N-K)) \tag{4.6}$$

Pode-se então construir uma estrutura de dados $B_K(P)$ ocupando memória $O(K(N-K))$, com a qual se pode responder as consultas de K vizinhos mais próximos (com K fixo) em tempo $O(\log N + K)$. Esta estrutura consiste essencialmente do diagrama $Viz_K(P)$ pré-processado para localização de pontos eficiente de acordo com o método de Kirkpatrick [Kir83], de modo que cada face φ de $Viz_K(P)$ seja conectada à lista $\mathcal{P}(\mathcal{F}_i)$ tal que[6] $\varphi \in \mathcal{F}_i$.

A estrutura $B_K(P)$ é uma árvore de localização, onde cada nó é associado a uma região composta de um ou mais grupos de faces de Υ. Ao nó raiz é associado todo o plano e aos nós descendentes vão sendo associadas regiões menores, contidas nas regiões associadas aos pais, até atingir regiões cujos conjuntos de vizinhos de ordem K (união das listas de vizinhança das faces

[5] A demonstração pode ser encontrada em [CCPY86].

[6] Podem haver vários grupos contendo a face φ. Quando isso ocorre, escolhe-se arbitrariamente algum grupo.

de $Viz_K(P)$ contidas em cada região) contenham $O(K)$ faces, para alguma constante multiplicativa conveniente. Nos nós folha são conectados os conjuntos de vizinhos correspondentes. Os K vizinhos mais próximos podem ser separados em tempo $O(K)$ comparando cada elemento do conjunto correspondente com o K-ésimo vizinho, o qual é obtido por meio de uma localização em $Viz_K(P)$ (que no caso é apenas um refinamento da localização efetuada em $\mathcal{B}_K(P)$). Uma árvore de localização para $Vor_3(P)$ é ilustrada na figura 4.6.

Uma estrutura para busca em subespaços circulares

A estrutura para solucionar busca em subespaços circulares tem como componente primária uma árvore binária $T(P)$, cujas folhas estão em correspondência um a um com os pontos da base de dados P (suponha uma atribuição arbitrária[7]).

A cada nó não folha v de $T(P)$ são associados (implicitamente) um conjunto $\mathcal{P}(v)$, composto dos pontos associados às folhas da sub-árvore de $T(P)$ com raiz em v e um escopo $K(v) > 0$. A cada um desses nós são conectadas (explicitamente) uma estrutura representando o diagrama $Viz_{K(v)}(\mathcal{P}(v))$ e uma coleção de estruturas para busca dos vizinhos mais próximos com escopo fixo no conjunto de pontos $\mathcal{P}(v)$:

$$B_{K(v)}(\mathcal{P}(v)) = \{\mathcal{B}_{2^i\lfloor \log N\rfloor}(\mathcal{P}(v)) ;\ 0 \leq i \leq \lceil \log \frac{K(v)}{\lfloor \log N\rfloor}\rceil\} \tag{4.7}$$

onde cada estrutura $B_{2^i\lfloor \log N\rfloor}(\mathcal{P}(v))$ tem a forma descrita anteriormente[8].

O algoritmo de busca opera recursivamente e uma consulta $\sigma(o,r)$ vai ocasionar o percurso de um caminho $\rho(\sigma(o,r))$ em $T(P)$, a partir da raiz. Para efetuar uma busca em uma estrutura $T(P)$ pelos pontos contidos em um círculo de centro o e raio r, basta chamar Busca($raiz(T(P))$,o,r), utilizando o algoritmo abaixo.

[7] Chazelle, Cole, Preparata e Yap [CCPY86] apresentam um algoritmo probabilístico para efetuar o pré-processamento, atribuindo os pontos de P às folhas de $T(P)$ segundo alguns critérios que vão possibilitar um uso de memória mais econômico. Veja as medidas de complexidade obtidas no final da seção 4.1.2.

[8] Para manter a consistência, define-se $B_K(\mathcal{P}) = \mathcal{P}$ quando $K \geq |\mathcal{P}|$.

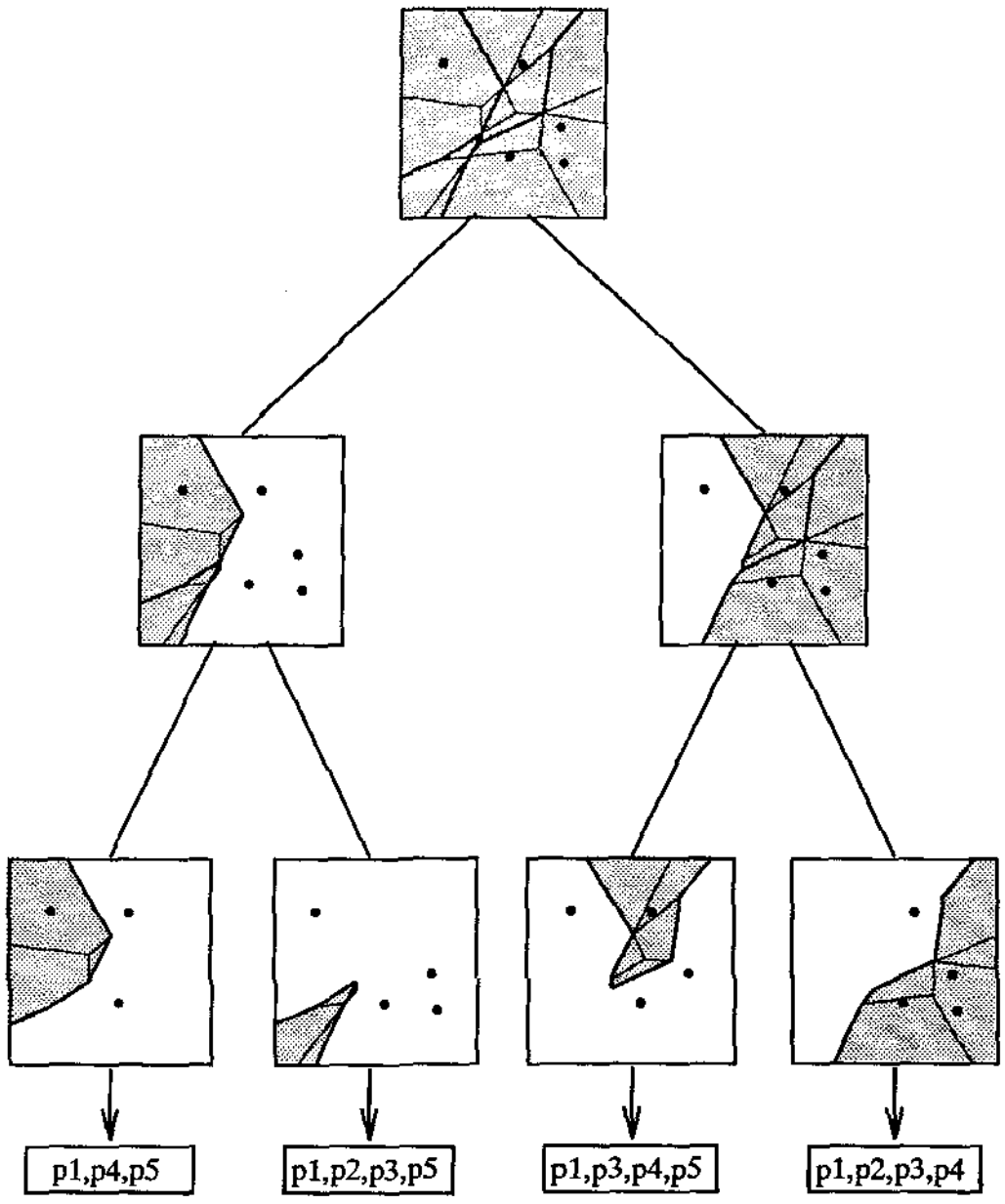


Figura 4.6: A árvore de localização para os grupos de faces de $Viz_3(P)$.

```
function Busca(v,o,r)
begin
    Calcule p_{K(v)}, o K(v)-ésimo vizinho de o em P(v);
    (Isso é feito por meio de uma localização de pontos em Viz_{K(v)}P(v))
    if dist(p_{K(v)}, o) > r then
        Recupere o conjunto dos O(K(v)) pontos de P(v) mais próximos de o;
        Uma filtragem deste produz os k̄ pontos de P(v) contidos em σ(o, r)
    else
        if v é folha then
            Retorne o conjunto de pontos armazenado em v
        else
            Busca(v.esq,o,r);
            Busca(v.dir,o,r);
end;
```

A atribuição de escopos aos nós de $T(P)$ é feita de forma a impedir que nós associados a subconjuntos $\mathcal{P}(v)$ de tamanho proporcional a N (nós próximos à raiz) tenham valores de escopo $K(v)$ maiores que $O(\log^c |\mathcal{P}(v)|)$, onde $c > 0$ é uma constante. Desta forma, se elimina a necessidade de diagramas de Voronoi com escopos excessivamente altos, os quais implicam em utilização de muita memória: lembre-se que o número de faces dos diagramas $Vor_K(\mathcal{P})$ e $Viz_K(\mathcal{P})$ é $O(K(|\mathcal{P}| - K))$ e que, com o recurso de agrupamento de faces consegue-se armazenar um diagrama $Viz_K(P)$ *com as listas de vizinhança* em memória de tamanho proporcional ao número de faces deste diagrama (estrutura $\mathcal{B}_K(\mathcal{P})$).

A aplicação desses dois recursos (agrupamento de faces e limitação de escopos) proporciona economia de memória considerável e melhoria do tempo de consulta, em relação à solução da seção 4.1.1. As medidas de complexidade de pior caso da solução de Chazelle, Cole, Preparata e Yap, obtidas mediante uma atribuição adequada de escopos $K(v)$ aos nós de $T(P)$ são apresentadas a seguir[9].

Complexidades

C: $O(\log N + k)$
M: $O(N \cdot (\log N \cdot \log\log N)^2)$
P: $O(N \cdot \log^5 N \cdot (\log\log N)^2)$

Chazelle, Cole, Preparata e Yap propõem a utilização do algoritmo de Lee [Lee82] para a construção dos diagramas de K-ésimos vizinhos mais próximos. Este método permite construir

[9] Detalhes em [CCPY86].

os diagramas $Vor_K(P)$ e $Viz_K(P)$ em tempo $O(K^2 N \log N)$ e retornar juntamente com o diagrama os conjuntos de K vizinhos mais próximos das faces, sem custo adicional, o que resulta no tempo de pré-processamento indicado acima.

Com o emprego de um algoritmo probabilístico para efetuar o pré-processamento, consegue-se uma utilização de memória de ordem $O(N \cdot \log^2 N)$, mantendo a mesma complexidade de tempo de consulta.

4.1.3 Solução de Aggarwal, Hansen e Leighton

Esta solução [AHL90] envolve separadores planares [LT79], filtragens sucessivas (seção 2.2) e um método probabilístico para compactação de diagramas de Voronoi de K-ésima ordem. O método de compactação permite reduzir a quantidade de memória necessária para armazenar $Vor_K(P)$ com as listas de vizinhança, de $O(K(N-K))$ para $O(N)$ (a descrição do método é feita no decorrer desta seção). Com essas técnicas eles obtêm soluções eficientes não apenas para o problema de busca em subespaço circular, mas também para o problema dos k vizinhos mais próximos de um ponto no plano e busca em semi-espaços em 3 dimensões (veja seção 6.3).

Método de compactação

Seja $\Phi(P)$ uma partição do plano dada pelo diagrama de Voronoi de K-ésima ordem $Vor_K(P)$ triangulado. Assuma os pontos de P em posição geral e tais que nenhum subconjunto de P tenha três pontos co-circulares. A partição $\Phi(P)$ pode ser vista como um grafo planar com $O(KN)$ vértices, arestas e faces. Seja $\mathcal{P}(\varphi)$ o conjunto dos K vizinhos mais próximos (lista de vizinhança) correspondente a uma face $\varphi \in \Phi(P)$. Aggarwal, Hansen e Leighton, da mesma forma que Chazelle, Cole, Preparata e Yap (seção 4.1.2) partiram do fato que listas de vizinhança relativas a faces adjacentes em $\Phi(P)$ diferem em no máximo um elemento, para efetuar a compactação das listas de vizinhança em conjuntos correspondentes a $\sim K^6$ faces de $\Phi(P)$, cuja união é uma região conexa em Γ.

Para separar as faces de $\Phi(P)$ em grupos utiliza-se o teorema do separador planar de Lipton e Tarjan [LT79], o qual se aplica ao dual de $\Phi(P)$, denotado por $\mathcal{D}(\Phi(P))$. O teorema do separador planar permite dividir $\mathcal{D}(\Phi(P))$ em duas partes, cada qual com aproximadamente a metade do número de vértices de $\mathcal{D}(\Phi(P))$, mediante a remoção de algumas arestas deste dual, correspondentes a adjacências de faces em $\Phi(P)$. Portanto, pode-se dividir as faces de $\Phi(P)$ em dois grupos, cada qual com aproximadamente a metade das faces de $\Phi(P)$. Aplicando-se o teorema recursivamente pode-se separar as faces de $\Phi(P)$ em grupos conexos, tais que a união das listas de vizinhança das faces componentes de cada grupo tenha o tamanho desejado.

O sistema de grupos de faces $\Upsilon = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_\tau\}$ em que são divididas as faces de $\Phi(P)$ contém τ grupos conexos de faces. Seja $\mathcal{P}(\mathcal{F}_i) = \bigcup_{\varphi \in \mathcal{F}_i} \mathcal{P}(\varphi)$, onde $\mathcal{P}(\varphi)$ é o conjunto de K vizinhos mais próximos relativo à face φ. Uma vez que o número total de arestas de $\Phi(P)$

removidas no processo de partição é $O(N/K^2)$, consegue-se provar que[10]:

$$m(K) = \sum_{i=1}^{\tau} \mathcal{P}(\mathcal{F}_i) = O(N) \tag{4.8}$$

A árvore de localização

Pode-se então construir uma **árvore de localização** $T_K(P)$ ocupando espaço linear, para efetuar localização de pontos no sistema de grupos de faces $\Upsilon = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_\tau\}$ pelo método de Kirkpatrick [Kir83]. A cada folha de $T_K(P)$ é associada uma região $\mathcal{F}_i \in \Upsilon$ e conectado o conjunto de pontos $\mathcal{P}(\mathcal{F}_i)$, armazenado em uma estrutura ocupando espaço linear. A cada nó intermediário v é associada uma região $\mathcal{R}(v)$, obtida pela união de todos os grupos de faces associados a folhas descendentes de v.

Com a árvore de localização $T_K(P)$ pode-se determinar, em tempo logarítmico, o grupo de faces $\mathcal{F}_i$ contendo o ponto de consulta o e portanto, o conjunto $\mathcal{P}(\mathcal{F}_i)$ com $O(K^7)$ pontos (correspondentes às $O(K^6)$ faces de $\mathcal{F}_i$) contendo os $O(K)$ vizinhos mais próximos de o. Assim, tem-se uma estrutura de dados de tamanho linear no número N de pontos de P, capaz de responder consultas de K vizinhos mais próximos, com K fixo, em tempo $O(\log N + K)$. Considera-se que as estruturas conectadas às folhas de $T_K(P)$ para armazenar os $O(K^7)$ pontos de cada $\mathcal{P}(\mathcal{F}_i)$ ocupam espaço linear, isto é, $O(K^7)$ e permitem separar os K vizinhos mais próximos em tempo $O(K)$. Então, para completar a descrição de $T_K(P)$, resta apresentar a forma dessas estruturas conectadas às folhas.

Estruturas de tamanho linear para busca nos conjuntos de pontos $\mathcal{P}(\mathcal{F}_i)$

Os pontos de $\mathcal{P}(\mathcal{F}_i)$ são distribuídos aleatória e uniformemente entre $s = \frac{K}{\log^2 K}$ subconjuntos $C_i^1, C_i^2, \ldots, C_i^s$. A intenção é buscar em cada um desses subconjuntos os $O(\log^2 K)$ vizinhos mais próximos de o. Para tal é necessário que, para toda consulta de K vizinhos mais próximos possível, o conjunto dos K vizinhos mais próximos esteja uniformemente distribuído entre os subconjuntos $C_i^1, C_i^2, \ldots, C_i^s$. Diz-se que uma atribuição de pontos é **satisfatória** se, para qualquer consulta de K vizinhos mais próximos, cada C_i^j contém $\log^2 K + O(\log^{3/2} K)$ pontos da resposta.

Ora, o conjunto de todas as respostas de consultas por K vizinhos mais próximos é dado pelas listas de vizinhança do diagrama de Voronoi de K-ésima ordem $Vor_K(\mathcal{P}(\mathcal{F}_i))$. Então, para testar se uma atribuição aleatória e uniforme dos pontos de $\mathcal{P}(\mathcal{F}_i)$ aos subconjuntos $C_i^1, C_i^2, \ldots, C_i^s$ é satisfatória, basta construir $Vor_K(\mathcal{P}(\mathcal{F}_i))$ e para cada face do mesmo verificar a distribuição dos pontos da respectiva lista de vizinhança entre os s subconjuntos.

Aggarwal, Hansen e Leighton demonstram que uma atribuição aleatória e uniforme dos pontos de $\mathcal{P}(\mathcal{F}_i)$ aos subconjuntos $C_i^1, C_i^2, \ldots, C_i^s$ é satisfatória com probabilidade maior ou igual

[10] Detalhes em [AHL90].

a $1 - 1/K$. Assim, são esperadas em média duas atribuições dos pontos de $\mathcal{P}(\mathcal{F}_i)$ para conseguir uma atribuição satisfatória. Disso resulta um **algoritmo Las Vegas**[11] para efetuar a atribuição dos pontos de $\mathcal{P}(\mathcal{F}_i)$ aos subconjuntos, com tempo de execução esperado $O(K^2N + N \log N)$ (que é um limite superior do tempo de construção do diagrama $Vor_K(\mathcal{P}(\mathcal{F}_i))$ [Lee82, AGSS87]).

Para possibilitar a recuperação dos pontos dos subconjuntos é construída recursivamente uma árvore de localização para cada C_i^j $(1 \leq i \leq \tau, 1 \leq j \leq s)$. No último nível desta estrutura dada por árvores de localização cujas folhas apontam para outras árvores de localização definidas recursivamente, quando se busca os $O(c)$ pontos mais próximos de o (c é uma constante fixa independente de K), simplesmente se constrói o diagrama de Voronoi de c-ésima ordem, com a estrutura de localização de pontos de Kirkpatrick e com cada face apontando para a lista de vizinhança correspondente. Como c é uma constante, esta estrutura do último nível ocupa memória linear.

Estrutura ótima para consultas de K vizinhos mais próximos

Tem-se uma estrutura de dados $T_K(P)$ para responder consultas de K vizinhos mais próximos com K fixo, que é a árvore de localização onde cada $\mathcal{P}(\mathcal{F}_i)$, associado a uma folha de $T_K(P)$ é dividido em subconjuntos, cada qual armazenado em uma árvore de localização construída recursivamente. As demonstrações de que a estrutura completa da árvore de localização ocupa memória linear e permite responder consultas de K vizinhos mais próximos (com K fixo) em tempo $O(\log N + K)$ podem ser encontradas em detalhes no artigo [AHL90].

As consultas pelos K vizinhos mais próximos de um ponto o contido em um grupo de faces $\mathcal{F}_i$ (que é determinada percorrendo os nós de $T_K(P)$) são respondidas mediante a recuperação de $\log^2 K + 4 \log^{3/2} K$ pontos de cada C_i^j. Obtém-se assim um conjunto de $O(K)$ pontos contendo os K vizinhos mais próximos. É possível determinar o K-ésimo vizinho p_K de o neste conjunto em tempo linear no tamanho da entrada, no caso $O(K)$ [AHU83]. Então, basta percorrer o conjunto recuperado e separar os pontos cujas distâncias a o sejam menores que $\text{dist}(o, p_K)$. A abordagem de produzir um conjunto de pontos candidatos de tamanho proporcional a K e então filtrá-lo para obter a resposta caracteriza o paradigma de filtragens sucessivas.

Estrutura para busca em subespaço circular

Para solucionar o problema de busca em subespaço circular, aplica-se novamente o paradigma de filtragens sucessivas. Utiliza-se uma seqüência de estruturas de árvores de localização, da forma descrita acima, cada qual com tamanho linear em $N = |P|$ e permitindo busca pelos K (fixo) vizinhos mais próximos em tempo $O(\log N + K)$, com $K = \log N, 2 \log N, 4 \log N, \ldots, N$ ($T_N(P)$ consiste simplesmente dos pontos de P armazenados de forma que se possa fazer uma busca seqüencial):

[11] Um algoritmo Las Vegas sempre produz a resposta correta, mas seu tempo de execução só é garantido com uma certa probabilidade, próxima de 1 em alguns casos.

$$T_{circ}(P) = \{T_{\log N}(P), T_{2\log N}(P), T_{4\log N}(P), \ldots, T_N(P)\} \tag{4.9}$$

O tamanho total da seqüência de estruturas $T_{circ}(P)$ é $O(N \log N)$.

Dada uma busca em subespaço circular $\sigma(o, r)$ pode-se recuperar os vizinhos mais próximos de o em cada estrutura da seqüência $T_{circ}(P)$ até uma estrutura $T_j(P)$, cujos vizinhos de o recuperados incluam algum ponto p tal que $dist(o, p) > r$ ou até que $j = N$. Então, basta percorrer a lista dos j pontos vizinhos de o, obtida de $T_j(P)$ e separar aqueles contidos em $\sigma(o, r)$.

O tempo gasto para executar todos os j estágios de busca e filtragem nas listas correspondentes de $T_1(P), \ldots T_j(P)$ é:

$$t = \sum_{i=1}^{j} (2^j + 1) \log N = O(2^j \cdot \log N + \log N) \tag{4.10}$$

o que é proporcional a $O(k + \log N)$ pois $k = O(2^j \log N)$.

As medidas de complexidade obtidas por Aggarwal, Hansen e Leighton são sintetizadas abaixo[12].

Complexidades

C: $O(\log N + k)$	onde k é o tamanho da saída	
M: $O(N \cdot \log N)$		
P: $O(N^3 \cdot \log N)$		(algoritmo determinístico)
$O(N^3)$		(algoritmo probabilístico)
$O(N \cdot \log^2 N \cdot \log \log N)$		(algoritmo Monte Carlo)

O tempo de pré-processamento do algoritmo probabilístico é tempo esperado. O algoritmo Monte Carlo[13] envolve algumas alterações na estrutura descrita acima, porém mantendo as medidas de complexidade do tempo de consulta e do uso de memória.

4.2 Cotas inferiores

Na falta de limites inferiores específicos para o problema de busca em sub-espaço circular, pode-se empregar os limites inferiores obtidos para busca em semi-espaços (veja capítulo 6), pois estes

[12] Detalhes em [AHL90].

[13] Um algoritmo Monte Carlo sempre executa dentro de um tempo limitado, mas só garante a correção da resposta fornecida com uma certa probabilidade, a qual pode ser feita tão próxima de 1 quanto se queira, através da alocação de mais tempo para processamento.

também se aplicam a busca em subespaço circular, já que um semi-espaço pode ser visto como um "caso particular" de hiper-disco: um hiper-disco com centro no infinito na direção normal ao semi-espaço. Esta observação permite derivar limites inferiores não triviais para busca subespaços circulares.

Brönnimann e Chazelle [BC92] estabeleceram limites inferiores para busca em semi-espaços (veja seção 6.2.3) e os estenderam para busca em subespaço circular em qualquer dimensão fixa. O limite inferior estabelecido por eles para o tempo de busca em subespaço circular em *modo de contagem* no modelo aritmético (seção 1.5.1) é:

$$C : \Omega\left(\frac{(N/\log N)^{1-\frac{d-1}{d(d+1)}}}{m^{1/d}}\right) \tag{4.11}$$

onde $d \geq 2$ é o número de dimensões e m é o uso de memória.

Assim, é possível derivar, por exemplo, que uma cota inferior para o tempo de busca em subespaço circular em modo de contagem no plano é:

$$C : \Omega\left(\frac{N^{1/3}}{\log N}\right) \tag{4.12}$$

para uso de memória $O(N)$.

4.3 Notas bibliográficas

Como foi mencionado no início deste capítulo e exemplificado na seção 2.8 (paradigma de linearização), pode-se obter soluções para busca em subespaços circulares a partir de soluções para o problema de busca em semi-espaços (capítulo 6). As soluções deste gênero usualmente permitem efetuar consultas em tempo $O(N^{\alpha})$, para algum $0 < \alpha < 1$, com uso de memória próximo do ótimo (diferindo por fatores $O(N^{\varepsilon})$, com $\varepsilon > 0$) e podem ser encontradas, por exemplo, em [Yao83, YDEP89, Mat93a].

Bentley [Ben90] apresenta uma rotina para efetuar busca em subespaço circular e outras para busca dos k-vizinhos mais próximos no plano euclideano utilizando *KD-Trees* (seção 3.1.1). Ele relata alguns experimentos sobre a eficiência dessas rotinas, nos quais o tempo médio de busca é $O(N^{\alpha})$, para alguns valores específicos de $\alpha < 1$. O método de busca utilizando *KD-Trees* requer memória linear e tempo de pré-processamento $O(N \log N)$. Bentley também fornece alguns argumentos que reforçam os resultados de seus experimentos.

Capítulo 5

Busca em subespaço poligonal

O problema de busca em subespaço poligonal diz respeito ao plano euclideano ($\mathbb{R}^2$) e consiste em determinar os pontos da base de dados P contidos em uma região poligonal de busca σ, conexa e possivelmente ilimitada, como ilustra a figura 5.1-a.

Busca em subespaço poligonal é uma restrição para duas dimensões do problema de busca em poliedros arbitrários. Historicamente, as soluções para busca em semi-espaços e regiões poligonais no plano foram a origem de muitas soluções propostas posteriormente para busca em semi-espaços e simplexos em mais dimensões.

Muitas soluções propostas para este problema baseiam-se em partições do plano em sub-regiões poligonais, tais que um número relativamente pequeno delas intercepte a fronteira da região poligonal de busca σ. Essas partições do plano devem induzir uma partição de P em subconjuntos de tamanhos aproximadamente iguais e os métodos de partição são denominados **esquemas de partição**. O esquema de partição serve para determinar uma estrutura do tipo árvore de partição (seção 2.3), onde os nós são associados às regiões provenientes da partição do plano e portanto, aos pontos de P nelas contidos. A árvore de Willard (seção 5.1.1) e o aperfeiçoamento proposto por Edelsbrunner e Welzl (seção 5.1.2) seguem esta abordagem. Um esquema de partição do plano é ilustrado na figura 5.1-b.

Um outro recurso utilizado para busca em subespaço poligonal (possivelmente em conjunção com árvores de partição) é a decomposição da região poligonal de busca σ em um conjunto de sub-regiões de formatos mais simples $\{\sigma_1, \sigma_2, \ldots, \sigma_s\}$, para as quais seja mais fácil efetuar as buscas. O conjunto dos pontos contidos na região poligonal σ é a união dos conjuntos de pontos contidos em cada uma das sub-regiões $\{\sigma_1, \sigma_2, \ldots, \sigma_s\}$[1]. A solução de Paterson e Yao (seção 5.1.3) explora este recurso em conjunção com dualidade (seção 2.7).

A seguir, estas soluções são descritas em mais detalhes.

[1] Diz-se que o problema de busca em subespaço é passível de decomposição (*decomposable*), pois pode ser decomposto em sub-problemas, de tal forma que a resposta do problema de busca original é a união das respostas dos sub-problemas [Ben79a].

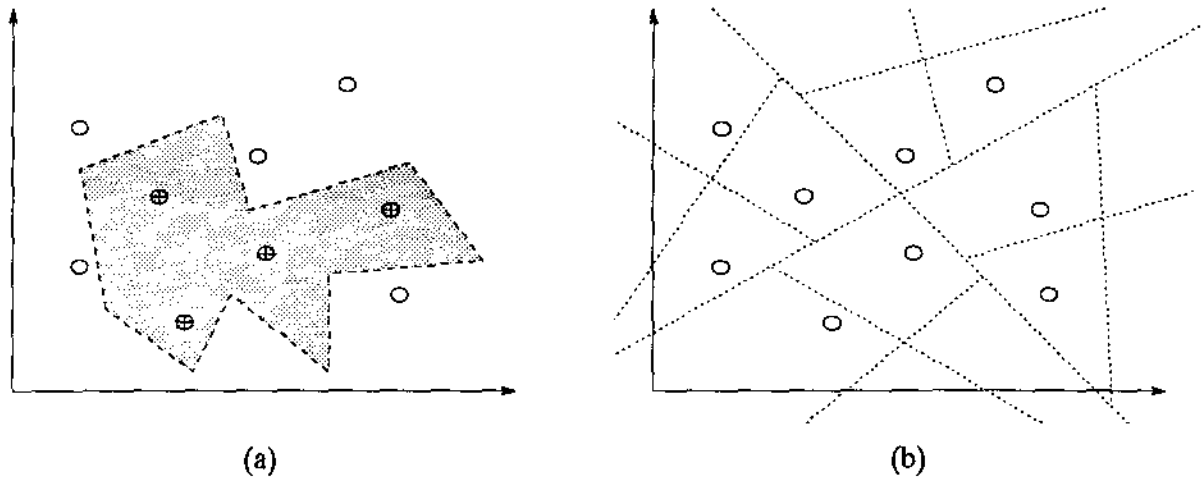


Figura 5.1: Busca em subespaço poligonal (a) e um esquema de partição no plano (b).

5.1 Algumas soluções

5.1.1 Árvore de Willard

A estrutura proposta por Willard [Wil82b] foi por ele denominada árvore poligonal (*polygon tree*) e consiste em uma árvore de partição (seção 2.3) baseada em um tipo particular de partição do plano (subdivisão planar) denominada J-partição (J-*way division*). Tal partição fundamenta-se em uma configuração de linhas divisórias obedecendo certas restrições que, uma vez satisfeitas, permitem estabelecer uma cota superior $O(N^{\log_{2J}(J+1)})$ para o número de faces da subdivisão planar que interceptam a linha de suporte de qualquer segmento de reta componente do contorno do polígono de consulta. Isso possibilita realizar uma busca em uma região poligonal em tempo $O(N^{\log_{2J}(J+1)})$ no pior caso.

A forma de uma J-partição

Um conjunto de J linhas retas $L = \ell_1, \ell_2, \ldots, \ell_J$ é dito ser uma J-**partição** do plano xy se e somente se as três condições seguintes são satisfeitas:

1. ℓ_1 e ℓ_2 são duas linhas retas distintas estendendo-se para infinito em ambas as direções;
2. cada ℓ_i, com $3 \leq i \leq J$ é uma semi-reta cujo ponto inicial está em uma das linhas $\ell_2, \ell_3, \ldots, \ell_{i-1}$ e localizada totalmente à direita (de acordo com a orientação de ℓ_1) de ℓ_{i-1};
3. a linha ℓ_1 intercepta cada uma das outras, $\ell_2, \ell_3, \ldots, \ell_J$.

As linhas de uma J-partição determinam as fronteiras da partição do plano xy em $2J$ regiões. Uma J-partição típica aparece na figura 5.2.

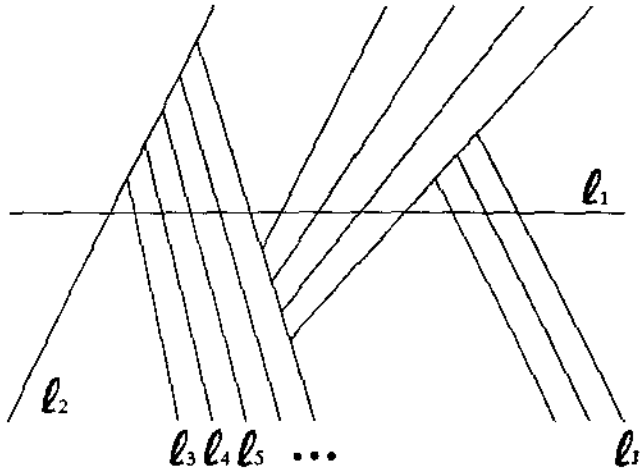


Figura 5.2: Uma J-partição típica.

Árvore de partição baseada em J-partições

Pode-se então construir uma estrutura de dados que Willard denomina árvore poligonal de J-partições quase ideal (*nearly ideal J-way polygon tree*) e que chamaremos simplesmente de **árvore de Willard**. Na árvore de Willard, cada nó v é associado a uma região $\mathcal{R}(v)$ do plano e portanto, ao conjunto de pontos $\mathcal{P}(v) = P \cap \mathcal{R}(v)$ e cada nó v possui uma J-partição associada, construída de acordo com a configuração dos pontos em $\mathcal{P}(v)$ e atuando sobre a região $\mathcal{R}(v)$. O nó raiz é associado a todo o espaço Γ e cada um dos outros nós é associado a uma face da J-partição do nó pai. Este processo de partição é aplicado recursivamente até as folhas, que ficam em correspondência um a um com os pontos da base de dados P. Prova-se que qualquer conjunto P de N pontos pode ser representado por uma árvore de Willard. Uma árvore de Willard para $J = 2$ e a hierarquia de J-partições correspondente são ilustradas na figura 5.3.

Algoritmo de busca

O algoritmo de busca é ilustrado a seguir. Dada uma árvore de Willard $T(P)$, relativa ao conjunto de pontos P, para determinar os pontos de P contidos em uma região poligonal de busca σ, basta efetuar a chamada Busca($raiz(T(P))$,σ).

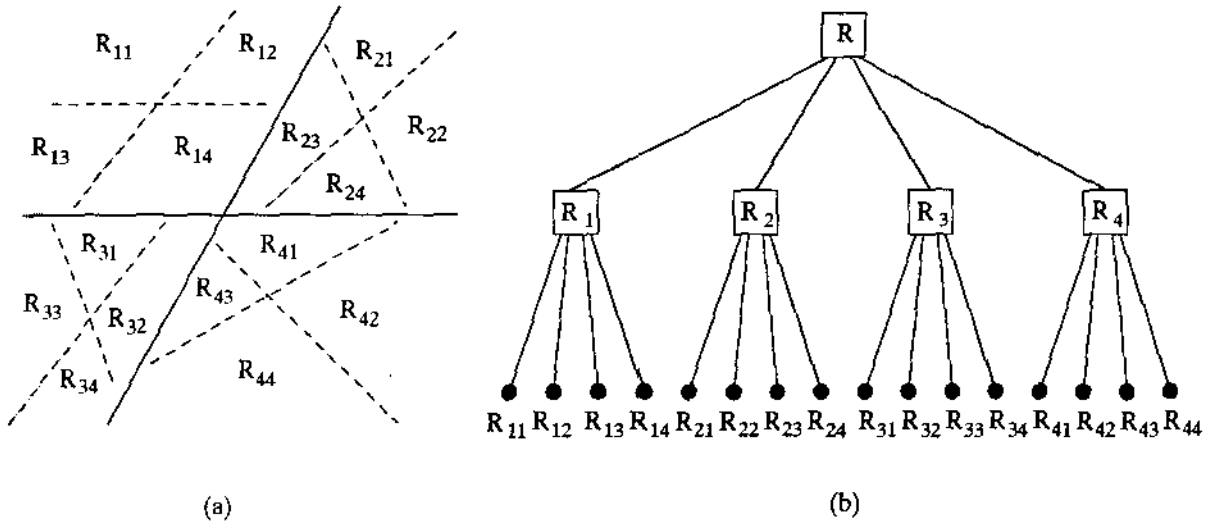


Figura 5.3: Uma hierarquia de J-partições (a) e a árvore de Willard correspondente (b).

```
function Busca(v,σ)
begin
   for i := 1 to 2 * J do
   (a J-partição associada a v decompõe R(v) em 2J sub-regiões {φ1,...,φ2J})
      if φi ⊆ σ then
         Todos os pontos de P(v) contidos em φi satisfazem à consulta;
      else if φi ∩ σ ≠ ∅ then
      (Se φi ∩ σ = ∅ nenhum ponto contido em φi satisfaz à consulta)
         Busca(v.filho(φi),σ);
         (v.filho(φi) é o filho de v relativo à face φi)
end;
```

O tempo de consulta ótimo para a solução de Willard é conseguido com $J = 3$. A seguir, são apresentadas as complexidades da solução de Willard para este valor de J.

Complexidades

C: $O(N^{\log_6 4} + k) \subset O(N^{0,774} + k)$
M: $O(N)$
P: $O(N^2)$ (pior caso)
$O(N \cdot \log^2 N)$ (caso médio)

5.1.2 Árvore de Edelsbrunner e Welzl

A solução de Edelsbrunner e Welzl [EW86] é um aperfeiçoamento da árvore de Willard para $J = 2$ (seção 5.1.1). A estrutura de dados proposta é também uma árvore de partição (seção 2.3), denominada **árvore conjugada** (*conjugation tree*) e baseia-se em um esquema de partição onde as regiões relativas a nós adjacentes na árvore *não* são particionadas independentemente: possuem uma linha divisória em comum. Uma árvore conjugada e a partição do plano correspondente são ilustradas na figura 5.4.

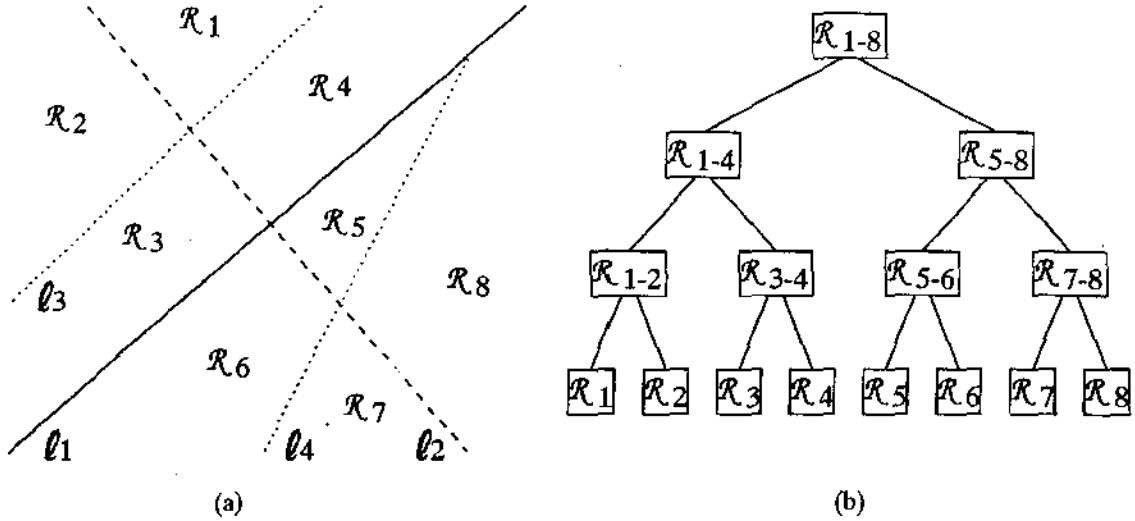


Figura 5.4: A solução de Edelbrunner e Welz: o esquema de partição do plano (a) e a árvore conjugada correspondente (b).

Esta estrutura permite reduzir o tempo de consulta em relação à solução de Willard.

Complexidades

C: $O(N^{\log_2(1+\sqrt{5})-1} + k) \subset O(N^{0,695} + k)$
M: $O(N)$
P: $O(N \cdot \log N)$ (pior caso)

5.1.3 Árvore de Paterson e Yao

Em [PY86], é descrito um procedimento de busca para o caso do espaço de busca σ ser uma região poligonal *convexa* não necessariamente limitada. Para regiões não convexas com fronteira composta de segmentos de reta o procedimento ainda funciona, após realizar uma decomposição adequada da região.

Decomposição do subespaço de busca

Seja o subespaço de busca σ uma interseção de semi-planos, diferente da região vazia e do plano todo. A fronteira de σ consiste de uma seqüência de arestas, sendo cada qual parte da reta limitante de algum semi-plano. Os vértices da fronteira de σ são os pontos onde duas arestas adjacentes se encontram.

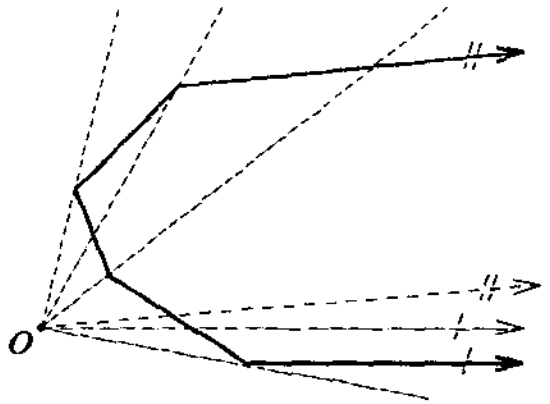


Figura 5.5: Decomposição de uma região poligonal de busca em setores.

O subespaço de busca σ é decomposto em um conjunto de sub-regiões mais simples, através da divisão do plano em **setores**, isto é, faixas em forma de cunha partindo da origem o do plano e estendendo-se para o infinito. A divisão em setores é obtida traçando-se raios semi-infinitos a partir da origem o do plano e passando por cada vértice de σ. Quando uma aresta de σ se estende para o infinito em alguma direção, introduz-se um raio paralelo a esta aresta (veja figura 5.5). Isso equivale a considerar um vértice no infinito em coordenadas homogêneas (seção 2.7).

A partição do plano em setores divide a região de busca em um conjunto de regiões que chamaremos **quadros** (*quads*). Cada quadro tem uma forma particularmente simples: pode ser visto como a interseção de um setor com uma **cunha dupla**[2], como é ilustrado na figura 5.6. Esta propriedade permite um algoritmo eficiente para busca dos pontos contidos dentro de um quadro, dado o conjunto de pontos contidos no setor correspondente e utilizando dualidade (seção 2.7).

Seja $\mathcal{P}(\varphi)$ o conjunto dos pontos de P contidos em um setor φ. No espaço dual, cada ponto $p = (a, b) \in \mathcal{P}(\varphi)$ equivale a uma reta r de equação $ax + by + 1 = 0$, enquanto uma cunha dupla D equivale a um segmento de reta $\overline{AB}$. O problema de determinar quais pontos de $\mathcal{P}(\varphi)$ estão contidos na cunha dupla D que cruza o setor φ equivale, no espaço dual, a determinar quais retas r duais aos pontos de $\mathcal{P}(\varphi)$ cruzam o segmento $\overline{AB}$ dual à cunha dupla D.

[2] Um par de retas que se cruzam divide o plano em quatro regiões. Uma cunha dupla (*double wedge*) pode ser definida como a união de duas dessas regiões cuja interseção seja apenas um ponto (o ponto de interseção das retas).

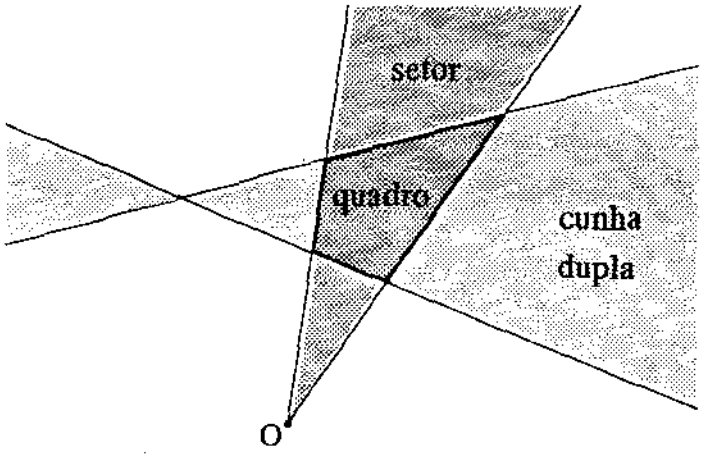


Figura 5.6: Um quadro como a interseção de um setor e uma cunha dupla.

Estrutura pré-processada para suportar buscas

Seja $T(P)$ a estrutura de busca para um conjunto P de N pontos no plano. Para construí-la, primeiramente os pontos de P são ordenados por seus ângulos em torno da origem o e rotulados de maneira que o i-ésimo ponto ($1 \leq i \leq N$) tenha ângulo em relação à origem θ_i com $0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_N < 2\pi$. Então, é construída a árvore binária de busca balanceada de profundidade $\lceil \log N \rceil$ com os N pontos nas folhas, nesta ordem da esquerda para a direita. Em cada nó intermediário desta árvore é armazenado o intervalo de ângulos $[\theta_i, \theta_j]$ correspondente às folhas dele descendentes. Esta estrutura de dados assemelha-se a uma árvore de segmentos (seção 3.1.3), sendo que ao invés da decomposição hierárquica de um domínio linear em segmentos canônicos, tem-se a decomposição hierárquica de um domínio circular (0 a 360 graus) em intervalos de ângulos canônicos (setores canônicos).

Uma recuperação restrita aos pontos com ângulos entre α e β é realizada efetuando-se buscas pelos valores min e max tais que $min = min\{i \mid \theta_i \geq \alpha\}$ e $max = max\{j \mid \theta_j \leq \beta\}$. Assim, um setor φ_j correspondente a um quadro Q_j, resultante da decomposição do subespaço de busca pode ser subdividido em no máximo $2 \cdot \log N$ setores canônicos, associados a nós v da árvore $T(P)$. A recuperação desejada pode então ser realizada em cada conjunto canônico de pontos $\mathcal{P}(v)$, contido no setor canônico $\varphi(v)$ associado ao nó v.

Uma região poligonal de busca σ com s lados é decomposta em $O(s)$ quadros para recuperações separadas. Então, a decomposição dos setores resultantes da subdivisão de σ gera um total de $O(s \cdot \log N)$ setores canônicos. O problema restante é enumerar todos os pontos desses setores canônicos contidos nas cunhas duplas correspondentes. Este problema pode ser resolvido em tempo $O(\log N + k_q)$ [Cha84] para cada setor canônico, onde k_q é o número de pontos contidos no quadro q correspondente. Então, uma consulta poligonal completa leva tempo $O(s \cdot \log^2 N + k)$. Aplicando-se buscas por propagação (seção 2.5), consegue-se reduzir o tempo de consulta para $O(s \cdot \log N + k)$.

Complexidades

C: $O(s \cdot \log N + k)$
M: $O(N^2)$
P: $O(N^2)$

onde s é o número de arestas da região poligonal de busca.

5.2 Cotas inferiores

Para busca em regiões poligonais (inclusive retringindo para regiões convexas) valem os limites inferiores obtidos para semi-espaços e simplexos (seção 6.2), com as equações de complexidade expressas para duas dimensões. Assim, do limite obtido por Chazelle (seção 6.2.1) resulta que uma busca em uma região poligonal no plano, no caso geral (permitindo consultas de contagem, enumeração e vacuosidade) no modelo aritmético (seção 1.5.1) requer tempo:

$$C : \Omega\left(\frac{N}{\sqrt{m}}\right) \tag{5.1}$$

onde m é um parâmetro indicando o uso de memória.

Para o caso particular de *enumeração* tem-se o limite de Chazelle e Rosenberg (seção 6.2.2) para o tempo de consulta, o qual é válido no modelo de máquinas de ponteiros (seção 1.5.3):

$$C : \Omega\left(\frac{N^{1-\varepsilon}}{\sqrt{m}} + k\right) \tag{5.2}$$

onde $\varepsilon > 0$ é uma constante, k é o tamanho da saída e m é o uso de memória.

Finalmente, ao caso particular de *contagem* dos pontos de P contidos em uma região poligonal do plano se aplica o limite inferior obtido por Brönnimann e Chazelle (seção 6.2.3), válido no modelo aritmético:

$$C : \Omega\left(\frac{(N/\log N)^{5/6}}{\sqrt{m}}\right) \tag{5.3}$$

como sempre, m é o uso de memória.

5.3 Notas bibliográficas

No início do artigo [PY86] pode ser encontrada uma pequena resenha dos resultados obtidos para o problema de busca em subespaço poligonal até então. A resenha de Matoušek sobre busca em semi-espaços e simplexos [Mat93a] é um trabalho mais recente e também contém referências ao

problema de busca em subespaço poligonal, embora esteja voltada primeiramente para busca em semi-espaços e simplexos (veja capítulo 6 a seguir).

Entre os trabalhos não apresentados nas soluções para busca em subespaço poligonal da seção 5.1 podemos citar [EKM82], que contém uma solução permitindo efetuar consultas em tempo $O(\log N + k)$, mas inviável devido à enorme utilização de memória: $O(N^7)$. Cole e Yap [CY85] obtiveram uma solução permitindo estabelecer relações de compromisso entre o uso de memória $O(N^{2+\varepsilon}/\log N)$ e o tempo de consulta $O(\log N \cdot \log(1/\varepsilon) + k)$, por meio do parâmetro $0 < \varepsilon < 1$. Ademais, com espaço $O(N^2/\log N)$ eles conseguem atingir tempo de consulta $O(\log N \cdot \log\log N + k)$.

Capítulo 6

Busca em semi-espaços e simplexos

Os problemas de busca em subespaços na forma de semi-espaços e busca em subespaços na forma de simplexos (ou simplesmente busca em semi-espaços e busca em simplexos) estão bastante relacionados, desde sua definição até as técnicas de solução a eles empregadas.

Um **semi-espaço** (figura 6.1-a) é uma região do espaço multidimensional delimitada por um hiperplano. Um **simplexo**, por sua vez (figura 6.1-b), é uma região conexa, cuja fronteira é composta de $d + 1$ faces relativas a $d + 1$ hiperplanos. Um simplexo pode ser visto como a interseção de $d + 1$ semi-espaços.

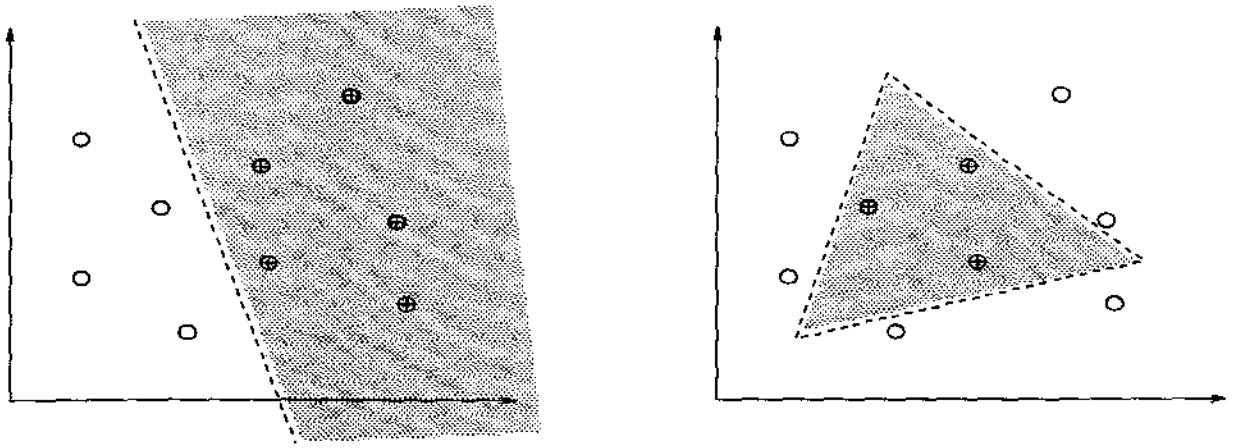


Figura 6.1: Busca em subespaços na forma de semi-espaços (a) e simplexos (b), em duas dimensões.

Busca em semi-espaços e busca em simplexos consistem em determinar os pontos da base de dados P contidos em subespaços na forma de semi-espaços e simplexos, respectivamente. Ambos os problemas são de fundamental importância. Soluções eficientes para estes problemas podem se traduzir em melhorias nas soluções de outros problemas como, por exemplo, "*ray*

shooting" [dBHO+91, AM92b, Sch92] e programação linear [Mat93b].

O problema de busca em semi-espaços destaca-se ainda pelo fato de problemas de busca em subespaços de diversos formatos serem redutíveis a ele via linearização (seção 2.8). Soluções para busca em semi-espaços podem ainda ser generalizadas para busca em simplexos (ou politopos com um número limitado de faces[1]), algumas vezes através de estruturas de dados multi-nível (seção 2.4). Há também certas soluções para busca em semi-espaços que se estendem diretamente para simplexos, sem necessidade de estruturas multi-nível [Mat93a]. Soluções para busca em simplexos são interessantes porque uma busca em um poliedro arbitrário pode ser reduzida a um conjunto de buscas em simplexos mediante uma partição do poliedro de busca.

As soluções para problemas de busca em semi-espaços e busca em simplexos seguem duas linhas: soluções utilizando memória aproximadamente linear, as quais só permitem efetuar buscas em tempo da ordem $O(N^{\alpha})$ $(1 - 1/\lfloor d/2 \rfloor \leq \alpha < 1)$ e soluções permitindo efetuar buscas em tempo poli-logarítmico, as quais requerem memória da ordem $O(N^{d+\beta})$ $(\beta \geq 0)$[2]. Algoritmos com relações de desempenho entre estes dois extremos podem ser obtidos por combinação dos dois tipos de soluções.

As soluções com memória quase linear têm sua origem nos primeiros métodos para busca em regiões poligonais no plano. Utilizam largamente dualidade (seção 2.7) e o paradigma de árvores de partição (seção 2.3), originário do trabalho pioneiro de Willard [Wil82b]. Diversos esquemas de partição têm sido propostos em uma série de trabalhos enfocando busca em semi-espaços, busca em simplexos ou ambos os problemas simultaneamente. A utilização de técnicas probabilísticas (tipicamente para geração dos esquemas de partição e construção das estruturas de dados) introduzida por Haussler e Welzl [HW86] e Clarkson [Cla86, Cla88] permitiu progressos na pesquisa de soluções mais eficientes. O paradigma de filtragens sucessivas (seção 2.2) é também aplicado freqüentemente (muitas vezes de forma implícita).

As soluções com tempo de busca poli-logarítmico baseiam-se, em geral, no paradigma do lugar geométrico (seção 2.1) em conjunção com algum método de compactação. Estas soluções são usualmente propostas apenas para busca em semi-espaços e não se generalizam para busca em simplexos a não ser com utilização de memória muito maior que $O(N^d)$. Uma solução para busca em simplexos em tempo poli-logarítmico com uso de memória próximo a $O(N^d)$ pode ser encontrada em [CSW90]; todavia é bem mais complicada do que as soluções para busca em semi-espaços com uso de memória aproximadamente igual.

A seguir (seção 6.1), são esboçadas algumas soluções para busca em semi-espaços e simplexos, a fim de ilustrar as técnicas envolvidas. Na descrição destas soluções, são evitados detalhes, por questão de espaço e para simplificar a leitura. O leitor interessado nos pormenores das soluções pode recorrer às referências fornecidas ao longo do texto. Na seção 6.1.1 é apresentado o fundamento teórico para soluções com tempo de consulta poli-logarítmico e na seção 6.1.2 é apresentado o algoritmo ótimo de Chazelle, Guibas e Lee [CGL85] para busca em semi-espaços

[1] Note que um simplexo é um politopo cuja fronteira consiste de um número constante de faces.

[2] Estes valores correspondem, aproximadamente, aos valores extremos que se pode obter para o tempo de busca e o uso de memória, variando o parâmetro m no limite inferior para o tempo de busca em simplexos $O(\frac{N}{m^{1/d}\log N})$ estabelecido por Chazelle [Cha89] (veja seção 6.2).

em duas dimensões no modo de enumeração. Este algoritmo utiliza buscas por propagação (seção 2.5) e foge um pouco ao estilo da maioria dos algoritmos propostos para problemas de busca em subespaços, os quais usualmente se utilizam de algum esquema de decomposição do espaço para induzir uma partição no conjunto de pontos.

Posteriormente, são destacados dois tipos de partições usualmente empregadas para produzir algoritmos baseados em estruturas de árvores de partição: partições simpliciais (*simplicial partitions*) e cortes simpliciais (*cuttings*) (seções 6.1.3 e 6.1.4, respectivamente)[3].

Ao final do capítulo, são oferecidas notas bibliográficas adicionais, para os interessados em se aprofundar em determinados assuntos ou estender os conhecimentos. Cabe ressaltar que grande parte das informações aqui contidas provêm da excelente resenha de Matoušek [Mat93a].

6.1 Algumas soluções

6.1.1 Busca em semi-espaços em tempo poli-logarítmico

O método básico para busca em semi-espaços com tempo de consulta poli-logarítmico provém do trabalho de Edelsbrunner, Kirkpatrick e Maurer [EKM82]. Este método consiste em aplicar o princípio da dualidade (seção 2.7), pelo qual cada ponto $p = (a_1, a_2, \ldots, a_d) \in \mathbf{R}^d$ corresponde a um hiperplano $\mathcal{D}(p)$, dado pela equação $a_1x_1 + a_2x_2 + \ldots + a_dx_d + 1 = 0$ no espaço dual e analogamente, cada hiperplano $h \in \mathbf{R}^d$ corresponde a um ponto $\mathcal{D}(h)$. Esta correspondência preserva não só relações de incidência, mas também de orientação: os pontos de P *acima* do hiperplano h correspondem aos hiperplanos de $H = \mathcal{D}(P)$ *acima* do ponto $q = \mathcal{D}(h)$. Aplicando esta dualidade, um problema de busca em *semi-espaço* é transformado em um problema de localização de pontos.

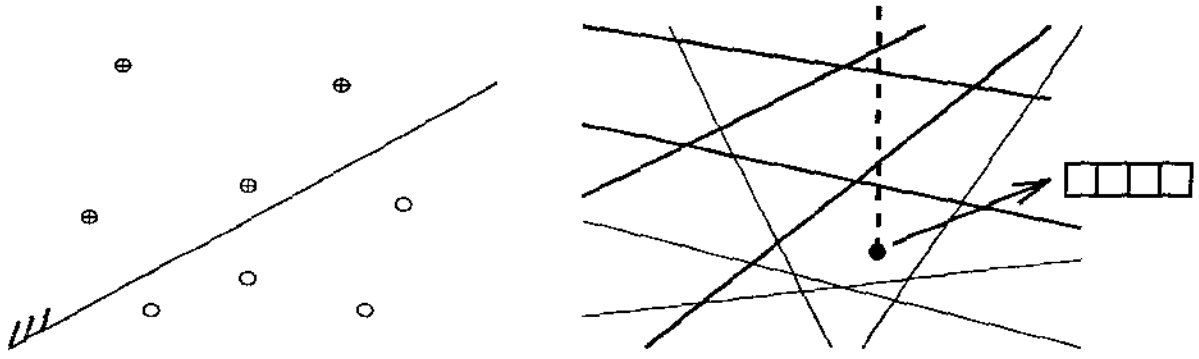
Ao conjunto P de N pontos corresponde um conjunto $H = \mathcal{D}(P)$ de N hiperplanos no espaço dual, denominado arranjo de hiperplanos.

Teorema 6.1 (Edelsbrunner [Ede87]) *Um arranjo H de N hiperplanos divide o espaço d-dimensional em $O(N^d)$ sub-regiões (faces), no pior caso.*

Seja $\sigma(h)$ o semi-espaço de busca acima do hiperplano h (em relação à coordenada γ_d). O caso em que $\sigma(h)$ fica abaixo de h pode ser tratado analogamente. O problema de determinar

[3]Temos aqui um problema de nomenclatura. Uma partição simplicial (*simplicial partition*) é dada por um conjunto de simplexos no espaço Γ *não necessariamente disjuntos ou cobrindo todo o espaço* e portanto, não corresponde a uma partição de Γ; mas induz uma partição de P. Um corte simplicial (*cutting*), por outro lado é uma partição do espaço Γ determinada por um conjunto de simplexos fechados, com interiores disjuntos e tais que sua união é igual a Γ. Um corte simplicial também induz uma partição do conjunto de pontos P, constituindo uma partição tanto do ponto de vista de P quando do espaço Γ e portanto, deveria ter recebido o nome de partição. Contudo, preferimos traduzir desta forma para manter a correspondência das denominações *simplicial partition* em inglês e partição simplicial em português, de modo a evitar confusões quando o leitor recorrer às referências em inglês.

os pontos de P acima do hiperplano h (contidos em $\sigma(h)$) reduz-se a determinar os hiperplanos de $H = \mathcal{D}(P)$ acima do ponto $q = \mathcal{D}(h)$ no espaço dual, como ilustra a figura 6.2.



(a) Busca em semi-espaço (b) Localização num arranjo de hiperplanos

Figura 6.2: Busca em semi-espaço (a) e o problema correspondente no espaço dual (b).

O arranjo de hiperplanos H divide o espaço dual em um conjunto de $O(N^d)$ regiões (teorema 6.1), em cada uma das quais o conjunto de hiperplanos acima é constante. Pode-se então armazenar as respostas pré-computadas para cada uma destas regiões (veja paradigma do lugar geométrico - seção 2.1) e utilizar localização de pontos para determinar a resposta a uma consulta. O problema de localização de pontos em um arranjo de N hiperplanos em d dimensões pode ser solucionado em tempo $O(\log N)$, mediante um pré-processamento adequado [Cla87, Cha93] (no plano pode-se utilizar o algoritmo de Kirkpatrick [Kir83]). O pré-processamento do arranjo de hiperplanos e o cálculo das respostas pré-computadas pode ser feito em tempo $O(N^d)$ (o arranjo de hiperplanos pode ser construído em tempo ótimo $O(N^d)$ [Ede87, CSW90]).

O inconveniente desta solução para o caso de *enumeração* é obviamente a quantidade de memória requerida: um fator $O(N)$ a mais que o limite inferior de Chazelle para busca em simplexos (seção 6.2.1), aproximadamente.

Complexidades

C: $O(\log N + k)$
M: $O(N^{d+1})$
P: $O(N^{d+1})$

Para problemas de *contagem*, desaparece o termo k do tempo de busca e o uso de memória reduz-se para $O(N^d)$.

6.1.2 Uma solução ótima para enumeração dos pontos contidos em um semi-espaço no plano

Chazelle, Guibas e Lee [CGL85] conseguiram solucionar o problema de busca em semi-espaços no modo de *enumeração* no plano em tempo logarítmico, utilizando memória linear. Esta solução foge ao padrão usual da maioria das soluções para busca em semi-espaços. Ao invés de utilizar um esquema de decomposição do espaço Γ para determinar uma partição do conjunto de pontos P (abordagem topológica), eles constróem as camadas convexas (*convex layers*) $C(P) = \{C_1, C_2, \ldots, C_s\}$, relativas a P e calculam sua interseção com o semi-plano de busca. As camadas convexas para um conjunto de pontos no plano são ilustradas na figura 6.3.

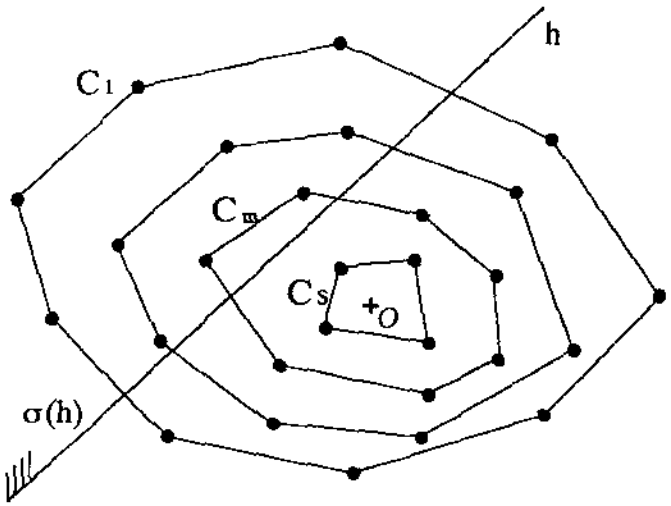


Figura 6.3: Camadas convexas cortadas por um semi-plano.

Seja $Conv(P)$ o conjunto dos vértices da envoltória convexa de um conjunto de pontos P. As camadas convexas $C(P) = \{C_1, C_2, \ldots, C_s\}$ para P são construídas segundo a seguinte definição recursiva:

$$C(P) := \begin{cases} \{Conv(P)\}, & \text{se } Conv(P) = P; \\ \{Conv(P)\} \cup C(P - Conv(P)), & \text{caso contrário.} \end{cases}$$

As camadas convexas podem ser construídas em tempo $O(N \log N)$ e ocupam memória $\theta(N)$ [Cha85b].

Chazelle, Guibas e Lee constróem uma malha de ponteiros bidirecional entre os cascos das camadas convexas $C(P)$, de modo que se possa fazer a enumeração dos pontos contidos em $\sigma(h)$, a partir da determinação de um vértice p de $C(P)$ contido em $\sigma(h)$, como ilustra a figura 6.4. O uso de memória é aumentado por um fator constante pela inserção desses ponteiros à estrutura de camadas convexas, permanecendo linear.

Esta estrutura de ponteiros possui similaridades com a estrutura proposta para solucionar

busca iterativa da seção 2.5, podendo também ser vista como uma aplicação da técnica de buscas por propagação.

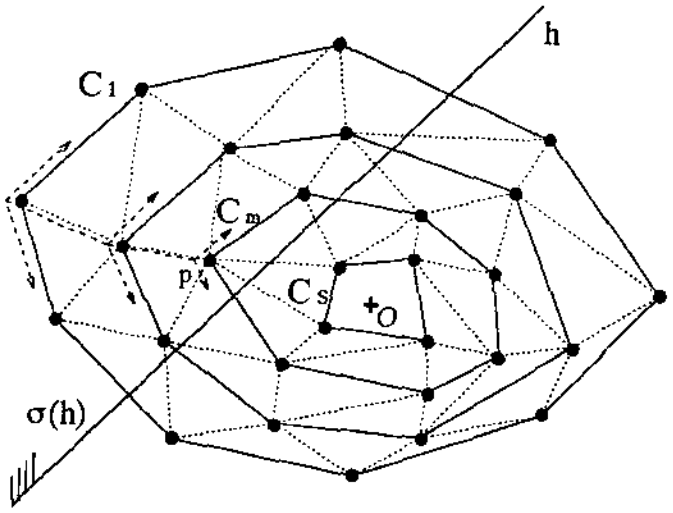


Figura 6.4: Busca em um semi-plano utilizando camadas convexas com cascos interligados.

Para efetuar uma busca em um semi-plano $\sigma(h)$ é necessário determinar se o hiperplano h limitante de $\sigma(h)$ (no caso uma linha) intercepta algum casco das camadas convexas $C(P)$. Chazelle, Guibas e Lee propõem utilizar a técnica de busca de Fibonacci [CD80] para solucionar este problema de forma simples em tempo $O(\log N)$.

Também se pode determinar se h intercepta algum casco de $C(P)$ utilizando dualidade (seção 2.7). As camadas convexas $C(P)$ correspondem a outras camadas convexas $\mathcal{D}(C(P))$ no espaço dual, com a ordem dos cascos convexos invertida. Determinar se o hiperplano h cruza um casco $C_i \in C(P)$ equivale no espaço dual a determinar se o ponto $\mathcal{D}(h)$ é interno ao polígono convexo $\mathcal{D}(C_i)$, o que pode ser realizado em tempo $O(\log n)$, onde n é o número de vértices de $\mathcal{D}(C_i)$, que é igual ao número de vértices de C_i.

Se h não intercepta nenhum casco de $C(P)$ isto é, não intercepta o casco mais externo de $C(P)$, então os pontos de P estão todos dentro ou todos fora de $\sigma(h)$, dependendo da orientação do semi-plano. Neste caso, o problema é trivial.

Se h intercepta $C(P)$, por outro lado, é necessário determinar o casco mais interno C_m interceptado por h. No espaço dual este problema se reduz a localização do ponto $q = \mathcal{D}(h)$ nas camadas convexas $\mathcal{D}(C(P))$ e pode ser solucionado em tempo $O(\log N)$, utilizando o método de Kirkpatrick [Kir83]. Entretanto, o emprego deste método constitui um esforço desnecessário neste ponto da solução. Chazelle, Guibas e Lee propõem um método mais prático (com constantes multiplicativas menores nas medidas de complexidade, resultando em maior eficiência na prática) para efetuar esta localização em tempo de pior caso $O(\log N)$ também, utilizando o paradigma de filtragens sucessivas (seção 2.2)[4]. A localização de C_m também pode ser efetuada

[4] Detalhes a respeito deste método de localização devem ser pesquisados no artigo [CGL85].

em tempo $O(\log^2 N)$, no pior caso, por meio de uma busca binária nas camadas convexas do espaço dual.

Uma vez determinado o casco C_m e um vértice p deste contido em $\sigma(h)$, basta percorrer os ponteiros ligando os vértices dos cascos na direção dada pela orientação de $\sigma(h)$, enumerando os pontos (correspondentes aos vértices dos cascos) contidos em $\sigma(h)$. O tempo gasto para efetuar este percurso é proporcional ao tamanho da saída. As linhas pontilhadas na figura 6.4 representam os ponteiros bidirecionais entre os cascos convexos e o caminho tracejado representa o processo de enumeração. Os detalhes desta solução podem ser encontrados no artigo [CGL85]) e as medidas de complexidade são sintetizadas abaixo.

Complexidades

C: $\theta(\log N + k)$
M: $\theta(N)$
P: $\theta(N \log N)$

6.1.3 Árvores de partição utilizando partições simpliciais

Muitas soluções para busca em semi-espaços e simplexos utilizando memória linear, como já foi mencionado na introdução deste capítulo, baseiam-se em alguma forma de árvore de partição (seção 2.4). Um esquema de partição possível de ser empregado neste contexto é a chamada **partição simplicial** (*simplicial partition*) [Mat93a, Mat91c, Mat91d].

Partição simplicial:

Seja P um conjunto de N pontos em Γ.
Uma partição simplicial é uma coleção $\Delta = \{(P_1, \delta_1), (P_2, \delta_2), \ldots, (P_\eta, \delta_\eta)\}$, onde os P_i's $(1 \leq i \leq \eta)$, denominados classes da partição simplicial, são subconjuntos disjuntos de P tais que $\bigcup_{i=1}^{\eta} P_i = P$ e cada δ_i é um simplexo contendo P_i.

O número η de simplexos de Δ é denominado **tamanho da partição simplicial**.

A figura 6.5 ilustra uma partição simplicial no plano. Note que os simplexos de Δ não precisam ser disjuntos nem cobrir o espaço Γ todo e que um mesmo ponto $p \in P$ pode estar contido em mais de um simplexo, sendo todavia associado a apenas um deles. Assim, os simplexos componentes de uma partição simplicial não determinam uma partição do espaço Γ, mas apenas do conjunto de pontos P.

O **número de cruzamentos em relação a um hiperplano** h (*crossing number*) de uma partição simplical Δ é o número de simplexos de Δ cortados por h. O **número de cruzamentos de uma**

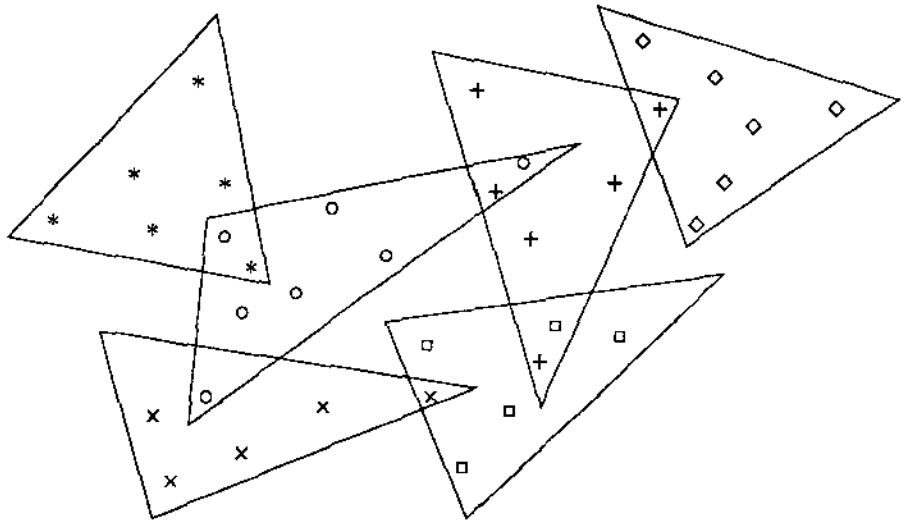


Figura 6.5: Uma partição simplicial no plano.

partição simplicial Δ é o máximo entre os números de cruzamentos de Δ com relação a qualquer hiperplano.

Teorema 6.2 (Matoušek [Mat91c]) *Sejam P um conjunto de N pontos em $\mathbf{R}^d$ e r um parâmetro, tal que $1 \leq r \leq N$. Então, existe uma partição simplicial Δ de tamanho r para P, satisfazendo $N/r \leq |P_i| \leq 2N/r$ para cada classe P_i $(1 \leq i \leq r)$ e com número de cruzamentos $O(r^{1-1/d})$.*

A partição simplicial satisfazendo tais critérios[5] pode ser construída em tempo $O(N \log r)$ no pior caso, para $r \leq N^{1-\lambda}$, onde $\lambda > 0$ é uma constante. Para qualquer $r \leq N$ uma partição simplicial satisfazendo os mesmos critérios pode ser conseguida em tempo $O(N^{1+\varepsilon})$, com $\varepsilon > 0$ [Mat91c].

Utilizando as partições simpliciais do teorema 6.2, Matoušek [Mat91c] constrói uma estrutura de dados T do tipo árvore de partição (seção 2.3), para auxiliar a responder buscas em semi-espaços e simplexos no modo de contagem.

Considere os pontos de P munidos de pesos relativos a um semi-grupo (seção 1.2.2). Cada nó intermediário de T corresponde a um subconjunto $P(v)$ de P e a uma partição simplicial $\Delta(v)$ para os pontos de $P(v)$, com as propriedades expressas no teorema 6.2. As folhas de T correspondem a uma partição de P em subconjuntos com tamanhos quase iguais (diferindo em no máximo 1) e limitados por uma constante. De cada partição simplicial $\Delta(v)$ são armazenados os simplexos (para calcular as interseções com o semi-espaço de busca) e os pesos acumulados dos pontos dos subconjuntos associados a cada um desses simplexos.

[5] Uma partição simplicial satisfazendo $N/r \leq |P_i| \leq 2N/r$ para cada classe P_i $(1 \leq i \leq r)$ é dita uma partição simplicial fina (*fine simplicial partition*).

O tamanho da partição simplicial associada a um nó v é $r = |P(v)|^{1-1/d}$. Assim, a cada nível da árvore o tamanho de $P(v)$ é diminuído de um fator $|P(v)|^{1/d}$, donde se pode concluir que a profundidade de T é $O(\log\log N)$ apenas.

Para responder uma busca em um subespaço σ (semi-espaço ou simplexo) parte-se da raiz. Estando em um nó v procede-se recursivamente nos filhos associados a simplexos de $\Delta(v)$ interceptados pela fronteira de σ. Os simplexos de Δ não interceptados por σ são tratados diretamente: as somas dos pesos relativos aos pontos associados a simplexos totalmente internos a σ são contabilizadas em cada nó do percurso de busca. A equação de recorrência para este processo de busca é:

$$T(N) \leq O(N^{1-1/d}) + O(r^{1-1/d})T(2N/r) \tag{6.1}$$

Considerando $r = N^{1-1/d}$ constante, a solução para a equação 6.1 é $O(N^{1-1/d}\log^c N)$, onde $c \geq 1$ é uma constante. Uma vez que um nó v da árvore de partição T baseada numa partição simplicial como especificada pelo teorema 6.2 usa espaço $O(|P(v)|^{1-1/d})$, o espaço total ocupado por T é linear. O tempo total de pré-processamento é $O(N\log N)$, pois cada nó v requer tempo de pré-processamento $O(|P(v)|\cdot\log|P(v)|)$ e o tamanho de $P(v)$ decresce como uma exponencial dupla com o aumento da distância de v à raiz[6].

Complexidades

C: $O(N^{1-1/d}\log^c N)$
M:$O(N)$
P: $O(N\log N)$

Pode-se utilizar estruturas de dados auxiliares penduradas aos nós de T para agilizar a detecção dos simplexos cortados pela fronteira de σ e o cálculo dos pesos acumulados dos pontos associados aos simplexos internos a σ. Com o uso de tais estruturas, pode-se diminuir o tempo de consulta para $O(N^{1-1/d}(\log\log N)^c)$, mediante um aumento do tempo de pré-processamento para $O(N^{1+\lambda})$, com $\lambda > 0$.

6.1.4 Árvores de partição utilizando cortes simpliciais

Outra construção importante em geometria computacional é o corte simplicial (*cutting*) [Mat93a, Mat91c].

[6] Detalhes em [Mat91c].

Corte simplicial (*cutting*):

Um corte simplicial é uma coleção de simplexos fechados $\Xi = \{\xi_1, \xi_2, \ldots, \xi_\eta\}$ com interiores disjuntos, cobrindo todo o espaço Γ.

O tamanho de um corte simplicial Ξ é o número η de simplexos[7] componentes de Ξ.

$(1/r)$-corte simplicial:

Dada uma coleção finita H de hiperplanos em Γ, diz-se que um corte simplicial $\Xi = \{\xi_1, \xi_2, \ldots, \xi_\eta\}$ é um $(1/r)$-corte simplicial (*$(1/r)$-cutting*) para H se:

$$|H(\xi_i)| \leq |H|/r \ \forall \xi_i \in \Xi \tag{6.2}$$

onde $H(\xi_i)$ é o subconjunto de hiperplanos de H interceptando o simplexo $\xi_i \in \Xi$ $(1 \leq i \leq \eta)$.

Cortes simpliciais podem ser usados como esquemas de partição e são de fundamental importância em algoritmos probabilísticos lidando com hiperplanos[8].

Sabe-se que, aplicando dualidade, o problema de busca em semi-espaços se reduz a calcular a interseção de um arranjo de hiperplanos com uma semi reta (seção 6.1.1). Um $(1/r)$-corte simplicial é pertinente neste contexto, pois permite reduzir o problema envolvendo N hiperplanos a um conjunto de sub-problemas envolvendo N/r hiperplanos. Esta estratégia de divisão, aplicada recursivamente, permite construir árvores de localização (seção 2.1) ou mesmo árvores de partição (seção 2.3) para auxiliar no processamento das consultas. Os algoritmos para busca em subespaços empregando cortes simpliciais podem utilizar resultados no espaço dual que implicam em outros resultados no espaço primal e vice-versa. Algumas vezes, estes algoritmos utilizam configurações no espaço primal possibilitando árvores de partição para solucionar os problemas de busca em subespaços e outras vezes, configurações no espaço dual possibilitando árvores de localização para solucionar os problemas correspondentes no espaço dual.

Teorema 6.3 (Chazelle e Friedman [CF90]) *Sejam H um conjunto de N hiperplanos em $\mathbf{R}^d$ e r um parâmetro, tal que $1 \leq r \leq N$. Então, existe um $(1/r)$-corte simplicial de tamanho $O(r^d)$ (que é o menor tamanho possível assintoticamente).*

[7] A definição de simplexo precisa ser estendida aqui para comportar também simplexos ilimitados, de modo que um conjunto finito de simplexos possa cobrir todo o espaço Γ. Considere então um simplexo como sendo uma região limitada ou não, constituída pelo espaço Γ todo ou pela interseção de um número finito de semi-espaços.

[8] Cortes simpliciais não são usados apenas como esquemas de partição. São empregados, por exemplo, como estruturas auxiliares na construção de partições simpliciais [Mat93a, Mat91c], entre outras aplicações.

Chazelle e Friedman [CF90] tratam da construção eficiente de cortes simpliciais e fornecem também um algoritmo probabilístico para construir um corte simplicial satisfazendo o teorema 6.3 em tempo esperado $O(Nr^{d-1})$. Este tempo de construção é ótimo se também forem retornadas as coleções $H(\xi_i)$ relativas a cada simplexo de Ξ, pois $O(Nr^{d-1})$ é um limite superior justo para a soma dos tamanhos dessas coleções.

Outros trabalhos sobre a construção eficiente de cortes simpliciais e aplicações a busca em subespaços publicados na mesma época são [Mat91a, Mat91b, Mat91d].

Recentemente, Chazelle [Cha93] conseguiu um algoritmo determinístico para calcular cortes simpliciais, com as mesmas medidas de complexidade do algoritmo probabilístico de Chazelle e Friedman.

O método de Chazelle permite construir uma hierarquia de cortes simpliciais, propiciando novas soluções para busca em subespaços. Esta hierarquia consiste de uma seqüência de cortes simpliciais $\{\Xi_0, \Xi_1, \ldots, \Xi_s\}$, onde Ξ_o corresponde ao único "simplexo" $\mathbf{R}^d$ e cada Ξ_i $(1 \leq i \leq s)$ refina Ξ_{i-1}, de tal forma que cada simplexo de Ξ_{i-1} é decomposto em uma coleção de simplexos de Ξ_i. Esta hierarquia sugere uma estrutura de árvore de partição.

Matoušek [Mat92] apresenta uma estrutura para busca em simplexos em $\mathbf{R}^d$ utilizando a hierarquia de cortes simpliciais de Chazelle. Ele demonstra também que é possível construir estruturas de dados multi-nível (seção 2.4) com uma sobrecarga (*overhead*) de apenas um fator logarítmico por nível da estrutura. As medidas de complexidade da solução de Matoušek (no modelo aritmético - seção 1.5.1) são enumeradas a seguir.

Complexidades

C: $O(N/m^{1/d} \cdot \log m/N)$
M: $O(m)$
P: $O(N^{1+\lambda} + m \log^{\lambda} N)$

onde m $(N \leq m \leq N^d)$ é um parâmetro permitindo diversas relações entre o uso de memória e o desempenho das buscas e $\lambda > 0$ é uma constante.

Com memória $m = O(N^d)$ pode-se efetuar consultas em tempo $O(\log^{d+1} N)$, mediante um pré-processamento da ordem $O(N^d \cdot \log^{\lambda} N)$. Com memória linear pode-se efetuar consultas em tempo $O(N^{1-1/d})$, sendo o tempo de pré-processamento $O(N^{1+\lambda})$.

A solução de Matoušek é quase ótima para $d = 2$ (salvo pelo fator $\log(m/N)$) e, de acordo com considerações a respeito do limite inferior obtido por Chazelle (seção 6.2.1), provavelmente é ótima também para $d > 2$.

6.2 Cotas inferiores

6.2.1 Limite inferior obtido por Chazelle

Chazelle [Cha89] estabelece o seguinte limite inferior para o tempo de busca em simplexos no modelo aritmético (seção 1.5.1):

$$C : \Omega\left(\frac{N}{m^{1/d}\log N}\right) \tag{6.3}$$

onde m ($N \leq m \leq N^d$) indica o uso de memória e $d > 2$ é o número de dimensões.

No plano, este limite aumenta para:

$$C : \Omega(N/\sqrt{m}) \tag{6.4}$$

em razão de uma prova mais acurada.

Isso sugere que o divisor logarítmico da equação 6.3 pode ser apenas resultado da técnica de prova utilizada.

Existem algumas cotas superiores para o problema de busca em simplexos atingindo os mesmos valores da cota inferior da equação 6.3, salvo por fatores multiplicativos de pequena significância: de ordem $O(N^\varepsilon)$ ou $O(\log^c N)$, com $\varepsilon > 0$ e $c > 0$ [CSW90, Mat92]. Assim, busca em simplexos é um problema quase resolvido no que diz respeito às cotas inferiores.

Acredita-se que o limite inferior de Chazelle (equações 6.3 e 6.4) se aplica também ao problema de busca em semi-espaços na sua definição geral (na qual não há restrições para os semi-espaços de busca e não importa se é efetuada contagem ou enumeração dos pontos ou se são efetuadas consultas de vacuosidade). Todavia, para alguns casos especiais de problemas de busca em semi-espaços existem soluções cuja eficiência supera o limite de Chazelle. Este é o caso, por exemplo, do problema de enumeração dos pontos em um semi-espaço [CGL85, CP86, CS89, AHL90] e do problema de vacuosidade [Mat91d].

6.2.2 Limite inferior obtido por Chazelle e Rosenberg para busca em simplexos no modo de enumeração

Chazelle e Rosenberg [CR92] demonstram o seguinte limite inferior para *enumeração* dos pontos contidos em um simplexo no modelo de máquinas de ponteiros (seção 1.5.3):

$$C : \Omega\left(\frac{N^{1-\varepsilon}}{m^{1/d}} + k\right) \tag{6.5}$$

onde $\varepsilon > 0$ é uma constante, k é o tamanho da saída, d é o número de dimensões do espaço e m é o uso de memória.

Este limite mostra que o problema de enumeração dos pontos em um simplexo é mais difícil que o problema de enumeração em semi-espaços, pois para o segundo problema existem soluções

com medidas de complexidade inferiores aos valores determinados pela cota inferior da equação 6.5, como por exemplo a solução ótima de Chazelle, Guibas e Lee para enumeração dos pontos contidos em um semi-espaço no plano (seção 6.1.2). Isso sugere que para simplexos não deve haver soluções mais eficientes para casos particulares do que as soluções para o caso geral, como ocorre para busca em semi-espaços.

6.2.3 Limite inferior obtido por Brönnimann e Chazelle para busca em semi-espaços no modo de contagem

Brönnimann e Chazelle [BC92] demonstram que um limite inferior para efetuar uma busca em *semi-espaço* no *modo de contagem* no modelo aritmético (seção 1.5.1) é:

$$C : \Omega\left(\frac{(N/\log N)^{1-\frac{d-1}{d(d+1)}}}{m^{1/d}}\right) \tag{6.6}$$

como sempre, d é o número de dimensões do espaço e m é o uso de memória.

Este limite (embora possa estar abaixo da complexidade intrínseca) mostra que o problema de busca em semi-espaços considerando pesos arbitrários é mais difícil que os casos particulares de enumeração dos pontos em um semi-espaço e o problema de vacuosidade, pois para estes existem algumas cotas superiores obtidas com valores de complexidade abaixo dos determinados pela equação 6.6, como por exemplo a solução ótima de Chazelle, Guibas e Lee para o caso de enumeração no plano (esboçada na seção 6.1.2) e algumas cotas superiores para enumeração e para o problema de vacuosidade com $d \geq 4$, encontradas em [Mat91d].

Eles também demonstraram que uma seqüência de N operações de inserção e busca em semi-espaços, partindo da base de dados vazia tem limite inferior:

$$\Omega\left(\frac{N^{2-\frac{3d+1}{(d+1)^2}}}{(\log N)^{\frac{d^2+1}{(d+1)^2}}}\right) = \Omega\left(\frac{N^{2-\Theta(1/d)}}{\log^{\theta(1)} N}\right) \tag{6.7}$$

6.3 Notas bibliográficas

Um quadro com cotas inferiores e superiores obtidas para problemas de busca em semi-espaços e busca em simplexos é apresentado no final da resenha de Matoušek [Mat93a].

A abordagem de árvores de partição tem sido predominante na pesquisa de soluções para busca em semi-espaços e simplexos. Uma série de artigos propuseram sucessivas reduções do expoente α do tempo de busca $O(N^\alpha)$ $(1 - 1/\lfloor d/2 \rfloor \leq \alpha < 1)$ com memória aproximadamente linear [Yao83, EH84, YDEP89][9] em 3 dimensões. Cole [Col85] obtém uma solução com tempo

[9] Esta evolução é caracterizada por Matoušek [Mat93a] de forma bem humorada, comparando-a ao estabelecimento de recordes esportivos.

de consulta sub-linear para 4 dimensões.

Essas soluções baseiam-se em decomposições do espaço Γ induzindo partições do conjunto de pontos P em subconjuntos de tamanhos iguais (tamanhos diferindo em no máximo 1, quando não for possível divisão inteira).

Avis [Avi84] demonstra que não é sempre possível particionar um conjunto de pontos em $\mathbf{R}^d$ em 2^d partes iguais utilizando d hiperplanos para $d \geq 5$. Isso implica que algumas estruturas baseadas em árvores de partição (por exemplo [Wil82b] e [Yao83]) não podem ser generalizadas para mais dimensões.

Andrew e Frances Yao [YY85], por outro lado, provam a existência de um esquema de partição em qualquer dimensão d fixa, tal que todo hiperplano em $\mathbf{R}^d$ cruza no máximo $2^d - 1$ regiões do espaço particionado. Eles apresentam uma construção elegante para efetuar esta prova, baseada numa analogia do conjunto de pontos P com uma certa distribuição de massas (no sentido da física mesmo) no espaço multidimensional. O espaço é dividido por d hiperplanos em 2^d regiões, cada uma das quais com uma fração 2^{-d} da massa total. Entretanto, esta construção não garante a distribuição dos pontos de P em partes iguais entre as regiões do esquema de partição. Eles provam apenas a existência e a unicidade de tal esquema de partição, deixando em aberto detalhes sobre sua construção e sua utilização na produção de estruturas de dados para auxiliar a resolver problemas de busca em subespaços.

Alon, Haussler e Welzl [AHW87] demonstram que qualquer problema de busca em subespaço de dimensão-*VC* igual a 1 (veja definição de dimensão-*VC* na seção 1.2.2) permite um esquema de partição que resulta em uma solução com tempo de busca sub-linear e também que o problema pode ser reduzido a busca em semi-espaço em um espaço de dimensão maior (veja técnicas de linearização na seção 2.8). Para alguns problemas de busca em subespaço abstratos com dimensão-*VC* maior que um a afirmação acima não é válida.

Haussler e Welzl [HW86] empregaram métodos probabilísticos para construir esquemas de partição. Eles introduziram o conceito de redes-ε (ε-*nets*) para um problema de busca em subespaço (P, Σ) com $P \subset \Gamma$ finito (o problema de busca em subespaço é formulado de forma abstrata). Outros trabalhos relacionados com redes-ε são [Mat89, KPW92]. Clarkson [Cla86, Cla88] também foi um dos pioneiros na aplicação de técnicas probabilísticas em geometria computacional.

Matoušek [Mat93a] descreve o esquema de partição de Haussler e Welzl com relativa simplicidade. Para um conjunto P de N pontos em $\mathbf{R}^d$ toma-se um subconjunto $Q \subset P$ com r pontos, onde r é um parâmetro tal que $1 < r \leq N/2$. O conjunto de todas as $\binom{r}{2}$ linhas determinadas por pares de pontos de Q constitui o esquema de partição. Haussler e Welzl provam que a árvore de partição baseada neste esquema tem alta probabilidade de responder consultas em tempo $O(N^{1-\frac{1}{d(d-1)+1}+\lambda})$ ($\lambda > 0$) no pior caso, utilizando memória linear.

Chazelle, Sharir e Welzl [CSW90] propõem um algoritmo para busca em simplexos utilizando vários esquemas de partição para um mesmo conjunto de hiperplanos (dual ao conjunto de pontos P), de tal forma que para qualquer hiperplano h sempre há um esquema de partição com baixo número de cruzamentos. Esta solução requer memória de ordem $O(N^{1+\varepsilon})$ e possibilita efetuar

buscas em tempo $O(N^{1-1/d+\varepsilon})$ $(\varepsilon > 0)$.

Agarwal, Eppstein e Matoušek [AEM92] seguem uma abordagem semelhante à de Chazelle, Sharir e Welzl para solucionar os problemas de enumeração dos pontos em um semi-espaço e de vacuosidade também relativo a semi-espaços, oferecendo diversas relações entre as medidas de desempenho para enumeração. Essencialmente, eles dinamizam a estrutura de dados de Clarkson [Cla87], utilizando a técnica de Chazelle, Sharir e Welzl para contornar obstáculos que impedem sua dinamização direta. Eles primeiro descrevem uma estrutura para o problema de vacuosidade e então a estendem para o problema de enumeração dos pontos contidos em um semi-espaço.

A maioria dos algoritmos e estruturas de dados para busca em semi-espaços têm uma característica predominantemente topológica. Eles se baseiam em algum esquema de partição, de uma forma ou de outra. Entretanto, existem outras abordagens na literatura. Entre as soluções utilizando abordagens não topológicas encontra-se a solução ótima de Chazelle, Guibas e Lee para enumeração dos pontos em um semi-espaço em duas dimensões [CGL85] (esta solução é descrita na seção 6.1.2).

Welzl [Wel88] propõe a idéia de árvore geradora com baixo nível de cruzamentos (*spanning trees with low crossing numbers*). Ao invés de calcular a interseção da fronteira do subespaço de busca com regiões de um esquema de partição, ele propõe calcular a interseção desta fronteira com as arestas de uma árvore geradora, a qual tem os pontos de P como vértices. O inconveniente desta abordagem é que o cálculo das interseções é quasè tão difícil quanto o problema original. Outros trabalhos utilizando árvores geradoras são [CW89, Mat90, Hau91].

Aggarwal, Hansen e Leighton [AHL90] conseguiram produzir uma solução para busca em semi-espaços em 3 dimensões, utilizando um método empregado para busca em subespaço circular no plano (veja seção 4.1.3). Eles aplicam uma transformação ao problema de busca em semi-espaço em três dimensões, reduzindo-o ao problema dos K-vizinhos mais próximos em duas dimensões, na métrica da "distância força" (*power distance metric*)[10]. Eles utilizam diagramas de Voronoi de K-ésima ordem na métrica da distância força para solucionar o problema, recorrendo novamente a uma solução de cunho topológico.

Matoušek e Welzl [MW92] descrevem um método para solucionar busca em semi-espaços (também busca em simplexos se os pesos puderem ser subtraídos) em duas dimensões, baseado em um lema de Erdős e Szekeres. Esta solução permite efetuar buscas em tempo $O(\sqrt{N}\log N)$ utilizando memória $O(N\log N)$. Parece não haver muita esperança de estender o método para mais dimensões. Todavia, é um algoritmo de fácil implementação e com pequenas constantes multiplicativas nas medidas de complexidade.

[10] Na métrica da distância força é associado um peso w_p a cada ponto $p \in \Gamma$. A distância entre dois pontos distintos p e q é definida na métrica da distância força como $dist_{dp}(p,q) = dist(p,q)^2 - {w_p}^2$, onde $dist(p,q)$ é a distância euclideana entre p e q.

Capítulo 7

Conclusões

Neste trabalho, foram estudadas algumas variações do problema de busca em subespaços (*range search*), segundo o enfoque de projeto de algoritmos eficientes. Em primeiro lugar, foram fornecidos os conceitos e classificações básicos relativos a problemas de busca em geral e particularmente busca em subespaços, a fim de prover a base teórica necessária e situar a área de estudo. Consideramos problemas de busca em subespaços com dados pontuais mergulhados em espaços de várias dimensões e exploramos algumas variações de formatos de subespaços de busca clássicas na literatura.

Foram analisadas algumas soluções encontradas na literatura para as diversas variações do problema (caracterizadas por diferentes formatos de subespaços de busca), de um ponto de vista abrangente, procurando enfatizar os seus aspectos em comum. Deste estudo, identificamos diversas abordagens e técnicas ocorrendo freqüentemente nas soluções, as quais denominamos paradigmas de algoritmos. Esses paradigmas foram descritos no capítulo 2, anteriormente aos capítulos dedicados às variações do problema (capítulos 3 a 6), com o intuito de introduzir ao leitor algumas formas alternativas de relacionar as diversas soluções, induzindo-o a desenvolver raciocínios gerais que lhe habilitem a perceber as semelhanças de concepção das soluções e as abordagens e técnicas gerais nelas empregadas.

7.1 Contribuições

Neste compêndio, procuramos descrever diversas idéias provenientes da pesquisa de algoritmos de maneira integrada e simples, com o objetivo de torná-las mais acessíveis tanto aos interessados na pesquisa dos problemas de busca em subespaços e suas soluções, quanto àqueles mais interessados nas possíveis aplicações dessas idéias ou dos próprios métodos de solução.

Dentre as contribuições deste trabalho podemos citar:

- a reunião e a formalização de conceitos relacionados com problemas de busca multidimensional e particularmente de busca em subespaços;
- um levantamento bibliográfico amplo, com classificação dos assuntos abordados na literatura, facilitando a identificação, localização e estudo de assuntos específicos;
- a descrição de diversas soluções para problemas de busca em subespaços sob uma visão unificadora e com notação uniforme, permitindo compará-las do ponto de vista de sua concepção e funcionamento;
- a coleta e a descrição de abordagens e técnicas gerais empregadas na solução de problemas de busca e outros, promovendo o desenvolvimento de raciocínios sistemáticos aplicáveis nas soluções de diversos problemas em computação.

Uma das questões que podem ser levantadas é o quanto paradigmas de algoritmos podem contribuir na produção de soluções gerais para problemas de busca em subespaços. Cabe lembrar então que não existe uma solução boa para todos os tipos de subespaços. Em diversos casos particulares as soluções gerais perdem em eficiência e simplicidade para as soluções específicas e em alguns casos, existem inclusive soluções específicas mais eficientes que os limites inferiores estabelecidos para o caso geral. Porém, algumas soluções genéricas podem ser úteis e, para algumas subclasses de problemas, as primeiras soluções não triviais foram conseguidas com a aplicação (às vezes implícita) de paradigmas de algoritmos, como é o caso das soluções para busca em subespaços polinomiais.

7.2 Possíveis extensões

Muitos trabalhos têm sido publicados nos últimos anos a respeito de problemas de busca em subespaços, especialmente semi-espaços, simplexos e subespaços polinomiais e semi-algébricos. Embora muitas das soluções alcancem as cotas inferiores, salvo por pequenos fatores multiplicativos, em geral poli-logarítmicos, ainda não se pode considerar o problema solucionado, mesmo do ponto de vista algorítmico, pois soluções mais simples e adequadas às diversas situações podem ser concebidas. Atualmente, tem se difundido também o emprego de técnicas probabilísticas em geometria computacional [Mul93], o que pode levar a soluções mais eficientes, especialmente no caso médio. Além disso, muitas das soluções obtidas jamais foram implementadas ou testadas em situações práticas.

Tem-se então um amplo campo de pesquisa em aberto, notadamente no que se refere a estudar a viabilidade de aplicação das idéias provenientes da pesquisa de algoritmos em diversas áreas. Entre os temas em aberto, relacionados com a aplicação dos resultados já obtidos, incluem-se:

- generalização das soluções de modo a permitir o tratamento de objetos de dados não pontuais [Sam90, Cox91];

- dinamização das estruturas de dados utilizadas para suportar as buscas [Ove83, CT92];
- adequação das estruturas de dados aos mecanismos de armazenamento disponíveis [Sil81, IKO88, OSdBvK90, SO90];
- implementação e experimentação das soluções.

Convenções de Notação

Convenções de Tipo

Texto normal: utilizado na narração comum.

Termos no idioma original: range, *simplex, cutting.*

Realce: repetidas consultas, dados pontuais, único.

Definição/conceito: busca em subespaço, partição-J.

Citações: Abençoados os que vêem beleza em tudo: têm espírito construtivo.

Abreviações: M:, MAOP, VC

Nomes de funções: empilha(), busca()

Símbolos, expressões e abreviações

Γ	espaço multidimensional ($\Gamma = \gamma_1 \times \gamma_2 \times \cdots \times \gamma_d$)
Δ	partição simplicial (*simplicial partition*) ($\Delta = \{(\mathcal{P}_1, \delta_1), (\mathcal{P}_2, \delta_2), \cdots, (\mathcal{P}_\eta, \delta_\eta)\}$ | $\mathcal{P}_i = P \cap \delta_i$, $\{\bigcup_{1 \le i \le \eta} \mathcal{P}_i\} = P$, $\mathcal{P}_i \cap \mathcal{P}_j = \emptyset \, \forall \, i \neq j$)
Ξ	partição do espaço multidimensional Γ em um conjunto de simplexos (*cutting*) ($\Xi = \{\xi_1, \xi_2, \cdots, \xi_\eta\}$ | $\{\bigcup_{1 \le i \le \eta} \xi_i\} = \Gamma$, $\xi_i \cap \xi_j = \emptyset \, \forall \, i \neq j$)
Σ	conjunto (possivelmente infinito) de subespaços de busca válidos para um determinado problema de busca em subespaço ($\Sigma \subseteq 2^\Gamma$)
Φ	conjunto de sub-regiões componentes de uma decomposição qualquer do espaço multidimensional Γ
Υ	conjunto de grupamentos de faces ($\Upsilon = \mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_\tau$)

φ	sub-região (face) componente de uma decomposição qualquer do espaço multidimensional ($\varphi \in \Phi$)
N	tamanho da base de dados ($N = \|P\|$)
P	conjunto de pontos da base de dados ($P \subseteq \Gamma$, $P = \{p_1, p_2, \cdots, p_N\}$)
d	número de dimensões do espaço multidimensional
k	tamanho da saída (em problemas de enumeração - seção 1.2.2)
$\mathcal{D}$	dual de uma entidade geométrica
$\mathcal{F}$	grupamento (subconjunto) de faces de uma decomposição do espaço multidimensional ($\mathcal{F} = \{\varphi \in \Phi\}$)
$\mathcal{P}$	subconjunto de P ($\mathcal{P} \subseteq P$)
$\mathcal{R}$	sub-região do espaço multidimensional Γ
$\mathbf{N}$	conjunto dos números naturais
$\mathbf{R}$	conjunto dos números reais
$\mathbf{Z}$	conjunto dos números inteiros
VC	dimensão de Vapnik-Chervonenkis ou dimensão-*VC* (*VC-dimension*)
$\partial\sigma$	fronteira de um subespaço de busca
$w(p)$	peso associado ao ponto $p \in P$
$Sub(P, \Sigma)$	todos os subconjuntos da base de dados P satisfazendo a consultas (*queries*) relativas a um conjunto de subespaços de busca Σ ($Sub(P, \Sigma) \subseteq 2^P$, $Sub(P, \Sigma) = \{\mathcal{P} = P \cap \sigma \mid \sigma \in \Sigma\}$)
$Viz_K(\mathcal{P})$	diagrama de k-ésimo vizinho mais próximo para o conjunto de pontos $\mathcal{P}$
$Vor_K(\mathcal{P})$	diagrama de Voronoi de k-ésima ordem relativo ao conjunto de ponto $\mathcal{P}$

Bibliografia

[AEM92] P. K. Agarwal, D. Eppstein, and J. Matoušek. Dynamic half-space reporting, geometric otimization, and minimum spanning trees. In *Proc. 33nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 80–89, 1992.

[AGSS87] A. Aggarwal, L. J. Guibas, J. Saxe, and P. Shor. A linear time algorithm for computing the Voronoi diagram of a convex polygon. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 39–45, 1987.

[AHL90] A. Aggarwal, M. Hansen, and T. Leighton. Solving query-retrieval problems by compacting Voronoi diagrams. In *Proc. 22nd Annu. ACM Sympos. Theory Comput.*, pages 331–340, 1990.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

[AHU83] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA, 1983.

[AHW87] N. Alon, D. Haussler, and E. Welzl. Partitioning and geometric embedding of range spaces of finite Vapnik-Chervonenkis dimension. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 331–340, 1987.

[AM92a] P. K. Agarwal and J. Matoušek. On range search with semialgebraic sets. In *Proc. 17th MFCS - Internat. Sympos. on Mathematical Found. of Computer Science*, volume 629 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 1992.

[AM92b] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 517–526, 1992.

[Avi80] D. Avis. Lower bounds for geometric problems. In *Proc. 18th Allerton Conf. Commun. Control Comput.*, pages 35–40, 1980.

[Avi84] D. Avis. Non-partitionable point sets. *Inform. Process. Lett.*, 19:125–129, 1984.

[BC92] H. Bronnimann and B. Chazelle. How hard is halfspace range searching? In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 271–275, 1992.

[Ben75] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[Ben79a] J. L. Bentley. Decomposable searching problems. *Inform. Process. Lett.*, 8:244–251, 1979.

[Ben79b] J. L. Bentley. Multidimensional binary search trees in database applications. *IEEE Trans. Softw. Engrg.*, SE-5:333–340, 1979.

[Ben80] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

[Ben90] J. L. Bentley. K-d trees for semidynamic point sets. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 187–197, 1990.

[BF79] J. L. Bentley and J. H. Friedman. Data structures for range searching. *ACM Comput. Surv.*, 11:397–409, 1979.

[BM79] J. L. Bentley and H. A. Maurer. A note on Euclidean near neighbor searching in the plane. *Inform. Process. Lett.*, 8:133–136, 1979.

[BM80] J. L. Bentley and H. A. Maurer. Efficient worst-case data structures for range searching. *Acta Inform.*, 13:155–168, 1980.

[BO83] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 80–86, 1983.

[BS75] J. L. Bentley and D. F. Stanat. Analysis of range searches in quad trees. *Inform. Process. Lett.*, 3:170–173, 1975.

[CCPY86] B. Chazelle, R. Cole, F. P. Preparata, and C. K. Yap. New upper bounds for neighbor searching. *Inform. Control*, 68:105–124, 1986.

[CD80] B. Chazelle and D. P. Dobkin. Detection is easier than computation. In *Proc. 12th Annu. ACM Sympos. Theory Comput.*, pages 146–153, 1980.

[CF90] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[CG86a] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.

[CG86b] B. Chazelle and L. J. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1:163–191, 1986.

[CGL85] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.

[Cha84] B. Chazelle. Intersecting is easier than sorting. In *Proc. 16th Annu. ACM Sympos. Theory Comput.*, pages 125–134, 1984.

[Cha85a] B. Chazelle. How to search in history. *Inform. Control*, 64:77–99, 1985.

[Cha85b] B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, IT-31:509–517, 1985.

[Cha86] B. Chazelle. Filtering search: a new approach to query-answering. *SIAM J. Comput.*, 15:703–724, 1986.

[Cha88] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17:427–462, 1988.

[Cha89] B. Chazelle. Lower bounds on the complexity of polytope range searching. *J. Amer. Math. Soc.*, 2:637–666, 1989.

[Cha90a] B. Chazelle. Lower bounds for orthogonal range searching, I: the reporting case. *J. ACM*, 37:200–212, 1990.

[Cha90b] B. Chazelle. Lower bounds for orthogonal range searching, II: the arithmetic model. *J. ACM*, 37:439–463, 1990.

[Cha93] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.

[Cla86] K. L. Clarkson. Further applications of random sampling to computational geometry. In *Proc. 18th Annu. ACM Sympos. Theory Comput.*, pages 414–423, 1986.

[Cla87] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 2:195–222, 1987.

[Cla88] K. L. Clarkson. Applications of random sampling in computational geometry, II. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 1–11, 1988.

[Col85] R. Cole. Partitioning point sets in 4 dimensions. In *Proc. 12th Internat. Colloq. Automata Lang. Program.*, volume 194 of *Lecture Notes in Computer Science*, pages 111–119. Springer-Verlag, 1985.

[Col86] R. Cole. Searching and storing similar lists. *J. Algorithms*, 7:202–220, 1986.

[Cox91] F. S. Cox Jr. Análise de métodos de acesso a dados espaciais aplicados a sistemas gerenciadores de bancos de dados. Master's thesis, Depto. de Ciência da Computação da Universidade Estadual de Campinas, Dezembro 1991.

[CP86] B. Chazelle and F. P. Preparata. Halfspace range search: an algorithmic application of k-sets. *Discrete Comput. Geom.*, 1:83–93, 1986.

[CR92] B. Chazelle and B. Rosenberg. Lower bounds on the complexity of simplex range reporting on a pointer machine. In *Proc. 19th International Colloquium on Automata, Languages, and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 439–449. Springer-Verlag, 1992. Also to appear in *Comput. Geom. Theory Appl.*

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[CSW90] B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 23–33, 1990.

[CT92] Y.-J. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. *Proceedings of the IEEE*, 80(9):1412–1434, 1992.

[CW89] B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete Comput. Geom.*, 4:467–489, 1989.

[CY85] R. Cole and C. K. Yap. Geometric retrieval problems. *Inform. Control*, 63:39–57, 1985.

[dBHO+91] M. de Berg, D. Halperin, M. Overmars, J. Snoeyink, and M. van Kreveld. Efficient ray shooting and hidden surface removal. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 21–30, 1991.

[DL76] D. P. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM J. Comput.*, 5:181–186, 1976.

[DM85] D. P. Dobkin and J. I. Munro. Efficient uses of the past. *J. Algorithms*, 6:455–465, 1985.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[EH84] H. Edelsbrunner and F. Huber. Dissecting sets of points in two and three dimensions. Report F138, Inst. Informationsverarb., Tech. Univ. Graz, Graz, Austria, 1984.

[EKM82] H. Edelsbrunner, D. G. Kirkpatrick, and H. A. Maurer. Polygonal intersection searching. *Inform. Process. Lett.*, 14:74–79, 1982.

[ES86] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete Comput. Geom.*, 1:25–44, 1986.

[EW86] H. Edelsbrunner and E. Welzl. Halfplanar range search in linear space and $O(n^{0.695})$ query time. *Inform. Process. Lett.*, 23:289–293, 1986.

[FB74] R. A. Finkel and J. L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Inform.*, 4:1–9, 1974.

[FBF77] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226, 1977.

[FGPR91] P. Flajolet, G. Gonnet, C. Puech, and J. M. Robson. The analysis of multidimensional searching in quad-trees. In *Proc. 2nd ACM-SIAM Sympos. Discrete Algorithms*, pages 100–109, 1991.

[Fre81a] M. L. Fredman. A lower bound on the complexity of orthogonal range queries. *J. ACM*, 28:696–705, 1981.

[Fre81b] M. L. Fredman. Lower bounds on the complexity of some optimal data structures. *SIAM J. Comput.*, 10:1–10, 1981.

[Hau91] D. Haussler. Sphere packing numbers for subsets of the Boolean n-cube with bounded Vapnik-Chervonenkis dimension. Tech. Report UCSC-CRL-91-41, University of California at Santa Cruz, 1991.

[HW86] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 61–71, 1986.

[IKO88] C. Icking, R. Klein, and T. Ottmann. Priority search trees in secondary memory. In *Proc. Internat. Workshop Graph-Theoret. Concepts Comput. Sci. (WG '87)*, volume 314 of *Lecture Notes in Computer Science*, pages 84–93. Springer-Verlag, 1988.

[IRKV88] S. S. Iyengar, N. S. V. Rao, R. L. Kashyap, and V. K. Vaishnavi. Multidimensional data structures: Review and outlook. *Advances in Computers*, 27:69–119, 1988.

[Kir83] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12:28–35, 1983.

[Knu73] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1973.

[KPW92] J. Komlós, J. Pach, and G. Woeginger. Almost tight bounds for ϵ-nets. *Discrete Comput. Geom.*, 7:163–173, 1992.

[KSSS] H. P. Kriegel, M. Schiwietz, R. Schneider, and B. Seeger. Performance Comparison of Point and Spatial Access Method. In *Lecture Notes in Computer Science*, volume 409.

[Lee82] D. T. Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comput.*, C-31:478–487, 1982.

[Lim76] Elon Lages Lima. *Elementos de topologia geral.* Livros técnicos e científicos, Rio de Janeiro, RJ, 1976.

[LT79] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Applied Mathematics*, 36:177–189, 1979.

[LT80] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9:615–627, 1980.

[Lue78] G. S. Lueker. A data structure for orthogonal range queries. In *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 28–34, 1978.

[Lue79] G. S. Lueker. A transformation for adding range restriction capability to dynamic data structures for decomposable searching problems. Report R129, Dept. Inform. Comput. Sci., Univ. California, Irvine, CA, 1979.

[LW77] D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Inform.*, 9:23–29, 1977.

[LW82] G. S. Lueker and D. E. Willard. A data structure for dynamic range searching. *Inform. Process. Lett.*, 15:209–213, 1982.

[Mat89] J. Matoušek. Construction of ϵ-nets. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 1989.

[Mat90] J. Matoušek. More on cutting arrangements and spanning trees with low crossing number. Technical Report B-90-2, Fachbereich Mathematik, Freie Universität Berlin, Berlin, 1990.

[Mat91a] J. Matoušek. Approximations and optimal geometric divide-and-conquer. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 505–511, 1991. Also to appear in *J. Comput. Syst. Sci.*

[Mat91b] J. Matoušek. Cutting hyperplane arrangements. *Discrete Comput. Geom.*, 6:385–406, 1991.

[Mat91c] J. Matoušek. Efficient partition trees. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 1–9, 1991.

[Mat91d] J. Matoušek. Reporting points in halfspaces. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 207–215, 1991.

[Mat92] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 276–285, 1992.

[Mat93a] J. Matoušek. Geometric range search. preprint, 1993.

[Mat93b] J. Matoušek. Linear optimization queries. *J. Algorithms*, 14:432 448, 1993. The results combined with results of O. Schwarzkopf also appear in *Proc. 8th ACM Sympos. Comput. Geom.*, 1992, pages 16–25.

[Mel84] K. Melhorn. *Data Structures an Algorithms 3: Multidimensional Searching and Computational Geometry.* Springer-Verlag, Berlin, 1984.

[MN90] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica*, 5:215 241, 1990.

[Mul93] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, New York, 1993.

[MW92] J. Matoušek and E. Welzl. Good splitters for counting points in triangles. *J. Algorithms*, 13:307-319, 1992.

[OSdBvK90] M. H. Overmars, M. H. M. Smid, M. T. de Berg, and M. J. van Kreveld. Maintaining range trees in secondary memory, part I: partitions. *Acta Inform.*, 27:423–452, 1990.

[Ove81a] M. H. Overmars. Searching in the past I. Report RUU-CS-81-7, Dept. Comput. Sci., Univ. Utrecht, Utrecht, Netherlands, 1981.

[Ove81b] M. H. Overmars. Searching in the past II: general transforms. Report RUU-CS-81-9, Dept. Comput. Sci., Univ. Utrecht, Utrecht, Netherlands, 1981.

[Ove83] M. H. Overmars. *The design of dynamic data structures*, volume 156 of *Lecture Notes in Computer Science.* Springer-Verlag, 1983.

[OvL82] M. H. Overmars and J. van Leeuwen. Dynamic multi-dimensional data structures based on quad- and k-d trees. *Acta Inform.*, 17(3):267–286, 1982.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[PY86] M. S. Paterson and F. F. Yao. Point retrieval for polygons. *J. Algorithms*, 7:441–447, 1986.

[RF94] P. J. Rezende and R. Fileto. Paradigmas de algoritmos na solução de problemas de busca multidimensional. In *Anais do XXI Seminário Integrado de Software e Hardware*, 1994.

[Sam80] H. Samet. Deletion in two-dimensional quad trees. *Commun. ACM*, 23(12):703–710, 1980.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley Publishing Company, 1990.

[Sch92] O. Schwarzkopf. Ray shooting in convex polytopes. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 286–295, 1992.

[Sed88] R. Sedgewick. *Algorithms.* Addison-Wesley, Reading, MA, 1988.

[Sil78] Y. V. Silva Filho. Multidimensional search trees as indices of files. Report ??, Univ. Kent, Canterbury, England, 1978.

[Sil81] Y. V. Silva Filho. Optimal choice of discriminators in a balanced k-d binary search tree. *Inform. Process. Lett.*, 13:67–70, 1981.

[SO90] M. H. M. Smid and M. H. Overmars. Maintaining range trees in secondary memory, part II: lower bounds. *Acta Inform.*, 27:453–480, 1990.

[Spr91] R. L. Sproull. Refinements to nearest-neighbor searching in Kd-trees. *Algorithmica*, 6:579–589, 1991.

[Sto91] J. Stolfi. *Oriented Projective Geometry: A Framework for Geometric Computations.* Academic Press, 1991.

[SY82] J. M. Steele and A. C. Yao. Lower bounds for algebraic decision trees. *J. Algorithms*, 3:1–8, 1982.

[Tar85] R. E. Tarjan. Amortized computational complexity. *SIAM J. Algebraic Discrete Methods*, 6(2):306–318, 1985.

[Vai89] P. M. Vaidya. Space-time tradeoffs for orthogonal range queries. *SIAM J. Comput.*, 18:748–758, 1989.

[Wel88] E. Welzl. Partition trees for triangle counting and other range searching problems. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 23–33, 1988.

[Wil78] D. E. Willard. Predicate-oriented database search algorithms. Ph.D. Thesis and Report TR-20-78, Aiken Comput. Lab., Harvard Univ., Cambridge, MA, 1978.

[Wil82a] D. E. Willard. A new time complexity for orthogonal range queries. In *Proc. 20th Allerton Conf. Commun. Control Comput.*, pages 462–471, 1982.

[Wil82b] D. E. Willard. Polygon Retrieval. *SIAM Journal on Computing*, 11(1):149–165, 1982.

[Wil85] D. E. Willard. New data structures for orthogonal range queries. *SIAM J. Comput.*, 14:232–253, 1985.

[Wil89] D. E. Willard. Lower bounds for the addition-subtraction operations in orthogonal range queries and related problems. In *Inform. and Computation*, pages 45–64, 1989.

[WL85] D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32:597–617, 1985.

[Yao83] F. F. Yao. A 3-space partition and its applications. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 258–263, 1983.

[Yao85] A. C. Yao. On the complexity of maintaining partial sums. *SIAM J. Comput.*, 14:277–288, 1985.

[YDEP89] F. F. Yao, D. P. Dobkin, H. Edelsbrunner, and M. S. Paterson. Partitioning space for range queries. *SIAM J. Comput.*, 18:371–384, 1989.

[YY85] A. C. Yao and F. F. Yao. A general approach to d-dimensional geometric queries. In *Proc. 17th Annu. ACM Sympos. Theory Comput.*, pages 163–168, 1985.

[Zol78] J. E. Zolnowsky. Topics in computational geometry. Report ??, Dept. Comput. Sci., Stanford Univ., Stanford, CA, 1978.