ANÁLISE DE MÉTODOS DE ACESSO A DADOS ESPACIAIS APLICADOS A SISTEMAS GERENCIADORES DE BANCO DE DADOS

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida pelo Sr. Frederico Sidney Cox Junior e aprovada pela Comissão Julgadora.

Campinas, 19 de dezembro de 1991.

Prof. Dr. Geovane Cayres Magalhães
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.

Aos meus pais, a
Ana e a Manuela

Agradecimentos

Este trabalho corresponde ao requisito parcial para obtenção do título de Mestre em Ciência da Computação. A sua realização se deve, não somente ao esforço e dedicação empenhados nesse curso, mas a todo um processo que, durante a minha vida, motivou o desejo de realizá-lo.

Este processo foi fruto de uma conscientização e admiração a certas pessoas, adquiridas durante a minha formação, que, direta ou indiretamente, incentivaram a realização desse curso. A começar por meus pais que, sempre presentes em todas as fases importantes da minha vida, buscaram, sem limite de esforços, proporcionar a melhor educação e orientação. Ao meu irmão, através do sucesso obtido em sua vida acadêmica e profissional. Ao meu ex-professor e chefe, Geovane Cayres Magalhães, que, posteriormente, concedeu-me a oportunidade e a honra de ser seu orientando. Por fim, a minha esposa, que tornou este trabalho possível através do seu apoio, carinho e compreensão, e de nossa filha, que conseguiu tornar alegre os momentos difíceis.

Gostaria de agradecer também ao amigo Daniel Tavares Correia Xavier, que sempre esteve disposto a discutir sobre este trabalho. À Universidade Federal da Bahia e a todos que se empenharam para concessão de minha liberação, ao apoio financeiro concedido por essa Universidade, através do Plano Interno de Capacitação de Docentes e, por fim, ao Departamento de Ciência da Computação (UNICAMP), que pela minha aceitação, proporcionou a realização desse tão desejado curso.

Conteúdo

Capítulo 1

Introdução

O desenvolvimento dos recursos computacionais (em especial estações de trabalho com capacidade gráfica) favoreceu a proliferação de sistemas computacionais em diversas áreas. Em particular, uma classe de aplicações denominada espaciais tem atraído pesquisas com o objetivo de fornecer condições ao desenvolvimento destas aplicações. A demanda destas pesquisas é justificada pela importância que essas aplicações espaciais têm para a sociedade, através do desenvolvimento tecnológico e modernização de diversos setores, como, por exemplo: medicina (tratamento de imagens médicas); agricultura (análise de imagens de satélite); urbanismo (gerência de rede de telefonia, esgoto, energia e tráfico viário); indústria (robótica, visão e reconhecimento de padrões).

Dentre as diversas aplicações classificadas como espaciais, encontram-se aquelas que manipulam grandes volumes de dados e que, portanto, requerem auxílio de Sistemas Gerenciadores de Banco de Dados (SGBDs). Os requisitos especiais destas aplicações impossibilitam a simples adequação dos sistemas utilizados nos modelos convencionais para prover o suporte necessário. Um dos maiores problemas neste suporte reside nos tipos de dados e seus relacionamentos bem distintos daqueles existentes nos SGBDs. A adequação do suporte às aplicações espaciais por parte dos SGBDs, quanto aos requisitos de dados, envolve soluções para duas classes de problemas: gerenciamento de tipos de dados, que estuda os mecanismos para o usuário construir os tipos de dados apropriados para sua aplicação; e acesso aos dados, que estuda a forma de organização dos dados espaciais através das estratégias de representação e dos métodos de acesso a estes dados. Em conseqüência das estratégias de organização dos dados e dos métodos de acesso

estarem relacionados com o desempenho dos SGBDs, estas áreas de pesquisa têm atraído a atenção da comunidade de Banco de Dados e têm se mostrado como novas e importantes tendências de pesquisa.

1.1 Motivação e Evolução dos SGBDs

As pesquisas desenvolvidas na área de Banco de Dados iniciaram-se na década de 60. Estas pesquisas, que inicialmente foram desenvolvidas nos centros industriais com o objetivo de atender ao grande desenvolvimento das aplicações comerciais, foram motivadas pela dificuldade de gerenciar grandes volumes de dados.

As aplicações comerciais da época caracterizavam-se por um conjunto de programas que manipulavam dados em diversos arquivos. Este ambiente, um típico processamento de arquivos, era suportado apenas pelo sistema operacional e trazia diversas desvantagens fundamentais:

- Redundância de dados e inconsistência – devido ao grande número de arquivos, algumas informações eram duplicadas nestes arquivos, facilitando, assim, a inconsistência do sistema que ocorria quando a alteração destas informações não era propagada em todos os arquivos.
- Dificuldade no acesso aos dados – solicitação de informações não definidas pelo sistema demandava tempo, pela necessidade de elaborar programas específicos para recuperá-las.
- Múltiplos usuários – dificuldade de permitir acesso simultâneo aos dados por diversos usuários.
- Controle de segurança – dificuldade de restringir acesso aos dados a usuários específicos.
- Integridade – dificuldade de manter o sistema íntegro, conseqüente às alterações de restrição de consistência[1] necessitarem de modificações nos programas que as utilizam.

Desta forma, como resultado destas pesquisas, surgiram os SGBDs, que tinham como objetivo tornar mais eficaz o gerenciamento de grandes quanti-

[1] restrições de consistência são condições especiais definidas pelo usuário a determinados atributos no Banco de Dados (e.g., o atributo salário em uma dada aplicação não pode ter valor menor que o salário mínimo).

dades de dados. O sucesso destes sistemas justificou a sua evolução com o objetivo de acompanhar os novos requerimentos das aplicações. Esta evolução foi caracterizada por:

- desenvolvimento do modelo de dados[2] relacional, no início da década de 70, em virtude dos modelos de dados, hierárquico e de rede, tornarem os Sistemas de Banco de Dados da época custosos[3].

- desenvolvimento dos SGBDs distribuídos, no início da década de 80, para atender a tendência de organização descentralizada que emergia nos finais dos anos 70.

- investigações feitas, nos últimos anos, sobre a filosofia de "programação orientada a objetos", proporcionando elementos de dados de tipos mais genéricos do que os números e *strings* (cadeia de caracteres) existentes nos modelos tradicionais.

Atualmente, embora as pesquisas em Banco de Dados estejam bem sedimentadas com a utilização dos SGBDs em quase todos os ambientes computacionais, a variedade de áreas de aplicação onde esta tecnologia tem sido requisitada vem dirigindo as pesquisas para prover o melhor suporte às aplicações peculiares. Nessas áreas de aplicação, inclusive, encontram-se aquelas que se mostram como tendências de aplicações futuras: aplicações científicas, aplicações médicas, sistemas de projeto e fabricação e aplicações militares, dentre outras. Assim, surgem como novas e importantes questões a serem avaliadas pela comunidade de Banco de Dados:

- suporte a novos tipos de dados – imagens, pontos ou sólidos no espaço, *large arrays*, etc.

- necessidade de novos modelos de dados – espaciais, temporais, etc.

- necessidade de adaptar os algoritmos dos SGBDs para aplicações com ordem de magnitude muito maior que das aplicações da atualidade – estratégia para backup, recuperação, etc.

---

[2] modelo de dados é uma coleção de ferramentas conceituais para descrição dos dados, relacionamento entre os dados, semântica e restrições dos dados.

[3] os modelos de dados, hierárquico e de rede, além de possuírem uma *interface* de baixo nível entre os programas de aplicação e o SGBD, causavam manutenções nos programas de aplicação quando novos tipos de informações eram adicionadas nos SGBDs.

- integrar outro nível de armazenamento, terciário (*archive*), aos dois níveis já utilizados, primário e secundário – necessidade de algoritmos eficientes para gerenciar a transferência de informação entre esses três níveis.

- necessidade de estratégias para gerenciar dados de transações de longas durações – integridade, compartilhação e recuperação destes dados.

- controle de versões e configurações.

1.2 Suporte de Aplicações Espaciais em SGBDs

O desenvolvimento do *hardware* através dos dispositivos gráficos de alta resolução, largas memórias e alta velocidade de processamento propiciou o desenvolvimento e aplicação de software que otimize diversas áreas da sociedade.

Em particular, aplicações CAD/CAM (projeto e fabricação auxiliado por computador), aplicações geográficas e cartográficas, aplicações de gerenciamento urbano (rede de telefonia, tráfego viário e de energia) e aplicações com imagens (visão, robótica e reconhecimento de padrões), dentre muitas outras, têm-se integrado aos sistemas computacionais em conseqüência de possibilitarem suporte a estas aplicações peculiares.

O crescente desenvolvimento dessas aplicações trouxe dificuldades para organizar, criar e manter suas coleções de dados. Assim, tentou-se incorporar a tecnologia dos SGBDs a essas aplicações objetivando trazer os mesmos benefícios que os encontrados na integração da referida tecnologia às aplicações comerciais: eficiência no acesso e modificação dos dados, persistência, concorrência e segurança.

A incorporação da tecnologia dos SGBDs às aplicações citadas, porém, não se resume à simples adequação de um SGBD existente a cada uma destas aplicações. As aplicações comerciais, as quais os SGBDs atuais visam atender, possuem características bem distintas dessas novas aplicações.

1.2.1 Características das Aplicações Espaciais

A principal distinção entre aplicações comerciais (convencionais) e aplicações espaciais (não-convencionais) são seus tipos de dados básicos. Enquanto as aplicações comerciais possuem como tipos de dados, inteiros e cadeias de

caracteres (*strings*), as aplicações espaciais, embora com características distintas, possuem em comum tipos de dados que são pontos ou sólidos em n-dimensões e algumas vezes imagens. Aplicações CAD/CAM utilizam como dados objetos geométricos como círculos, pontos, linhas e retângulos, que são modelados a fim de formarem outros objetos mais complexos. Aplicações VLSI CAD (aplicações CAD para construção de circuitos VLSI), uma classe específica da anterior, possuem como objetos de dados polígonos de faces paralelas ao sistema de coordenadas. Aplicações geográficas e cartográficas requerem grande quantidade de dados, que são pontos, linhas e polígonos, normalmente no espaço bidimensional, para representar seus dados (ruas, bairros, avenidas, etc.) que são caracterizados por extrema irregularidade. Estes tipos de dados básicos, pontos ou sólidos em n-dimensões, são denominados **dados espaciais**, e as aplicações que os utilizam são caracterizadas como **aplicações espaciais**.

A denominação aplicações espaciais é muitas vezes atribuída a aplicações que possuem como característica o gerenciamento de imagens. Rigorosamente falando, as aplicações espaciais possuem como tipos de dados básicos pontos ou sólidos em n-dimensões com conhecimento explícito sobre os objetos, suas extensões e posições no espaço. As demais aplicações onde se encontra menos ênfase na análise dos dados e mais no armazenamento e recuperação de dados pictoriais não analisados (imagens) são denominados aplicações de imagens. A presença de imagens como tipo de dados nas aplicações espaciais é possível e freqüentemente utilizada, porém serve ou como fonte de recurso de onde os dados espaciais são derivados ou como abstração destes dados para prover uma melhor interface gráfica do sistema.

Os tipos de dados das aplicações espaciais também diferem das aplicações convencionais pela forma como se relacionam. Ao invés da relação existente nas aplicações convencionais, onde se busca determinar a proximidade em valor entre seus objetos de dados (superioridade, igualdade e inferioridade), nas aplicações espaciais busca-se determinar as relações entre os objetos de dados através de suas características (extensão e localização no espaço). Essa nova forma de relacionamento, denominada **relacionamento espacial**, visa determinar, dentre outras características: sobreposição, adjacência, intersecção e proximidade entre objetos.

Para exemplificar uma consulta envolvendo predicados espaciais, apresentou-se na figura 1.1, através da linguagem de consulta GEOQL proposta em [SDDO87], uma consulta que pretende determinar todas as cidades que são atravessadas pelo rio Tietê e que possuem população superior a 1.000.000 de habitantes. Nessa consulta, percebe-se que as aplicações espaciais também

possuem atributos não-espaciais (RIO.nome, CIDADE.população) e que, portanto, também devem ser suportados.

```
SELECT    CIDADE.nome
FROM      CIDADE, RIO
WHERE     RIO.nome = "TIETE" and
          CIDADE.população > 1.000.000 and
          RIO intersects CIDADE
```

Figura 1.1: Exemplo de consulta na linguagem GEOQL.

Em decorrência dos requisitos de dados das aplicações espaciais serem bem distintos dos das aplicações convencionais, os SGBDs destinados às aplicações espaciais devem possuir algumas características próprias para essa classe de aplicações (portanto, inexistentes nos SGBDs convencionais). O projeto de SGBDs capazes de suportar essas áreas de aplicação não tradicionais, **SGBDs Espaciais**, tem emergido como nova e importante tendência para pesquisas em Banco de Dados. Os requisitos necessários aos SGBDs, para prover suporte às aplicações espaciais, vão desde novas soluções pertinentes ao nível conceitual até o nível físico de implementação.

1.2.2 Novos Requisitos na Arquitetura Convencional

Devido às peculiaridades das aplicações espaciais, a integração destas aos SGBDs requerem a presença de requisitos que são ausentes nos modelos convencionais. Com o objetivo de melhor apresentar estes novos requisitos, a seguir será discutida a arquitetura de um SGBD Espacial proposto em [Ooi90] (figura 1.2) em três níveis de abstração: nível físico, representando a camada interna de um SGBD, onde encontram-se funções responsáveis pelo armazenamento e recuperação dos dados em disco; nível intermediário, que dentre outras funções deve prover a otimização das consultas feitas pelo usuário; e o nível de interface, que é responsável pela comunicação entre o sistema e o meio externo.

Nível Físico

Além de armazenar os dados não-espaciais que também estão presentes nas aplicações espaciais, um SGBD Espacial deve permitir o armazenamento dos

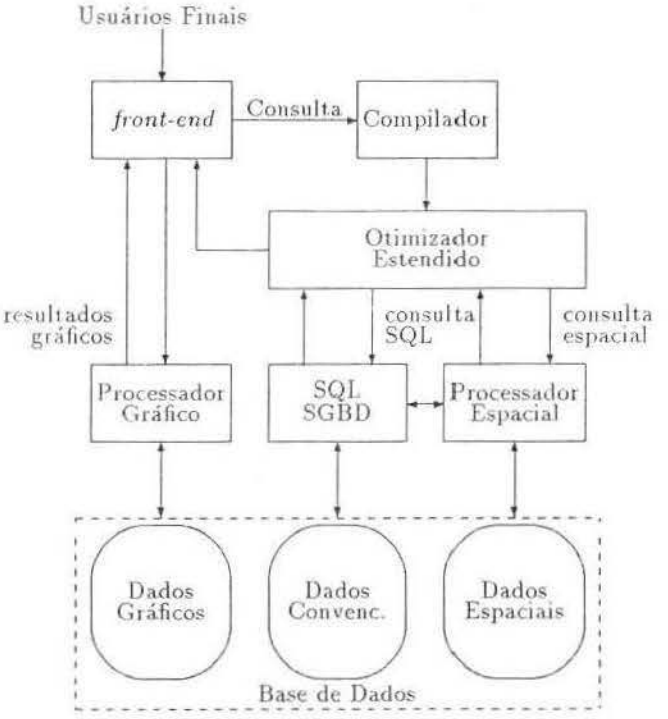


Figura 1.2: Arquitetura de um SGBD não convencional.

dados espaciais e dos dados gráficos. Estes dois últimos tipos de dados são inexistentes nos modelos convencionais e são responsáveis, respectivamente, pelos pontos e sólidos no espaço e pela representação gráfica dos dados espaciais. Assim, nas aplicações geográficas para gerenciamento urbano (telefonia, energia elétrica e saneamento), por exemplo, os dados espaciais estariam relacionados à representação das diversas ruas, avenidas, bairros, enquanto os dados gráficos relacionados, à representação destes mesmos objetos nos dispositivos gráficos.

A representação destes novos tipos de dados implica na necessidade de mecanismos para o seu armazenamento e a sua recuperação. Como será apresentado posteriormente, os mecanismos de recuperação para os dados espaciais são bem distintos dos utilizados para recuperação dos dados não-espaciais e representam novas modificações na arquitetura convencional.

Nível Intermediário

No nível intermediário, pode-se ressaltar a maior distinção da arquitetura convencional, o módulo responsável por otimizar as consultas (otimizador de consultas). Embora presente na arquitetura convencional, para SGBDs Espaciais, este módulo deve sofrer alterações.

A função de um otimizador de consultas é avaliar, para todos os predicados envolvidos em uma consulta, a melhor estratégia a ser utilizada para satisfazer a consulta. Como as consultas podem envolver predicados espaciais e não-espaciais, o otimizador de consulta deve ser estendido à interpretação destes novos predicados (espaciais) inexistentes na arquitetura convencional. Como exemplo, para a consulta apresentada na figura 1.1, o otimizador de consultas deve avaliar o melhor plano para satisfazê-la de forma mais rápida. Se é preferível recuperar primeiro todos os rios que possuem nome Tietê ou todas as cidades que possuem população superior a 1.000.000 de habitantes, através das técnicas convencionais, ou recuperar todas as cidades que são interceptadas por rios para depois avaliar os predicados restantes. A ordem em que os predicados são processados tem significado decisivo para a eficiência da resposta à consulta.

Nível de Interface

O nível de interface é responsável pela comunicação entre o sistema e o usuário. Como uma das principais motivações para o surgimento das aplicações espaciais foi a disponibilidade de recursos, como dispositivos gráficos de

alta resolução e máquinas com grande velocidade de processamento, entende-se que estas aplicações, em sua maioria, exploram ao máximo a interface do sistema através da representação gráfica de seus dados espaciais. Então, alguns requisitos são essenciais aos SGBDs, como:

- uma linguagem de consulta capaz de permitir a definição e a visualização de consultas, tanto através de predicados espaciais e não-espaciais como também da combinação destes através de forma gráfica e/ou textual.

- mecanismos gráficos para o usuário criar e atualizar seus dados espaciais.

- mecanismos para, a partir dos dados espaciais representados por meio de gráficos, permitir ao usuário a abstração, a especialização e a navegação de determinadas áreas de interesse.

Além dessas facilidades de comunicação entre o usuário e o sistema, este nível de interface deve possuir, ainda, o compilador da linguagem de consulta, que tem por finalidade validar uma consulta especificada pelo usuário e transformá-la em uma estrutura apropriada, a ser avaliada pelo otimizador de consulta. Desta forma, o compilador de consultas também deve ser estendido para incorporar os predicados espaciais existentes nas consultas.

1.2.3 Estratégias para Construção de SGBDs Espaciais

Anteriormente, afirmou-se a impossibilidade dos SGBDs convencionais fornecerem suporte às aplicações espaciais, apresentando, inclusive, na seção anterior, alguns dos requerimentos que se fazem necessários para que determinado SGBD possa prover suporte a estas aplicações. Todavia, em teoria, os SGBDs convencionais têm condições de suportar tais aplicações. Considere, por exemplo, um SGBD relacional e suas respectivas tabelas apresentadas nas figuras 1.3, 1.4, 1.5 e 1.6.

Neste exemplo, a tabela CIDADE representa as informações necessárias de uma dada aplicação geográfica. A tabela REGIÃO tem como objetivo

CIDADE				
id	nome	população	área	região
c1	Campinas	1.200	400	r1
...				

Figura 1.3: Tabela que representa a relação denominada CIDADE composta dos atributos id (identificação), nome, população (representada em milhares de habitantes), área (representada em Km^2) e região (que, para simplificação, é representada por um polígono quadrático que abstrai geograficamente a área referente a essa cidade).

REGIÃO				
id	Lx	Ly	Hx	Hy
r1	10	15	24	25
r2	7	11	30	40
r3	25	48	37	50
...				

Figura 1.4: Tabela que representa a relação denominada REGIÃO composta dos atributos id (identificação), Lx, Ly, Hx e Hy representando a coordenada inferior esquerda e superior direita do polígono através de (Lx, Ly) e (Hx, Hy) respectivamente.

R-TREE		
id	tipo-nó	subnó
rt1	não-folha	sn1
rt2	folha	sn2
...		

Figura 1.5: Tabela que representa a relação denominada R-TREE composta dos atributos id (identificação), tipo-nó (que especifica o tipo do nó da estrutura R-Tree) e subnó (que representa as demais informações do nó).

SUBNÓ		
id	região	rtree-cidade
sn1	r2	rt2
sn1	r3	rt3
sn2	r1	c1
...		

Figura 1.6: Tabela que representa a relação denominada SUBNÓ composta dos atributos id (identificação), região (referente à tabela REGIÃO) e rtree-cidade (referente às tabelas RTREE e CIDADE).

abstrair a disposição da cidade no mapa através de um retângulo bidimensional que consegue contê-la, sendo Lx e Ly as coordenadas inferiores do retângulo no eixo X e Y, respectivamente, e Hx e Hy as coordenadas superiores. As tabelas R-TREE e SUBNÓ representam o método de acesso (a ser apresentado no próximo capítulo) que serve como índice de acesso a tabela CIDADE através do atributo *região*.

Os algoritmos para detectar intersecção e demais relacionamentos espaciais são escritos em uma linguagem de programação de propósito genérico que permite o encaixamento dos comandos de recuperação e armazenamento, fornecidos pelo sistema, para acesso ao Banco de Dados (linguagem hospedeira).

Para esta estratégia, o modelo de dados relacional, como o definido por Codd [Cod70], é suficiente. Extensões no modelo de dados para incorporar novos conceitos, como tipos de dados espaciais, não são necessárias nem tampouco extensões na linguagem e no otimizador de consulta. Contudo, o funcionamento do sistema não se faz de forma satisfatória. Para satisfazer a um determinado tipo de consulta espacial através da tabela R-TREE são necessárias repetidas ativações da interface entre a linguagem hospedeira e o SGBD. Para exemplificar, na figura 1.7 é apresentada uma consulta em SQL para determinar os nós filhos de um dado nó da tabela R-TREE (rt1), cujas regiões de seus subnós contenham o retângulo (Lx, Ly, Hx, Hy) representado por (15,20, 20, 25).

Através da consulta especificada na figura 1.7, verifica-se que, na prática, a utilização de SGBDs convencionais para modelagem de aplicações espaciais não é apropriada, pois:

```
SELECT   SUBNÓ.rtree-cidade
FROM     R-TREE, SUBNÓ, REGIÃO
WHERE    R-TREE.id = rt1 and
         R-TREE.subnó = SUBNÓ.id and
         SUBNÓ.região = REGIÃO.id and
         REGIÃO.Lx <= 15 and REGIÃO.Ly <= 20 and
         REGIÃO.Hx >= 20 and REGIÃO.Hy >= 25 and
```

Figura 1.7: Exemplo de consulta na linguagem SQL.

- a representação dos dados espaciais (CIDADE.região) e do método de acesso R-TREE é transformada em diversas relações, dificultando a utilização do Banco de Dados pelos usuários, uma vez que para recuperar os dados é necessário um conhecimento intrínseco da definição do esquema.

- para satisfazer a consulta, que recupera os filhos de um nó sob uma dada condição, foi necessário a execução de três operações de junção (*joins*). Todo este esforço para recuperar um nó pode afetar consideravelmente o desempenho dos algoritmos, já que diversos nós devem ser recuperados para satisfazer a um único predicado espacial.

Desta forma, surgem como soluções alternativas para construção de SGBDs Espaciais a utilização de SGBDs extensíveis ou, como melhor solução, a modificação ou criação de um novo modelo de dados. Embora esta estratégia exija um esforço de implementação maior, espera-se que apresente um melhor desempenho, visto que em seu projeto as questões fundamentais de como encaixar os aspectos espaciais em um modelo de dados foram estudadas e solucionadas.

A estratégia de incorporar os requisitos fundamentais ao suporte das aplicações não-convencionais nos SGBDs já existentes, através de SGBDs extensíveis, pode ser feita em duas formas: através da utilização de estruturas de arquivos de propósito especial para dados espaciais, como no sistema ARC/INFO [Moo85]; ou através de uma forma mais genérica, com a possibilidade de declaração de novos tipos de dados definidos pelo usuário, como também, a definição de novos operadores e mecanismos de acesso sobre esses

dados, como no *POSTGRES* [RS87].

1.3 Requisitos de Dados em SGBDs Espaciais

Como apresentado nas seções anteriores, as características das aplicações espaciais impossibilitam a simples adequação nos SGBDs convencionais. A integração destas aplicações em tais sistemas só tem sido possível com a reformulação dos diversos aspectos envolvidos na arquitetura destes sistemas.

Diversas soluções têm sido propostas para atender cada um dos problemas envolvidos nesta integração, porém, na maioria das vezes, os estudos realizados para avaliar os pontos favoráveis e desfavoráveis destas soluções não têm sido satisfatórios. Assim, dentre a variedade de alternativas possíveis, muitas vezes se desconhece em que condições cada uma delas é apropriada.

O presente trabalho pretende enriquecer esta discussão com uma análise das diversas soluções existentes para integração das aplicações espaciais em SGBDs, restringindo-se aos aspectos de dados. Entende-se que os maiores problemas existentes na integração das aplicações espaciais nos SGBDs residem justamente nos tipos de dados básicos distintos entre os que são necessários por estas aplicações e os que são suportados por estes sistemas. A determinação das condições ideais em que tais tipos de dados podem ser suportados eficientemente deve ser premissa para os demais estudos sobre esta integração.

O suporte de novos tipos de dados em SGBDs necessita da solução de duas classes de problemas:

- Gerenciamento de tipos de dados;
- Acesso aos dados.

O problema de gerenciamento de tipos de dados, não abordado neste trabalho, está relacionado ao estudo dos mecanismos em que os usuários do sistema constróem os tipos de dados apropriados para sua aplicação. Com o objetivo de situar os problemas envolvidos nesta área de pesquisa, deve-se supor uma aplicação cartográfica onde existam como tipos de dados, ruas, avenidas, bairros, lagos e parques florestais de uma cidade. O sistema deve prover os mecanismos necessários para o usuário abastecer o Banco de Dados com estas informações. Neste exemplo, as informações poderiam ser obtidas através da digitalização de um mapa, como exemplificado na figura 1.8, ou de fotografias aéreas seguidas de algum processo para converter, a partir

da imagem gerada, as informações realmente relevantes para o sistema, a extensão e localização destes dados no espaço. As soluções adotadas para uma aplicação específica podem ser impróprias para outras aplicações. Deste modo, os requisitos de dados para aplicações espaciais devem ser estudados a fim de que os SGBDs propostos sejam flexíveis a esta variedade de aplicações.

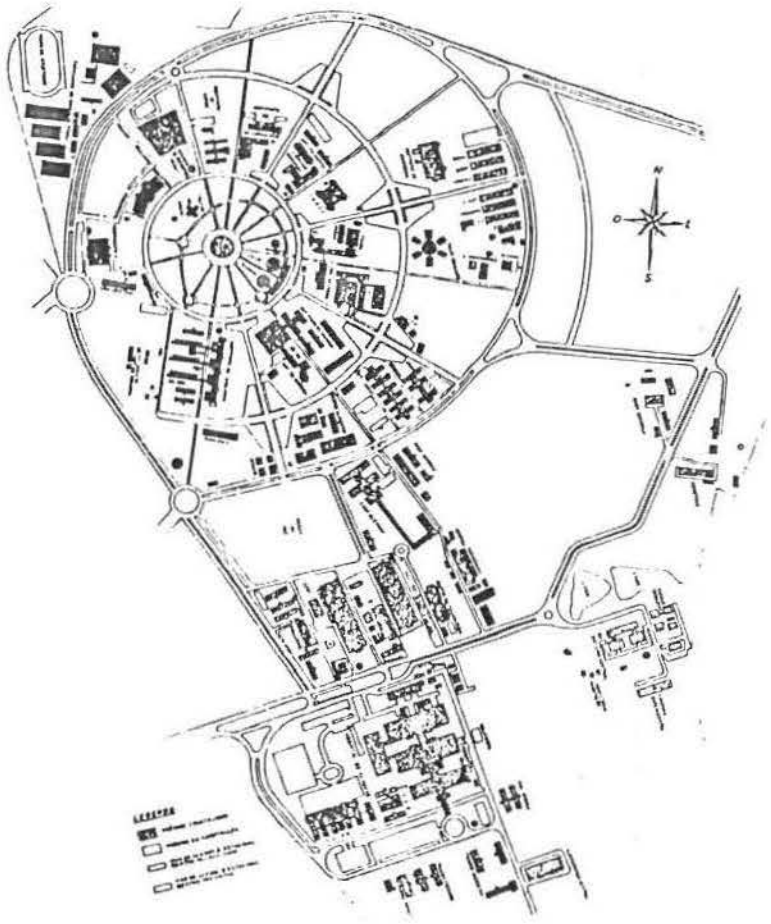


Figura 1.8: Exemplo do levantamento dos dados referentes a uma aplicação geográfica, a partir da digitalização de uma mapa.

A classe de problemas definida como acesso aos dados destina-se a estudar a forma de organização dos dados espaciais. Assim, procura-se definir técnicas de representação destes dados bem como mecanismos para permitir

um rápido acesso a estes (métodos de acesso).

1.3.1 Representação dos Dados

As aplicações convencionais possuem como tipos de dados ou atributos valores numéricos e alfanuméricos. A representação destes tipos de dados nos SGBDs não requer nenhuma estratégia específica dos Sistemas de Banco de Dados, uma vez que fazem parte do conjunto básico fornecido pelas linguagens de programação nas quais os SGBDs são escritos: inteiros, reais, caractere (*char*), cadeias de caracteres (*strings*), dentre alguns outros.

As aplicações espaciais, por sua vez, possuem os tipos de dados que são pontos, linhas e polígonos no espaço. Sendo estes tipos inexistentes nas linguagens de programação, para que os SGBDs possam provê-los, a fim de que sejam utilizados pelas aplicações espaciais como tipos básicos de dados, estes devem possuir um mecanismo de representação desses dados nos tipos mais simples já comentados.

Embora a abstração de dados multidimensionais (dados espaciais) em dados escalares (dados convencionais como inteiros e strings) possa ser efetuada por estratégias distintas, o suporte eficiente às operações sobre estes dados, acesso e recuperação, são fatores decisivos na escolha da representação dos dados espaciais pelos SGBDs.

Exemplificando a influência da estratégia de representação de dados no desempenho de um SGBD Espacial, deve-se supor uma aplicação geográfica com a representação de um mapa através da escala em dezenas de metros. Assim, uma avenida de 1 km de extensão por 10 m de largura pode ser representada pelo conjunto de 100 pontos, onde cada ponto através de sua coordenada X,Y representa 100 m^2. Segundo esta representação, toda vez que for preciso recuperar ou gravar um dado deste tipo, será feita a leitura/gravação de 100 pares de valores numéricos. Além da grande quantidade de espaço necessária para representação de um único dado, este processo de leitura/gravação é extremamente ineficiente. Quanto ao suporte às operações espaciais, esta estratégia em nada facilita a determinação dos relacionamentos espaciais entre os objetos. A determinação de relacionamentos como intersecção e proximidade, por exemplo, ocasionam algoritmos complexos.

Normalmente a representação dos dados espaciais nos SGBDs vai além da simples decomposição do objeto de dados em um conjunto de objetos geométricos mais simples, como no exemplo acima. A necessidade da recuperação destas informações e do suporte às operações com eficiência exigem

a utilização de métodos eficazes para representação dos dados. As aplicações espaciais interferem na escolha dos métodos de representação pelo tipo de dados que utilizam e pelas possíveis operações que são efetuadas sobre estes dados. Até o momento, não foi identificada nenhuma estrutura de dados capaz de representar com eficiência os tipos de dados básicos das aplicações espaciais, pontos, linhas e polígonos no espaço, como também oferecer excelente desempenho para todas as operações espaciais, determinação de intersecção, proximidade, adjacência, dentre outras operações. Desta forma, os SGBDs devem utilizar estruturas diferentes, seja para prover flexibilidade no suporte aos diversos tipos de dados ou para prover melhor desempenho às operações espaciais.

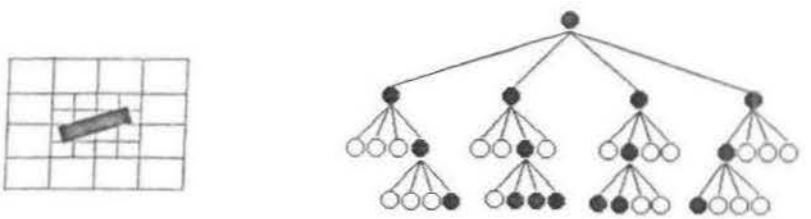
Em [Sam90a] encontra-se um levantamento sobre diversas estruturas de dados a serem utilizadas para representação dos dados espaciais. Dentre estas, estruturas Quadtree [Sam90a, Sam90b, FB74] e suas variantes têm sido amplamente utilizadas em função da flexibilidade e eficiência na representação das mais variadas formas de dados, pontos, áreas, curvas, superfícies, volumes e imagens.

Convém, neste ponto, distinguir as formas de representação dos dados espaciais aqui mencionadas das formas de representação a serem utilizadas pelos métodos de acesso. Pode-se dizer que a ênfase das duas representações são opostas, pois enquanto a representação discutida aqui preocupa-se em representar o objeto de dados através da sua decomposição (especialização), de forma que possa ser representado e manipulado eficientemente sem a perda da precisão, na representação através dos métodos de acesso há preocupação em abstrair (generalizar) os objetos de dados através da representação destes por formas geométricas mais simples. Como exemplo, a figura 1.10 ilustra a representação do dado espacial apresentado na figura 1.9 (obtida a partir da figura 1.8), sob estes dois enfoques. Em 1.10a e 1.10b apresenta-se o objeto de dados através de uma possível representação interna utilizada pelo SGBD, Quadtrees, enquanto na figura 1.10c apresenta-se a abstração deste objeto pelo menor retângulo que consegue contê-lo, estratégia utilizada por alguns métodos de acesso, como o método R-Tree.

É óbvio que esta generalização, causada pela estratégia de representação de alguns métodos de acesso, pode implicar numa perda da precisão do objeto quanto às suas características espaciais, extensão e localização no espaço. Contudo esta perda pode ser identificada e contornada pela aplicação. Na figura 1.10c verifica-se que certa porção do espaço, contida dentro do retângulo de representação do objeto de dados, não faz referência ao dado em si. Este espaço, denominado *dead space*, pode causar a recuperação deste objeto de

(a) (b)

Figura 1.9: Ilustração de um dado espacial (b), através da sua obtenção a partir de uma imagem digitalizada (a).



(a) (b)

(c)

Figura 1.10: Exemplificação da modelagem de uma dado espacial sob dois enfoques: utilizados pelos SGBDs (b) e pelos métodos de acesso (c).

dado através da determinação de algum relacionamento espacial sobre esta área. Das soluções propostas para evitar a exposição deste dado como resposta à consulta, o sistema pode recuperar a representação interna desse dado (figura 1.10b) e então submetê-la ao predicado da consulta com o fim de confirmar ou não a existência do relacionamento espacial sobre o dado.

1.3.2 Métodos de Acesso

Os métodos de acesso, compostos de estruturas de dados e algoritmos de pesquisa para recuperação de dados, são elementos críticos em SGBDs como um dos maiores determinantes do desempenho total do sistema. Os métodos de acesso são projetados como um caminho otimizado aos dados com base em um conjunto definido de predicados sobre os atributos. Como exemplos, podem ser citadas árvores tipo B (*B-Trees*), largamente utilizadas nos sistemas convencionais e construídas para suportar operadores de igualdade (=), inferioridade (<) e superioridade (>) entre seus atributos, e estruturas *hash* para determinação de igualdade. Assim, verifica-se que estes métodos requerem a existência de alguns predicados e de procedimentos, como parte de sua implementação, para determinação destes predicados, e.g. procedimentos para determinação de igualdade, superioridade e inferioridade entre atributos.

As aplicações espaciais possuem, através dos relacionamentos espaciais, relacionamentos entre atributos não existentes entre os dados não-espaciais, tais como vizinhança, intersecção, sobreposição. Os métodos de acesso existentes nos SGBDs convencionais, capazes de processar seleções de dados não-espaciais eficientemente, não são apropriados para executar a avaliação destes predicados espaciais com a mesma eficiência, fazendo com que métodos de acesso específicos sejam utilizados na determinação destes relacionamentos. Esses métodos serão aqui referenciados como **Métodos de Acesso a Dados Espaciais**.

Através de uma avaliação dos métodos de acesso a dados espaciais existentes, percebe-se que os desempenhos estão relacionados a um pequeno conjunto de fatores (tipos dos dados, tamanho, distribuição, dinâmica dos dados e tipos de consultas). Logo, a escolha dos métodos mais apropriados para utilização nos SGBDs Espaciais depende do comportamento das aplicações nas quais o SGBD se propõe a prover suporte, quanto a estes fatores.

Conforme Guenther e Buchmann [GB90], existe escassez de pesquisas que visam determinar, através destes fatores, em que condições cada método de

acesso é mais apropriado. Vários autores [GB90, Gre89, KSSS89] discutem a realização destas pesquisas com a implementação destes métodos e com a comparação do desempenho destes sob diferentes tipos e distribuição de dados e consultas. A justificativa dos estudos não-analíticos sobre o desempenho desses métodos, ao invés dos estudos analíticos, é devido à dificuldade de capturar o funcionamento destes em modelos matemáticos, e de algumas condições relacionadas ao desempenho só serem detectadas com implementações reais (*buffer pool*, dentre outras).

A seguir serão caracterizadas as aplicações espaciais e os métodos de acesso quanto a estes fatores.

Tipos de Dados

As aplicações espaciais possuem como tipos de dados os pontos, linhas e polígonos no espaço. Abstraindo os métodos de acesso, quanto a representação dos dados pode-se classificá-los em: métodos para representação de pontos (como K-d Tree, Quadtree e Grid File) e métodos para representação de objetos de tamanho não-zero (como Spatial K-d Tree, R-Tree e suas variantes e Cell Tree). A classificação, contudo, não é rigorosa, uma vez que existem estratégias para transformação de objetos no espaço em pontos num espaço de dimensionalidade maior [HN83] e por alguns métodos, sob a segunda classificação, também suportarem a representação de pontos. Enfim, alguns métodos de acesso podem suportar os diferentes tipos de dados necessários às aplicações espaciais, mas os mecanismos que provêem tal versatilidade pode degradar o desempenho da estrutura para determinados tipos de operações, como, por exemplo, consultas.

Quanto às aplicações espaciais, algumas destas podem manipular com os diferentes tipos de dados ou até mesmo necessitar da representação de um objeto com diferentes níveis de abstração, exigindo distintas técnicas de representação. Para exemplificar tal aplicação, pode-se imaginar uma aplicação geográfica onde o mapa de um país seria representado por pontos (cidades) e linhas (as estradas). Para suportar a determinação de relacionamentos espaciais entre estes dados, haveria métodos de acesso. A aplicação poderia ainda especializar um ponto, cidade, recuperando todos os objetos que a compõe (ruas, avenidas, bairros, etc.) e provendo também a detecção de relacionamentos entre estes objetos. Nesse caso, a cidade não seria mais representada por um ponto no espaço e sim pela coleção dos diversos objetos que a compõe.

Tamanho

Greene [Gre89] realizou experimentos comparativos sobre os métodos de acesso R-Tree e R^+-Tree e encontrou uma relação de desempenho destes métodos referente à dimensão dos objetos que representam. Por Greene, a R^+-Tree apresenta um melhor desempenho quando seus objetos de dados, além de não se sobreporem, possuem um tamanho pequeno quando comparado ao do espaço total representado pela aplicação. A R-Tree obtém um desempenho melhor nas condições contrárias, com sobreposição de objetos e área individual maior em relação ao espaço. Sellis [SRF87] também realizou experimentos comparativos entre esses dois métodos e concluiu na superioridade da R^+-Tree na representação de numerosas quantidades de objetos grandes.

De acordo com esses resultados, verifica-se a possibilidade de variação do desempenho dos métodos de acesso sob distribuição de dados com diferentes tamanhos.

Esta característica dá indício da necessidade de avaliação das aplicações espaciais quanto ao tamanho dos objetos de dados que estas representam. As aplicações cartográficas e geográficas, por exemplo, são caracterizadas por pequenas quantidades de objetos grandes (ruas, bairros, avenidas, etc.), enquanto que as aplicações VLSI/CAD, por grande quantidade de pequenos objetos (pequenos retângulos comparados ao espaço representado por um circuito integrado).

Distribuição

Alguns métodos de acesso baseados em árvores, como K-d Tree e Quadtree, não se comprometem em gerar estruturas balanceadas[4], possibilitando que a profundidade da raiz às folhas possuam diversos comprimentos. Embora os estudos realizados nessas estruturas mostrem que, sob distribuições uniformes de dados, a busca de elementos demanda tempo médio proporcional a logarítmico ($O(\log N)$), determinadas distribuições podem ocasionar tempo linear ($O(N)$) altamente indesejável para o gerenciamento de grande quantidade de dados armazenados no disco.

Por outro lado, estruturas balanceadas, como R-Tree e R^+-Tree, também são dependentes, em uma escala bem menor, da distribuição dos dados. Nos estudos comentados no tópico anterior, Greene concluiu a superioridade da

[4] uma árvore é dita balanceada quando a profundidade da subárvore esquerda de cada nó nunca difere de ± 1 da profundidade de sua subárvore direita.

R-Tree ao invés da R^+-Tree, para objetos de dados de tamanho grande. Sellis, por sua vez, apresentou resultados contrários sob as mesmas condições de dados. A justificativa desses resultados antagônicos pode ser explicada pela característica inserida nos experimentos de Greene, que, pela sobreposição dos dados, podem ter gerado distribuições distintas.

Dinâmica dos Dados

Quanto à dinâmica dos dados, alguns métodos de acesso são classificados como dinâmicos[5] (R-Tree, R^+-Tree e Cell Tree) e, outros, classificados como estáticos[6] (K-d Tree e Quadtree). Os métodos de acesso dinâmicos são capazes de suportar a intercalação de remoções e inserções de dados sem a necessidade da reorganização da estrutura de dados. Os métodos estáticos, por sua vez, necessitam da reorganização periódica da estrutura, pois as inserções e remoções podem degradar o seu desempenho. Contudo, as rotinas de reorganização normalmente são impróprias em conseqüência da periodicidade com que são executadas e do alto custo da reorganização da estrutura armazenada no disco.

Os SGBDs convencionais utilizam métodos de acesso dinâmicos, uma vez que as aplicações convencionais, via-de-regra, caracterizam-se pelo grande volume da atualização dos dados. As aplicações espaciais, em contrapartida, podem requisitar um menor volume de atualizações, permitindo, assim, a utilização de métodos estáticos nos SGBDs. Mesmo que a utilização de tais métodos vá de encontro à proposta de flexibilidade dos SGBDs, em virtude das aplicações com grande atualização dos dados, estes métodos podem ser usados em conjunto com outros métodos dinâmicos, desde que o seu desempenho sobre as demais operações executadas pela aplicação (i.e. consultas) justifique a sua utilização.

Como exemplo de aplicações espaciais estáticas, pode-se comentar as aplicações geográficas e cartográficas de gerenciamento urbano (telefonia, saneamento e luz). Estas aplicações possuem como dados o mapa da cidade integrado ao mapa da rede de telefonia, saneamento e luz. Os dados refe-

[5] o termo dinâmico quando atribuído a métodos de acesso, estrutura de índices ou de dados, muito utilizado na literatura, refere-se à capacidade destes em intercalar operações de inserção, remoção e consulta sem que nenhuma reorganização periódica seja requerida.

[6] o termo método de acesso (estrutura de índices ou de dados) estático está sendo empregado nesse trabalho como antônimo do termo dinâmico. Dessa forma, refere-se a métodos de acesso que necessitam de rotinas de reorganização, uma vez que as operações de inserção e remoção podem degenerar a estrutura degradando o desempenho de suas operações básicas (inserção, remoção e consulta).

rentes ao mapa da cidade, i.e., as ruas, avenidas e bairros, possuem pouca dinâmica, ou seja, a inserção e remoção destes elementos de dados possuem uma periodicidade muito pequena, podendo justificar, desta forma, a utilização de um método de acesso estático. Os dados referentes à rede possuem uma dinâmica um pouco maior, dado que tais aplicações visam proceder o gerenciamento urbano através de expansões e melhorias nos seus serviços, provocando diversas alterações na topologia da rede existente. Para estes dados, os métodos de acesso dinâmicos são mais apropriados.

Outro ponto a ser discutido sobre a dinâmica dos dados é que não somente as inserções e remoções de dados causam atualizações na estrutura de dados referente ao método de acesso. A alteração de um objeto já existente através da mudança de quaisquer de suas características espaciais, extensão e localização no espaço deve causar uma remoção, atualização e inserção deste objeto na estrutura de dados. Algumas aplicações espaciais como CAD/CAM (projeto e fabricação auxiliados por computador) não se caracterizam pela inserção e remoção de grande quantidade de dados, porém, a atualização sobre os objetos de dados existentes é muito grande.

Tipos de Consultas

As consultas efetuadas nos SGBDs Espaciais podem ser divididas em duas categorias:

- *point queries* – que visam determinar os objetos que contêm um dado ponto no espaço.

- *range queries* – que visam determinar todos os objetos que satisfazem a um determinado relacionamento sobre uma área de pesquisa (*window*). Estes relacionamentos podem ser: intersecção (todos os objetos que possuem pontos em comum com a *window*); inclusão (todos os objetos que contêm a *window*); e não-inclusão[7] (todos os objetos contidos na *window*).

As aplicações requisitam os diferentes tipos possíveis de consultas na base de dados com diferentes freqüências. Como em alguns métodos de acesso a dados não-espaciais, os métodos de acesso a dados espaciais não suportam eficientemente todos os possíveis tipos de consultas. Dessa forma, surge

[7] esse termo será utilizado na ausência de uma melhor tradução para a palavra *within*, em inglês.

a possibilidade da utilização de diversos métodos de acesso, por parte dos SGBDs, com o propósito de dar flexibilidade às aplicações espaciais.

1.4 Sinopse da Dissertação

No próximo capítulo é apresentado um *survey* sobre os métodos de acesso a dados espaciais existentes. Buscou-se, no levantamento desses métodos, identificar os mais citados na literatura bem como os adotados nos SGBDs Espaciais existentes. Este último objetivo não foi alcançado com muito sucesso, já que identificou-se que a maioria dos sistemas, propostos como SGBDs Espaciais, utilizam-se de mecanismos alternativos para permitir um rápido acesso às informações, como por meio da utilização de diversos arquivos. A justificativa dessas alternativas não está relacionada com sua melhor eficiência, comparada à utilização de métodos de acesso, mas sim ao estágio inicial em que se encontram as pesquisas nessa área. Os métodos de acesso serão apresentados por meio da descrição do seu funcionamento, das metodologias para inserção, remoção e consulta, com o fim de se identificar, através dessas operações, as melhores relações de desempenho, flexibilidade e simplicidade de cada método de acesso.

Foram implementados os métodos identificados como mais apropriados, de acordo com as melhores relações obtidas no capítulo 2. O capítulo 3 apresenta a justificativa da escolha desses métodos e descreve as suas implementações, dando-se ênfase às divergências aqui existentes, em comparação com as implementações propostas nos artigos originais. Em decorrência dos algoritmos propostos não serem completos, sentiu-se necessidade, durante a implementação desses métodos, da adoção de particularidades.

O capítulo 4 discute a necessidade da realização de experimentos, com a finalidade de enriquecer a análise dos métodos de acesso. Avalia as estratégias adotadas para realização dos experimentos, por alguns autores, e apresenta uma proposta alternativa para sua realização que permite, por meio dos seus resultados, uma análise mais completa do que as obtidos pelos demais experimentos.

Por fim, o capítulo 5 conclui a dissertação, apresentando as contribuições e sugestões para futuros trabalhos.

Capítulo 2

Métodos de Acesso a Dados Espaciais

2.1 Taxonomia

Na literatura existem diversos métodos de acesso a dados espaciais. Alguns autores [Ooi90, SRF87, KSSS89] sugerem uma classificação para estes métodos de acordo com os objetos de dados que estes representam. Logo, os métodos de acesso são classificados como *Métodos de Acesso a Pontos*, para objetos que são pontos no espaço, e *Métodos de Acesso a Objetos de Tamanho Não-Zero*, para representar as demais formas de dados espaciais.

2.1.1 Métodos de Acesso a Pontos

Os métodos de acesso a pontos foram estudados no passado com o objetivo de solucionar problemas relacionados à recuperação de registros através de consultas com múltiplas chaves. Esta idéia foi facilmente estendida para o gerenciamento de pontos no espaço pela simples associação dos K atributos chaves do registro para os valores escalares em cada uma das dimensões do espaço K-dimensional. Estes métodos representam seus objetos de dados, pontos no espaço K-dimensional, através de uma K-tupla $x = (x_1,\ldots,x_K)$, onde x_i representa a distância do ponto à origem do eixo de coordenadas na dimensão i.

Alguns autores propuseram uma classificação desses métodos de acesso na definição das características comuns neles existentes. O objetivo dessa classificação foi auxiliar a escolha de um método de acesso específico para

uma determinada aplicação de acordo com estas características. Foi identificado que o princípio básico de todos os métodos de acesso a pontos reside na partição do espaço representado pelo método de acesso (espaço de dados) em sub-regiões, de forma que todos os registros em uma sub-região sejam armazenados em uma mesma página de dados (no caso de gerenciamento de dados em disco, estas páginas de dados são as próprias páginas de disco). Sellis [SRF87] propõe a classificação dos métodos de acesso de acordo com a metodologia utilizada para partição do espaço, enquanto Kriegel [KSSS89] sugere a classificação de acordo com três propriedades dessas sub-regiões.

Por Sellis, a operação de partição pode ser classificada em três atributos:

- posição – que define a posição em que a região é cortada. Essa posição pode ser predeterminada, fixa (Grid File [NHS84]), ou determinada pelos objetos de dados, adaptável (K-d Tree [Ben75]). A estrutura Point Quadtree [FB74] permite os tipos de divisão, contudo, normalmente é empregada a estratégia adaptável.

- dimensão – que define as dimensões em que é feita a partição, podendo ser através de um único hiperplano, partição em uma dimensão (K-d Tree) ou através de todas as dimensões, por K-hiperplanos (Point Quadtree).

- localização – que define a abrangência da partição. Somente na região afetada (Point Quadtree e K-d Tree) ou em todas as regiões em uma dada direção (Grid File).

Kriegel, por sua vez, define outros atributos:

- se as sub-regiões são disjuntas duas a duas (Grid File, Bang File [Fre87] e Buddy Hash Tree [SK90]) ou não (Twin Grid File [HSW88]).

- se as sub-regiões são retangulares (Grid File e Buddy Hash Tree) ou não (Bang File).

- se a união de todas as sub-regiões refere-se ao espaço de dados total (Bang File e Grid File) ou não (Buddy Hash Tree).

2.1.2 Métodos de Acesso a Objetos de Tamanho Não-Zero

Os métodos de acesso a objetos de tamanho não-zero[1] surgiram recentemente, dada a impossibilidade dos métodos de acesso a pontos na representação de outros objetos de dados espaciais, como linhas e regiões no espaço. Muitas aplicações espaciais possuem objetos de dados com formas geométricas imprecisas ou não predefinidas (representação de ruas, lagos e estradas em aplicações geográficas ou cartográficas). A determinação dos relacionamentos espaciais sobre estas formas ocasionam algoritmos complexos e caros. Assim, surgem estratégias a serem utilizadas para representar esses objetos de formas irregulares por outros de formas mais simples. Duas estratégias podem ser utilizadas para simplificar a representação dos objetos espaciais de formas irregulares, *region decomposition* e *minimum bounding rectangle* (MBR).

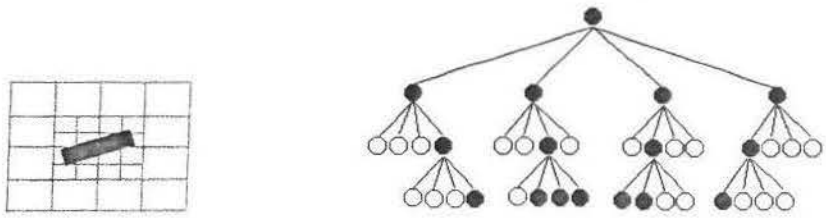
A estratégia *region decomposition* é caracterizada pela precisão em que os objetos de dados são representados. A idéia básica reside na decomposição da região associada ao objeto de dados em um conjunto de objetos de formas mais simples. Duas metodologias podem ser destacadas: a decomposição em objetos retangulares disjuntos e a decomposição em poliedros convexos regulares. A primeira metodologia é mais difundida e baseia-se na decomposição recursiva do objeto original em objetos retangulares disjuntos até uma desejada resolução (figura 2.1). Essa metodologia é melhor compreendida através de sua estrutura de dados associada, quadtree [Sam84]. A estrutura de dados utilizada para gerenciar os objetos de dados (figura 2.2), apesar de simples, só pode representar um único objeto, não podendo, então, ser utilizada como métodos de acesso. Essa metodologia é apropriada para o processamento de imagens, como também para a representação de objetos espaciais por parte dos SGBDs.

A segunda metodologia baseia-se na decomposição de poliedros em um conjunto de poliedros mais simples. Dessa forma, os objetos espaciais irregulares são primeiro representados por poliedros para depois serem decompostos (figura 2.3). Nessa metodologia, cada objeto de dados é representado como *convex polyhedral chain* [GW87], i.e., como a soma de poliedros p_i convexos também denominados de *cells* (células). Formalmente cada objeto é representado como *polyhedral chain* (cadeia de poliedros) d-dimensional no espaço Euclidiano E^d da seguinte forma:

[1] será utilizada essa nomenclatura ao invés de métodos de acesso a objetos espaciais, para não prover dúvidas quanto às demais nomenclaturas de dados espaciais e relacionamentos espaciais.

(a) (b) (c)

Figura 2.1: Exemplo da decomposição da imagem apresentada em (b) por objetos retangulares (c).



(a) (b)

Figura 2.2: Ilustração da estrutura Quadtree (b) para representação do objeto em (a).

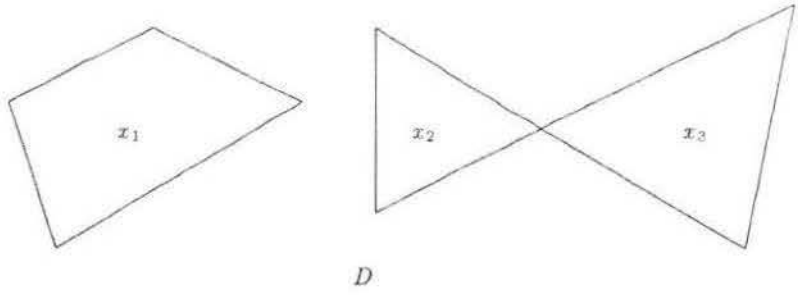
$$x_p = \sum_{i=1}^{m} p_i$$

onde p_i são poliedros d-dimensionais regulares não necessariamente *bounded*.

(a) (b) (c)

Figura 2.3: Exemplo da representação do objeto apresentado em (b) por poliedros (c).

Polyhedral Chains são ferramentas simples e poderosas para descrever vários tipos de objetos poliédricos. Na figura 2.4 é ilustrada a representação de objetos simples (que não se interceptam).



D

$$x_D = x_1 + x_2 + x_3$$

Figura 2.4: Exemplo da representação de um objeto (D) através da soma de poliedros regulares.

A representação ou decomposição do objeto de dados nas *convex cells*

(células) é feita através da intersecção de *halfspaces*[2] (subespaços) codificado em um vetor. Cada subespaço pode ser representado por um produto $h.H$ onde:

- H é um hiperplano $(d\text{-}1)$-dimensional orientado;
- h é um número inteiro.

O produto $h.H$ é definido como $1.H$ (que indica o subespaço próximo à direita do hiperplano H), $-1.H$ (que indica o subespaço próximo à esquerda do hiperplano H) e $0.H$ (que indica a ausência do hiperplano H ou o próprio espaço E^d). Dessa forma, seja $H = H_1 H_2 ... H_{|H|}$ a lista de hiperplanos $(d\text{-}1)$-dimensionais e f a face de qualquer objeto de dados no base de dados, existe um hiperplano H que faz adjacência a f.

Cada célula p pode então ser representada por um vetor ternário $h_p = 0,1,-1^{|H|}$, tal que $p = \bigcap_{i=1}^{|H|} (h_p)_i.H_i$. Na figura 2.5 é apresentada a representação de um objeto de dados p no espaço bidimensional denotado por cinco hiperplanos.

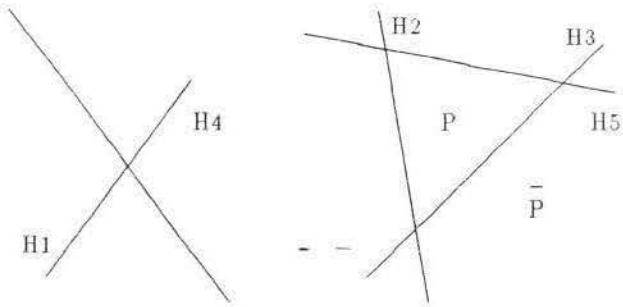


Figura 2.5: Exemplo da representação do objeto P pela estratégia *region decomposition*.

A inserção de objetos de dados na base de dados pode causar a inserção de hiperplanos se nenhum dos hiperplanos existentes faz adjacência ao novo

[2]a tradução mais apropriada para o termo em inglês halfspace é subespaço. Nesse contexto, cada halfspace através de um hiperplano $(d\text{-}1)$-dimensional divide o espaço d-dimensional em dois subespaços.

objeto de dados. De maneira análoga, a remoção de objetos pode tornar hiperplanos redundantes, causando a sua remoção.

Uma característica interessante dessa metodologia é que a noção de vértices é completamente abandonada. Todas as operações sobre as células, como determinação de união e intersecção por exemplo, são calculadas eficientemente pela decomposição dessas operações em duas partes: através das operações nos vetores (h_p) sem referência para as coordenadas geométricas dos hiperplanos e através de uma operação geométrica que testa quando um vetor é nulo, i.e., quando as intersecções dos subespaços especificadas por h_p são vazias [GW87, Gun88].

A outra estratégia utilizada para representar objetos de formas irregulares, ao contrário da *region decomposition*, é caracterizada pela simplicidade da representação do objeto de dados por um único objeto de forma geométrica mais simples. Essa estratégia, MBR, representa cada objeto de dados pelo menor retângulo K-dimensional que consegue contê-lo em sua totalidade (figura 2.6).

(a) (b) (c)

Figura 2.6: Exemplo da representação do objeto apresentado em (b) através da estratégia MBR (c).

Dentre as estratégias para representação de objetos espaciais de formas irregulares, *region decomposition* e MBR, a estratégia MBR tem sido largamente utilizada por diversos métodos de acesso: R-Tree, R^+-Tree, R^*-Tree, Spatial K-d Tree, Balanced Quad List Quad Tree, MX-Quadtree. A justificativa para a grande utilização dessa estratégia é devido:

- à pouca quantidade de espaço necessária para representar qualquer objeto espacial. Nessa representação, um objeto no espaço K-dimensional pode ser representado por 2 K-tuplas, $(a_1,...,a_K)$ e $(b_1,...,b_K)$, onde $[a_i,b_i]$ representa o intervalo do retângulo K-dimensional na dimensão

i.

- à simplicidade e eficiência na determinação de relacionamentos espaciais entre esses objetos, MBRs.
- à característica principal dos objetos espaciais ainda ser preservada após a abstração da representação, i.e., localização e extensão dos objetos em cada uma das dimensões.

O ponto desfavorável nessa estratégia é que os objetos dispostos diagonalmente no espaço são representados com uma perda significativa de sua precisão. O espaço representado pelo MBR vai além do espaço associado ao objeto. Essa área representada em excesso, *dead space*, pode causar a recuperação desse objeto se algum relacionamento espacial sobre essa área for satisfeito. Todavia, essa característica não chega a ser um problema, pois na maioria das aplicações espaciais a indicação da existência de um relacionamento já satisfaz a consulta. Nas demais aplicações onde se faz necessário maior precisão, são utilizados os limites exatos dos objetos que estão armazenados nos SGBDs, para confirmação do subconjunto obtido na detecção dos relacionamentos entre os MBRs. A figura 2.7 ilustra a representação de dois objetos de dados de formas irregulares (A e B) em retângulos. Nessa ilustração é exemplificada a perda da precisão na representação dos objetos, pois, embora os MBRs sejam adjacentes, os objetos que estes representam não o são.

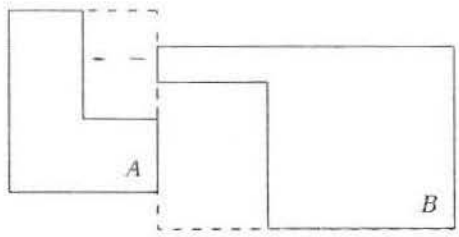


Figura 2.7: Ilustração da perda da precisão na representação de objetos através da estratégia MBR.

A perda da precisão da estratégia MBR para objetos dispostos diagonalmente no espaço, por sua vez, é muito menos acentuada na estratégia *region decomposition*. Conforme já apresentado, esta estratégia é caracterizada pela

maior precisão entre a representação resultante e a representação real do objeto de dados. Como resultado, destacando a metodologia de decomposição em poliedros regulares, esta estratégia permite uma representação de objetos espaciais complexos e uma maior precisão nas respostas das consultas. Em contrapartida, o processo de decomposição do objeto requer certa complexidade. Sob essa metodologia de representação, conhece-se apenas o método de acesso Cell Tree.

Da mesma forma que nos métodos de acesso a pontos, classificações são propostas para a caracterização dos diversos métodos de acesso a objetos de tamanho não-zero existentes. Essas classificações, restritas aos métodos que utilizam a estratégia MBR, têm se baseado na técnica especial que permite a extensão de um método de acesso a pontos para um método de acesso a objetos de tamanho não-zero. Então o desempenho desses métodos de acesso é dependente do método de acesso a pontos utilizado e da técnica aplicada. Kriegel [KSSS89] descreve estas técnicas como:

- *Clipping* – assume a partição do espaço de dados em regiões disjuntas. A inserção dos retângulos é feita de forma semelhante à inserção de pontos. Contudo, quando um retângulo intercepta mais de uma região, este é particionado em um conjunto de retângulos, de forma que cada retângulo desse conjunto intercepte somente uma única região (R^+-Tree [SRF87]).

- *Overlapping* – diferente da estratégia de *clipping*, permite que as sub-regiões que dividem o espaço de dados se sobreponham. A vantagem desta estatégia sobre a anterior é que a utilização do espaço depende somente do método de acesso a pontos utilizado, já que cada retângulo é unicamente representado no arquivo (R-Tree [Gut84]).

- *Transformation* – representa cada MBR por um ponto no espaço de dimensionalidade maior. Logo, os métodos de acesso utilizados são os próprios métodos de acesso a pontos (Grid File, como proposto em [HN83]).

Sellis [SRF87] também descreve três técnicas, mas com características diferentes: *transformation*, com idéia semelhante à apresentada por [KSSS89]; *overlapping* e *clipping*, onde o espaço de dados é dividido em sub-regiões que podem ser sobrepostas ou não (semelhante a *clipping* e *overlapping* proposta por [KSSS89]; e *space filling curves*, que possui como idéia principal a transformação de um objeto K-dimensional em segmentos de linhas.

2.2 Classificação Proposta

Para apresentar os métodos de acesso aqui estudados não será empregada a classificação dos métodos de acesso recém discutida, métodos de acesso a pontos e métodos de acesso a objetos de tamanho não-zero. Dois pontos desfavoráveis nessa classificação levam a propor uma classificação mais apropriada ao propósito deste projeto. Em primeiro lugar, com rigor, a classificação acima não existe. Tanto os métodos de acesso a pontos podem representar objetos espaciais de formas irregulares, através da abstração destes em MBRs e da posterior transformação destes MBRs em pontos num espaço de dimensionalidade maior, como também os diversos métodos de acesso a objetos de tamanho não-zero podem suportar a representação de pontos no espaço através da abstração destes pontos em MBRs de extensão nula em todas as dimensões. Nesse último caso, pode ser citado os experimentos realizados por Beckmann [BKSS90] sobre o desempenho de um método de acesso a objetos de tamanho não-zero comparado ao desempenho de alguns métodos de acesso a pontos. Nesses experimentos, para objetos de dados que são pontos, foi detectada a superioridade do primeiro método sobre os demais.

O segundo ponto desfavorável é que essa classificação ajuda em muito pouco o objetivo do trabalho, na identificação dos métodos de acesso mais apropriados para serem utilizados em SGBDs. Foi identificado que as estruturas de dados utilizadas, tanto nos métodos de acesso a pontos como nos métodos de acesso a objeto de tamanho não-zero, são extensões das estruturas de dados utilizadas para o gerenciamento de dados escalares (unidimensionais). Muitas das características dessas estruturas de dados básicas, quanto aos requisitos necessários por parte dos SGBDs (gerenciamento de dados em memória secundária, suporte a atualização de dados, dentre outras), além de serem conhecidas são preservadas nas suas estruturas de dados estendidas. Isto permite, em uma primeira instância, determinar os métodos de acesso mais apropriados a serem utilizados nos SGBDs, através das relações de desempenho, flexibilidade e simplicidades conhecidas desses métodos bases. Assim, foi proposto, neste trabalho, a apresentação dos métodos de acesso a dados espaciais como derivados de três distintos grupos de estruturas de dados: árvores binárias, estruturas *hash* e *multiway trees* ou árvores multi-árias.

Antes porém de prosseguir com a apresentação das estruturas, é apresentado nas figuras 2.8 e 2.9 o conjunto de objetos de dados a serem utilizados pelos métodos de acesso a pontos e a objetos de tamanho não-zero, respec-

Bairro	X	Y
Cambuí	64	57
V. Georgina	71	34
J. das Palmeiras	96	52
Centro	57	51
V. Aeroporto	5	9
J. Sta. Genebra	50	90
J. Nilópolis	75	92
J. Eulina	16	61
Ponte Preta	62	46

Figura 2.8: Lista seqüencial de alguns bairros da cidade de Campinas

tivamente. Parte-se do pressuposto que a representação dos dados é feita no espaço bidimensional, com o objetivo de simplificar as ilustrações, e afirma-se que a correspondência entre as coordenadas e o nome dos bairros da cidade de Campinas não está geograficamente correta.

Ident.	Bairro	X_l	X_r	Y_l	Y_r
A	Cambuí	57	78	58	62
B	V. Georgina	71	85	28	36
C	J. das Palmeiras	92	100	55	62
D	Centro	50	64	47	58
E	V. Aeroporto	00	14	07	15
F	J. Sta. Genebra	42	53	86	94
G	J. Nilópolis	71	78	89	93
H	J. Eulina	14	25	58	67
I	Ponte Preta	60	67	42	47

Figura 2.9: Lista seqüencial de alguns bairros da cidade de Campinas. X_l e X_r correspondem as coordenadas esquerda e direita do eixo X, enquanto Y_l e Y_r correspondem as coordenadas inferior e superior do eixo Y.

2.3 Derivados de Árvores Binárias

Os primeiros métodos de acesso a dados espaciais surgiram através de extensões de árvores binárias. As estruturas de dados utilizadas nesses métodos de acesso, em sua idéia básica, estenderam o funcionamento da árvore binária de pesquisa [Knu73b] para n dimensões.

Entretanto, alguns autores não consideram tais métodos como métodos de acesso a dados espaciais, em conseqüência das características básicas das árvores binárias, serem estáticas e de baixo *fan-out*[3] por nó, a tornarem impróprias para o gerenciamento de dados em disco [SRF87, KSSS89, FSRM]. Nas estruturas estáticas, a intercalação de operações de inserção e remoção pode degradar o desempenho da estrutura exigindo a execução de rotinas para sua reorganização. Knuth [Knu73b], por exemplo, apresenta que o desempenho de consultas nesses casos pode ser degradado de logarítmico (em caso de árvores binárias ótimas) a linear. As rotinas de reorganização, por sua vez, são custosas. Como a estrutura de dados está armazenada em disco, a sua reorganização requer diversos acessos a disco, o que interrompe todas as aplicações que utilizam esse método.

Conforme [SRF87], outra característica é que árvores com baixo *fan-out* não são facilmente estendidas para gerenciar dados em armazenamento secundário. A razão disso é que as páginas de disco normalmente podem conter a ordem de 50 apontadores, assim, árvores com largos *fan-out* são mais apropriadas uma vez que árvores com dois ou quatro folhas por nó normalmente causam várias *page faults*[4].

Contudo, na literatura existem diversos métodos de acesso derivados de árvores binárias de pesquisa propostos como métodos de acesso a dados espaciais. Dentre estes, dois métodos foram bases para os demais métodos propostos, Point Quadtree [FB74] e K-d Tree [Ben75]. A partir destes, outros métodos surgiram trazendo como inovações artifícios para contornar os problemas que tornam inadequadas as árvores binárias para o gerenciamento de dados em disco.

Quanto ao fato das árvores binárias serem estáticas, Finkel [FB74] e Overmars [Ov82] propuseram estratégias para construção de Point Quadtrees otimizadas[5], enquanto Bentley [Ben75] propôs estratégia semelhante para K-d Tree. Estas estratégias, contudo, exigem que os dados sejam conhe-

[3] fan-out de um nó está associado ao número de apontadores ou filhos desse nó.

[4] requisição de páginas que não se encontram na memória.

[5] por [FB74] uma árvore é otimizada se nenhuma subárvore de um nó A possui mais do que a metade do elementos existentes na árvore A.

cidos a priori, possibilitando assim, a sua utilização apenas em base de dados estáticas. Hsiao [HJ] tenta contornar este problema, por meio de uma rotina para balanceamento[6] estática proposta ao método Quad List Quad Tree (QLQT) [Wd89]. Por Hsiao, esta rotina seria executada com uma certa periodicidade, porém, como já discutido, esta solução é imprópria devido ao alto custo e as demais conseqüências em um ambiente de multi-processamento (a interrupção de todas as aplicações durante a execução desta rotina).

A solução mais interessante, contudo, tem sido sugerida por Ooi [OMSD87, Ooi90] na implementação da Spatial K-d Trees. Para evitar as *page faults*, que normalmente aconteceriam se os nós da estrutura estivessem dispostos em diferentes páginas de disco, Ooi sugere a utilização da estratégia *multiway trees* [Knu73b]. Conforme a figura 2.10, essa estratégia preocupa-se em abstrair as subárvores da estrutura de dados em páginas de disco. Assim, espera-se otimizar o desempenho da estrutura em decorrência da melhor relação dos nós armazenados no disco.

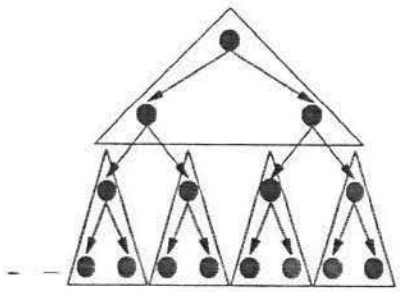


Figura 2.10: Ilustração da estratégia *multiway trees*.

A seguir é apresentado a descrição das estruturas Point Quadtree e K-d Tree (para representação de pontos no espaço) e de suas variantes MX-CIF Quadtree e Spatial K-d Tree (para representação de objetos espaciais de tamanho não-zero).

[6] uma árvore é dita balanceada quando a profundidade da subárvore esquerda de cada nó nunca difere de ± 1 da profundidade de sua subárvore direita.

2.3.1 Point Quadtree

A Point Quadtree, por Finkel e Bentley [FB74], é uma estrutura de dados hierárquica cuja propriedade é baseada no princípio de decomposição recursiva do espaço.

A estrutura possui as seguintes características:

- representada por uma árvore onde cada nó tem grau[7] 2^k, onde k é a dimensão em que os objetos de dados são representados. O termo Quadtree é generalizado para representação de dados no espaço bidimensional, tendo assim grau igual a 4.
- cada nó representa um único objeto de dado (ponto).
- a decomposição do espaço pode ser feita em partes iguais ou governada pelos dados de entrada, este último caso mais comum para Point Quadtree.
- dois objetos de dados não ocupam o mesmo espaço (não possuem as mesmas coordenadas).

Assim observa-se que cada nó em uma Point Quadtree (bidimensional) possui sete campos:

- 4 apontadores para seus respectivos filhos, denominados como quadrantes e identificados geograficamente como: Noroeste (subárvore 1), Nordeste (subárvore 2), Sudoeste (subárvore 3), Sudeste (subárvore 4).
- 2 campos referente as coordenadas x e y.
- 1 campo contendo a identificação desse nó, ou em nosso contexto, um apontador para o registro associado com esse dado.

Na figura 2.11 é apresentada uma Point Quadtree correspondente aos dados da figura 2.8.

Inserção

Inserções em Point Quadtree são semelhantes às inserções feitas em árvores binárias de pesquisa. Dessa forma, o primeiro elemento inserido será denominado como raiz da árvore e dividirá o espaço em quatro subespaços ou

[7] número de subárvores por nó.

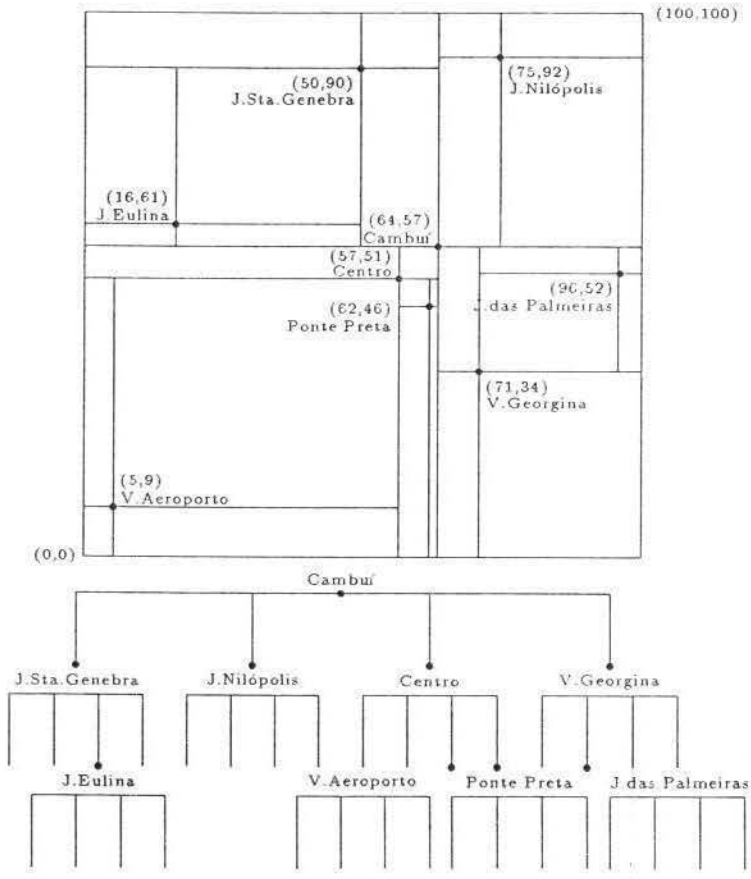


Figura 2.11: Uma Point Quadtree e os registros que esta representa.

quadrantes. A partir daí, este elemento já inserido irá direcionar todos os demais percursos na estrutura através de seus quadrantes.

Cada novo elemento a ser inserido é armazenado nas folhas da estrutura (como na árvore binária de pesquisa). Para isso, inicialmente, a estrutura é percorrida de acordo com a posição relativa do elemento a ser inserido com o nó raiz. Assim, por exemplo, se o elemento estiver a noroeste do elemento do nó raiz, percorre-se pela subárvore 1 tomando o nó apontado por esta como nossa nova raiz e então repetindo o processo.

Na figura 2.12 é ilustrado a seqüência inicial para a construção da Point Quadtree da figura 2.11, através da inserção dos nós: Cambuí, V. Georgina, J. das Palmeiras e Centro.

Remoção

Em [FB74], Finkel e Bentley propõem que para remoção de um determinado nó todos os demais subordinados a este deverão ser reinseridos. Para gerenciar grandes quantidades de dados armazenados em disco essa estratégia mostra-se ineficiente, pois os elementos a serem reinseridos provavelmente causariam inúmeros acessos a disco.

Uma solução mais eficiente foi proposta por Samet [Sam80] possibilitando assim a utilização dessa estrutura para o gerenciamento de base de dados espaciais. A idéia é a mesma para remoção em árvore binária de pesquisa, onde o nó que substituirá o nó a ser removido será o que possuir a chave mais próxima em valor. Para Point Quadtree, a dificuldade reside na verificação do nó a ser escolhido, pois devido a dimensionalidade dos dados é preciso encontrar um nó que seja simultaneamente próximo nas direções do eixo x e do eixo y.

Suponha através da figura 2.11, o processo de remoção e substituição do nó Cambuí pelo nó V. Aeroporto. Facilmente verifica-se que uma grande quantidade de espaço necessita ser reavaliada, pois com esta substituição, alguns dados passam a ter posições geográficas diferentes em relação a nova raiz. Nesse exemplo, os nós Centro e Ponte Preta, anteriormente no quadrante sudoeste da raiz Cambuí, se encontrarão agora no quadrante nordeste da nova raiz V. Aeroporto. Da mesma forma, após a substituição, os nós V. Georgina, J. das Palmeiras, J. Eulina e J. Sta. Genebra também tornarão a estrutura inconsistente. O nó J. Nilópolis não sofrerá alteração, pois este ainda estará na mesma posição relativa a nova raiz, a nordeste. Assim, entende-se que quanto mais próximo o nó substituto estiver do nó a ser removido uma menor quantidade de espaço deverá ser avaliada. Na figura 2.13

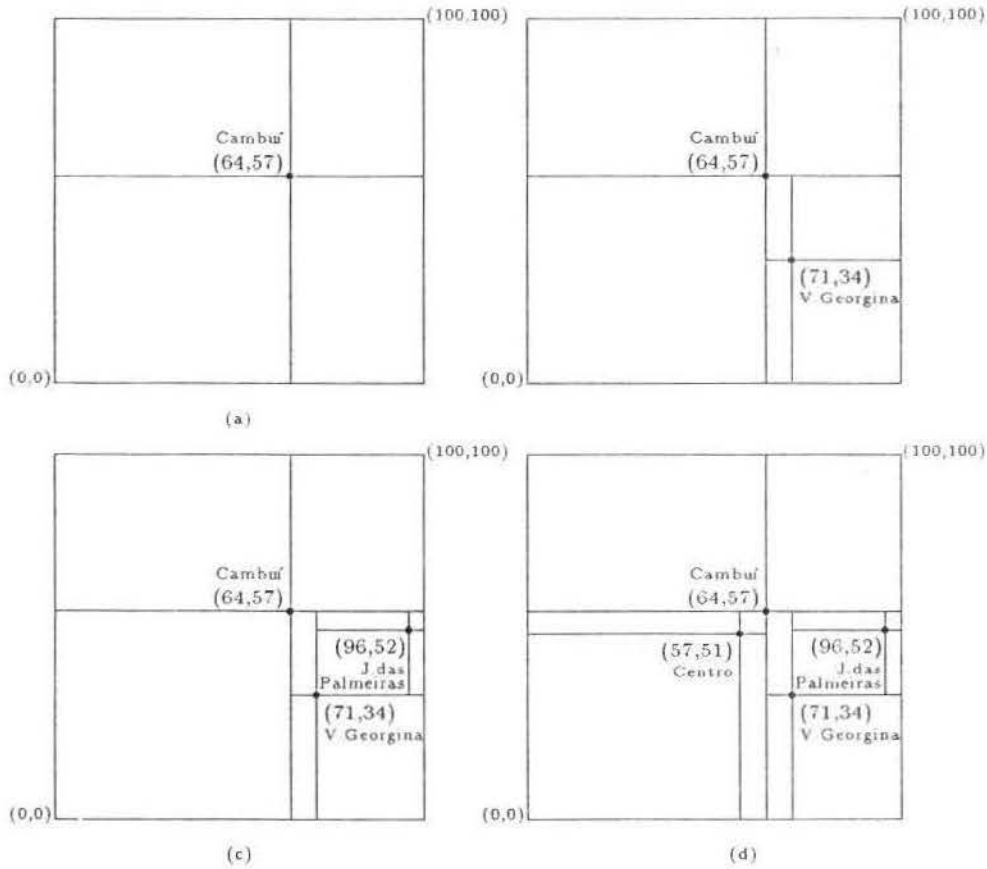


Figura 2.12: Demonstração de inserções em uma Point Quadtree através dos elementos: Cambuí (A), V. Georgina (B), J. das Palmeiras (C) e Centro (D).

verifica-se que a substituição do nó A pelo nó G irá requerer apenas que os espaço marcado como 1 e 2 necessitem ser avaliados, sendo então a melhor escolha dentre todos os demais nós da estrutura.

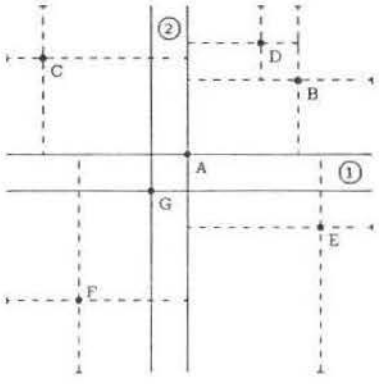


Figura 2.13: Exemplo de uma Point Quadtree onde a remoção de um nó não requer reinserções de outros elementos.

Segundo Samet, a partir do nó a ser removido tem-se quatro nós candidatos, um referente a cada subárvore. Estes deverão ser os nós mais próximos, espacialmente, do nó a ser removido. Para determinação do nó mais apropriado entre os quatro é utilizado dois critérios.

O primeiro critério designa a escolha do nó vencedor a ser o mais próximo de ambos os eixos do que outros candidatos do mesmo lado desses eixos (figura 2.14).

O critério acima pode acarretar em mais de um nó vencedor, como também na ausência deste (figura 2.15). Neste caso, o critério 2 é usado escolhendo como nó vencedor o que possuir menor valor métrico a ser definido abaixo (figura 2.16).

$$L_x.d_y + L_y.d_x - 2.d_x.d_y$$

onde:

- L_x e L_y representam o comprimento do espaço referente aos eixos x e y respectivamente.

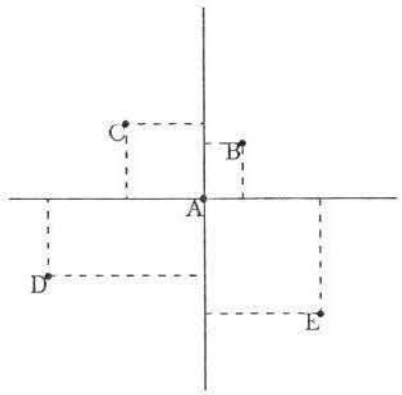


Figura 2.14: Quadtree com o nó B satisfazendo o critério 1.

- d_x e d_y representam a distância referente aos eixos x e y de um nó candidato para com o nó a ser removido.

Consulta

Point Quadtree são estruturas apropriadas para aplicações que visam detectar relacionamentos de proximidade, e.g., quais os nós que estão a uma distância z de um ponto (x,y).

Em [FB74] encontram-se algoritmos para detectar sobreposição sobre janelas (*windows*) de tamanho arbitrário. [Wil82] define consultas para detectar sobreposição de objetos mais complexos.

Considerações

O custo para se construir uma Point Quadtree é igual ao *total path length*[8] (TPL) da estrutura, que também reflete o custo para se pesquisar todos os seus elementos. Finkel e Bentley [FB74] mostram que o TPL sob inserções randômicas é proporcional a logarítmico em relação ao número de elementos ($O(N \log_4 N)$), enquanto que no pior caso[9] é quadrático ($O(N^2)$).

[8] o TPL de uma estrutura é igual a soma do comprimento do caminho de cada nó à raiz.

[9] o pior caso é dependente da ordem em que os dados são inseridos, ou melhor, quando cada sucessivo nó é armazenado como filho do nó mais profundo na estrutura.

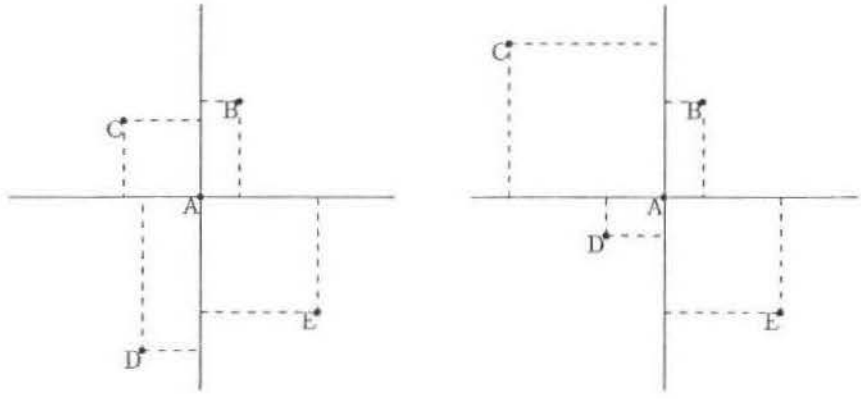


Figura 2.15: Quadtree: (a) sem nós satisfazendo o critério 1, (b) com B e D satisfazendo o critério 1.

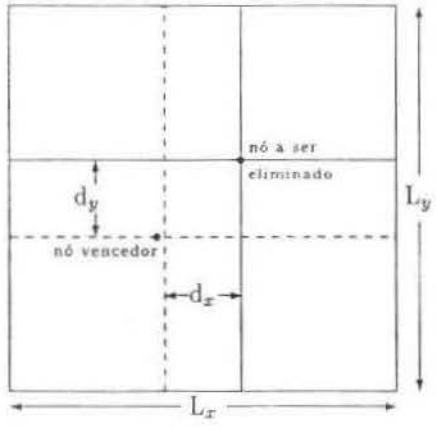


Figura 2.16: Definição do segundo critério de Samet para remoção.

Como o custo para buscas nesta estrutura esta relacionado com o TPL, é extremamente importante manter este valor o mais baixo possível. Finkel [FB74] propõe uma política para construção de uma Point Quadtree balanceada, porém, o inconveniente é que todos os nós tem que ser conhecidos a priori, tornando a aplicação desta estratégia somente interessante em base de dados estáticas.

Outro ponto importante para manter baixo o TPL é a estratégia utilizada para remoção de nós. Segundo Samet [Sam90a], o método proposto por Finkel causa um aumento significante, enquanto o método em [Sam80] causa uma redução.

Consultas para determinar se um ponto está ou não armazenado na estrutura podem ser satisfeitas, em média, em tempo $O(\log_4 N)$ (se o TPL da mesma é $O(N\log_4 N)$), ou até mesmo, em O(N) (se o TPL for $N(N-1)/2$). Este último resultado é insatisfatório para aplicações de banco de dados. [BSW77] apresenta estudos que mostram que o pior caso para consultas que visam detectar proximidade é $O(2N^{1/2})$.

2.3.2 K-d Trees

A K-d Tree [Ben75] é uma árvore binária de pesquisa multidimensional. A diferença desta estrutura para com a árvore binária de pesquisa é que em cada nível diferentes atributos (chaves) são utilizados para se determinar a direção da árvore a ser tomada. Uma K-d Tree possui as seguintes características:

- cada registro do arquivo está associado a um nó da árvore.
- cada nó possui dois apontadores, um para uma subárvore à esquerda e outro para uma subárvore à direita.
- associado a cada nível da árvore existe um campo discriminador que identifica qual subchave será utilizada como chave teste daquele nível. Uma subchave é um dos k valores que definem o objeto de dados no espaço k dimensional (e.g., um ponto no espaço de 2 dimensões possui 2 subchaves, um valor referente à coordenada x e outro referente à coordenada y).
- o primeiro nível da estrutura, a raiz, possuirá como discriminador a primeira subchave, o segundo nível possuirá a segunda até atingirmos o k-ésimo nível que possuirá a k-ésima subchave. A partir daí o processo se repete, o nível $k+1$ possuirá a primeira subchave e assim por diante.

- se um nó possui como discriminador a subchave referente ao eixo (dimensão) x, então todos os nós contendo valores referentes a x menores do que esse estarão na sua subárvore à esquerda. Analogamente, valores maiores estarão na sua subárvore à direita. Por convenção, discriminadores com valores iguais são alocados na subárvore direita.

Assim observa-se que cada nó em uma K-d Tree possui cinco campos:

- 2 apontadores para seus respectivos filhos.
- 2 campos referente às coordenadas x e y.
- 1 campo contendo a identificação desse nó, ou em nosso contexto, um apontador para o registro associado a esse.

Na figura 2.17 é apresentada uma Point Quadtree correspondente aos dados da figura 2.8.

Inserção

Como na Point Quadtree, o algoritmo para inserção de elementos em uma K-d Tree é semelhante ao da árvore binária de pesquisa. O primeiro elemento a ser inserido na estrutura será denominado de raiz da árvore e todos os demais serão sempre armazenados nas folhas da estrutura.

A distinção do funcionamento da K-d Tree em relação à árvore binária de pesquisa é justamente na forma em que se define em qual das subárvores, a partir de um determinado nó, deve-se percorrer para se chegar nas folhas da estrutura. Diferente da árvore binária de pesquisa onde o percurso é definido através do teste da chave do nó raiz da subárvore para com a chave do nó a ser inserido, na K-d Tree, somente parte da chave, indicada pelo discriminador daquele nível, será utilizada para teste.

Para melhor ilustrar o processo de inserção, na figura 2.18 é apresentada a seqüência inicial para construção da K-d Tree da figura 2.17 através da inserção dos nós: Cambuí, V. Georgina, J. das Palmeiras e Centro.

Remoção

A remoção de um determinado elemento em uma K-d Tree é mais complexa do que a eliminação em uma árvore binária de pesquisa. Nesta estrutura, o elemento a ser removido deve ser substituído por outro elemento de valor mais próximo a este. Este, por sua vez, será sempre um nó folha da subárvore

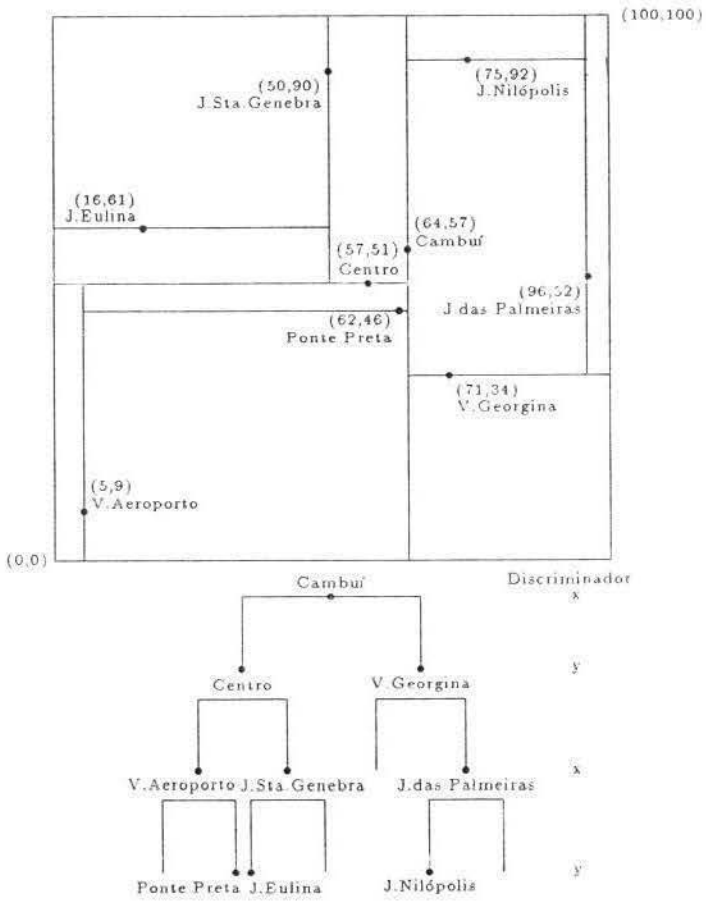


Figura 2.17: Uma K-d Tree e os registros que esta representa.

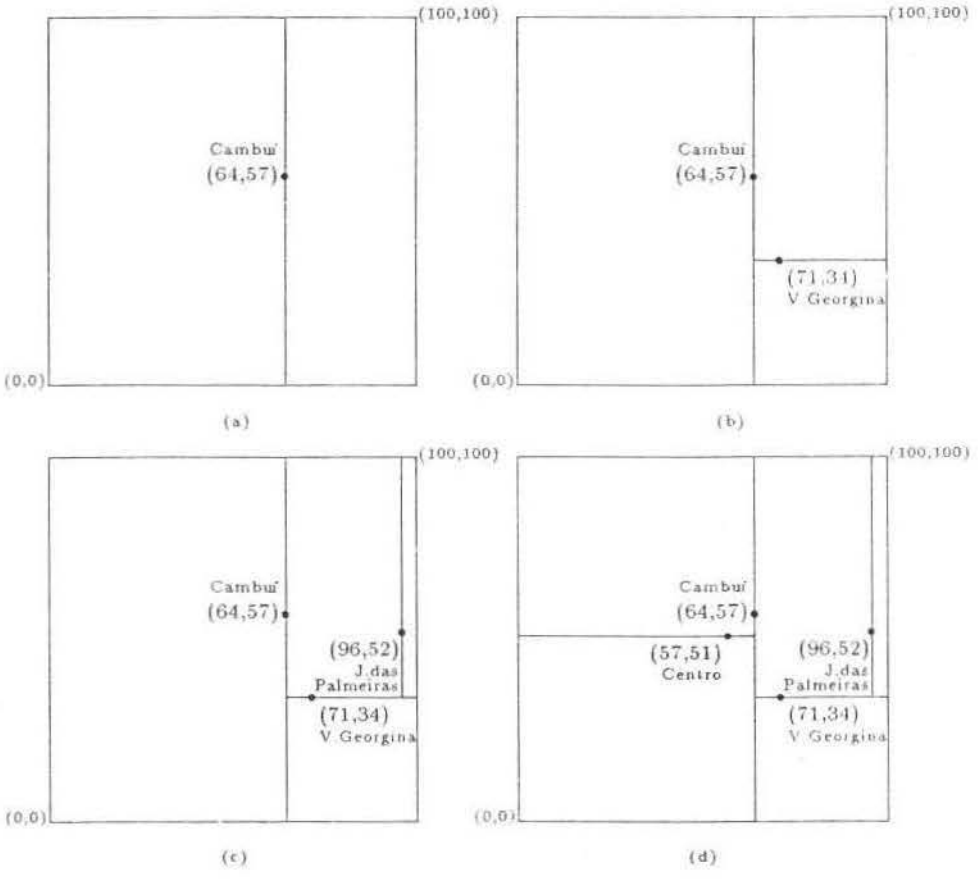


Figura 2.18: Demonstração de inserções em uma K-d Tree através dos elementos: Cambuí (A), V. Georgina (B), J. das Palmeiras (C) e Centro (D).

cuja raiz contém o nó a ser removido, ou o maior elemento em valor na subárvore esquerda, ou o menor elemento na subárvore direita.

A característica da K-d Tree que aumenta a complexidade da rotina de remoção é que nem toda subárvore de uma K-d Tree é uma K-d Tree (analisando a árvore da figura 2.19a, a subárvore de raiz (20,5) não é uma K-d Tree uma vez que o filho à esquerda da raiz possui coordenada x maior que o da raiz, enquanto deveria possuir coordenada menor). A principal conseqüência desta característica é que o processo de remoção não se resume na escolha de um nó substituto e, caso não seja um nó folha, na ascenção de sua subárvore em um nível. Dessa forma, a estratégia utilizada pela árvore binária de pesquisa não pode ser utilizada. A figura 2.19 ilustra o processo de remoção do nó raiz da K-d Tree apresentada em (a), através da escolha do nó (25,4) e ascenção da subárvore de raiz (30,1), ocasionando na K-d Tree inconsistente da figura (b).

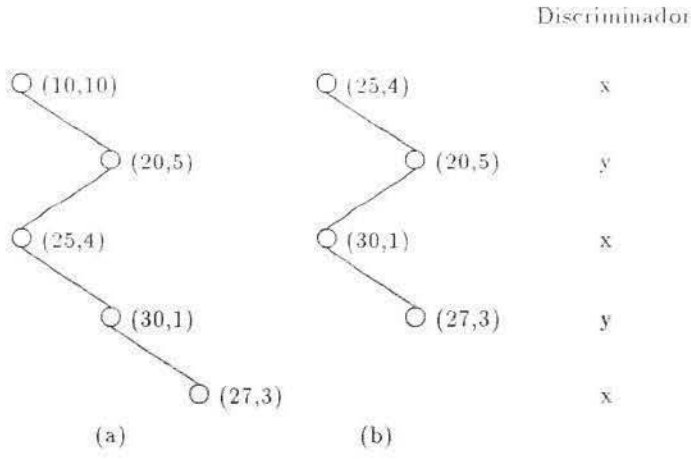


Figura 2.19: Um exemplo de inconsistência na eliminação do nó (10,10) em (a), resultando a árvore (b).

A solução para este problema é intrínseca ao funcionamento da rotina de remoção com a utilização de recursão nesse processo. Assim, a remoção de um nó em uma K-d Tree é definida da seguinte forma:

- se o nó possui subárvores vazias, troca-se este nó por uma subárvore vazia.

- se o nó possui subárvores não vazias, encontra-se um nó apropriado (c,d) e então o submete ao processo de remoção. Uma vez que um nó substituto ao nó (c,d) tenha sido encontrado, troca-se o nó (a,b) por (c,d).

Nesse ponto, a escolha de um nó apropriado é semelhante à escolha em uma árvore binária de pesquisa. Na subárvore esquerda procura-se o nó com maior valor na subchave indicada pelo discriminador do nó a ser removido, ou o nó com menor valor na subárvore direita.

Caso o nó escolhido esteja contido na subárvore esquerda deve-se certificar que não existam valores iguais a este, pois caso existam, a substituição do nó removido por este causará uma inconsistência na estrutura. A inconsistência existirá porque ao invés da subárvore esquerda da nova raiz possuir valores menores do que o da raiz, pelo menos um nó desta subárvore possuirá valor igual.

Em conseqüência da maior complexidade para determinação da existência de valores repetidos, pode-se dar preferência na escolha do nó substituto a partir da subárvore à direita, onde tal exigência não se faz necessário. Samet [Sam90a] apresenta um solução eficiente caso o nó a ser removido possua subárvore à esquerda não vazia e subárvore à direita vazia.

Na figura 2.20 é apresentada a configuração da K-d Tree da figura 2.19a após a eliminação a raiz (10,10), utilizando-se a metodologia para remoção de K-d Tree.

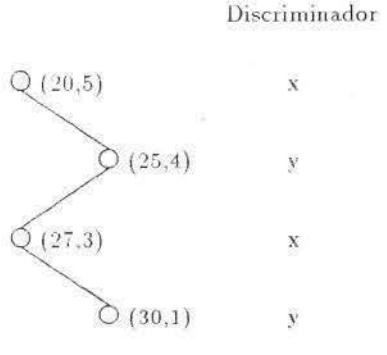


Figura 2.20: Um exemplo de uma K-d Tree.

Consulta

De maneira análoga à Point Quadtree, a estrutura K-d Tree é apropriada para detectar relacionamentos de proximidade.

Bentley [Ben75] define alguns tipos de consulta como também a metodologia para determinação das mesmas.

Considerações

Bentley [Ben75] mostra que o TPL de uma K-d Tree construída pela inserção em ordem randômica de N pontos em uma árvore inicialmente vazia é $O(N \log_2 N)$. Analogamente à Point Quadtree, no pior caso o TPL da estrutura é $N(N-1)/2$.

Para evitar este último resultado, que pode ser obtido a partir de uma má distribuição dos dados a serem inseridos na estrutura, algumas soluções são propostas. Bentley [Ben75] propõe uma estatégia para construção da estrutura partindo do princípio que todos os objetos de dados são conhecidos a priori, enquanto que Overmars [Ov82] propõe uma estratégia para reorganização da estrutura a ser executada periodicamente. Como já discutido, a utilização destas estratégias em um ambiente de Banco de Dados nem sempre é possível. A estratégia de Bentley mostra-se apropriada apenas para aplicações estáticas, enquanto a de Overmars traz demais conseqüências em um ambiente de multi-processamento, já que a cada reorganização da estrutura, as aplicações que as utilizam devem ser interrompidas.

Como na Point Quadtree, consultas com o objetivo de determinar se um ponto está ou não armazenado na estrutura podem ser satisfeitas em média, em tempo $O(\log_2 N)$ (se o TPL da mesma é $O(N \log_2 N)$), ou até mesmo em $O(N)$ (se o TPL for $N(N-1)/2$). Este último resultado é insatisfatório para aplicações de banco de dados.

2.3.3 MX-CIF Quadtree

A MX-CIF Quadtree é uma Quadtree cuja decomposição recursiva do espaço é feita em partes iguais (decomposição regular). Esta estrutura associa cada objeto de dados, representado pela estratégia MBR, a um nó na Quadtree de menor bloco que contém esse objeto em sua totalidade. O bloco de um nó em uma Quadtree é o espaço representado por todos os quadrantes deste nó (na figura 2.21, por exemplo, o bloco do nó que representa os objetos de dados G e A é representado pelo retângulo de limite inferior esquerdo (50,50) e limite superior direito (100,100)). As subdivisões cessam sempre

que um bloco de determinado nó não contém mais retângulos, ou em algumas soluções alternativas, quando o bloco da Quadtree se torna menor do que um tamanho predeterminado (este tamanho normalmente é designado para ser igual ao menor tamanho esperado de um retângulo).

A partir da definição acima consegue-se identificar algumas características desta estrutura.

- os retângulos podem estar associados a nós folhas, como também, a os nós intermediários.
- um ou mais retângulos podem estar associados a um mesmo nó na Quadtree.

Acrescida às características acima há a restrição de que uma vez que um retângulo esteja associado a um nó da Quadtree, este não poderá estar associado a nenhum dos nós sucessores a este (seus filhos). Em outras palavras, todos os retângulos que cortam as linhas que passam através do centro do bloco de um nó denominado X, desde que já não tenham sido associados a nenhum outro nó antecessor de X, serão associados a X.

Para completar a definição da MX-CIF Quadtree resta saber como organizar o conjunto de retângulos associado a cada nó da estrutura. As soluções são diversas, desde a utilização de estruturas mais simples, como listas ligadas, a representação destes por outra MX-CIF Quadtree de uma dimensão, que na realidade, são árvores binárias (solução projetada por [Ked82]). Esta última estratégia, descrita abaixo, será a adotada em nosso estudo da MX-CIF Quadtree.

O conjunto de retângulos associado a cada nó da estrutura é dividido em dois subconjuntos. Para cada nó, esses conjuntos serão determinados pelos retângulos que passam pelo centro do bloco representado por este nó. Se o bloco representado por um nó P qualquer possui centro (C_x,C_y), então os retângulos que interceptam a linha $x = C_x$ farão parte de um conjunto (denominado de x), enquanto que os retângulos que interceptam a linha $y = C_y$ farão parte do outro conjunto (denominado de y). Por convenção, o retângulo que interceptar as duas linhas estará representado no conjunto y.

Na figura 2.21 é apresentada a MX-CIF Quadtree correspondente aos dados da figura 2.9. Para ilustrar a representação dos retângulos associados a um dado nó, na figura 2.22 é apresentado a árvore binária[10] referente ao

[10] a construção dessas árvores binárias é semelhante a construção da MX-CIF Quadtree para o espaço unidimensional, assim, para cada um dos conjuntos, as partições recursivas são feitas sobre apenas um dos eixos.

nó raiz da figura 2.21.

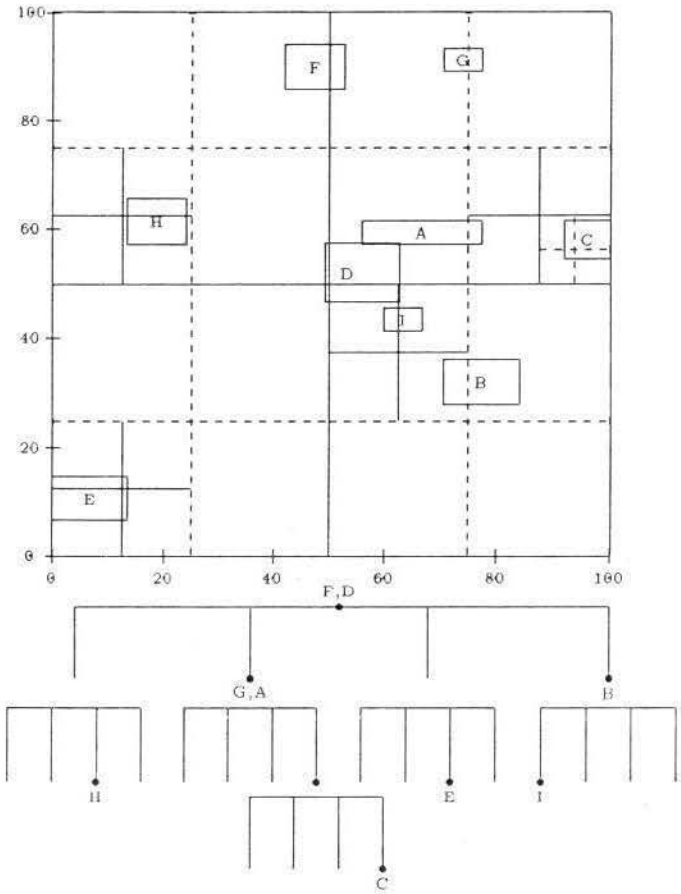


Figura 2.21: Uma MX-CIF Quadtree e os retângulos que esta representa.

Verifica-se que para a representação de uma MX-CIF Quadtree são necessários três tipos de registros:

- o que representa os retângulos, *rect*, composto de 5 campos:
  - 4 campos correspondentes às coordenadas do centro e da distância do centro até a borda desse retângulo, para os eixos x e y.

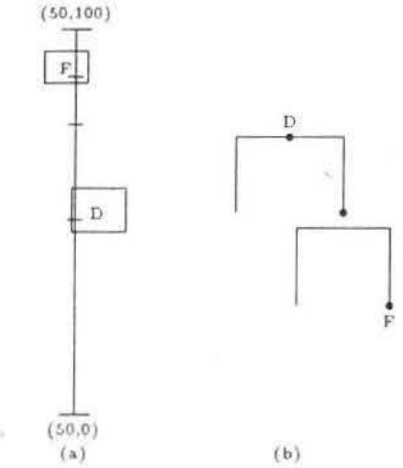


Figura 2.22: Um exemplo da árvore binária em (b), representando os retângulos em (a).

- 1 campo contendo a descrição desse retângulo.

- o que representa os nós das árvores binárias, *bin-node*, composto de 3 campos:
  - 2 apontadores para as subárvores à direita e à esquerda.
  - 1 apontador para um registro do tipo *rect*, referente ao retângulo representado por esse nó na árvore binária.

- o que representa os nós da árvore MX-CIF Quadtree, *cif-node*, composto por 6 campos:
  - 4 apontadores para as subárvores à noroeste, nordeste, sudoeste e sudeste.
  - 2 apontadores para o registro do tipo *bin-node*, que representam os dois conjuntos de retângulos associados a este nó.

Inserção

Para a inserção de um retângulo em uma MX-CIF Quadtree é necessário percorrer a estrutura até que se encontre um nó, onde pelo menos uma das

linhas que passam pelo centro do bloco associado a este interceptem esse retângulo.

Como a divisão do espaço é feita de forma regular, a raiz possui o centro do seu bloco nas coordenadas (C_x,C_y), onde C_x e C_y referem-se a metade do comprimento do espaço nos eixos x e y. A partir daí, para cada filho da raiz, a subdivisão do espaço é feita de forma semelhante. Assim, possuirão como coordenada central de seus blocos os pontos ($C_x/2$,$3C_y/2$), ($3C_x/2$,$3C_y/2$), ($C_x/2$,$C_y/2$) 3 ($3C_x$,$C_y/2$), referentes aos quadrantes noroeste, nordeste, sudoeste e sudeste respectivamente.

A direção durante o percurso na estrutura é determinada pela posição relativa do retângulo com o nó corrente, e incorre em sucessivas subdivisões do espaço (denominadas *splitting*) até que a condição apresentada anteriormente seja satisfeita.

Após determinado o nó da MX-CIF Quadtree que se refere ao retângulo a ser inserido, deve-se armazená-lo em um dos conjuntos associados a este nó. Como já comentado, esse conjunto é representado como uma MX-CIF Quadtree de uma dimensão. Dessa forma, o processo de inserção é semelhante ao apresentado acima distinguindo-se apenas em que o espaço a ser particionado será único, no eixo x se o retângulo estiver associado ao conjunto referente a este eixo, ou eixo y se estiver associado ao conjunto y.

Para ilustrar este processo de inserção é apresentado na figura 2.23 a seqüência inicial para a construção da MX-CIF Quadtree através dos retângulos: Cambuí (A); V. Georgina (B); J. das Palmeiras (C); e Centro (D).

Remoção

A remoção de retângulos na estrutura MX-CIF Quadtree é mais simples que a remoção de pontos na Point Quadtree.

Dado um retângulo a ser removido deve-se primeiro encontrar o nó associado a este. Este processo de busca é semelhante ao utilizado para inserção desse elemento, ou seja, deve ser encontrado o nó cujas linhas que passam pelo centro do seu bloco interceptem esse retângulo.

Uma vez encontrado o nó na MX-CIF Quadtree, a árvore binária referenciada por este deve ser pesquisada com o objetivo de remover a representação desse retângulo nessa estrutura. O percurso na árvore binária também é semelhante ao utilizado para inserir esse retângulo nessa estrutura.

Após a determinação do nó, na árvore binária, que contém referência para esse retângulo, essa referência é removida. A partir deste ponto, se este

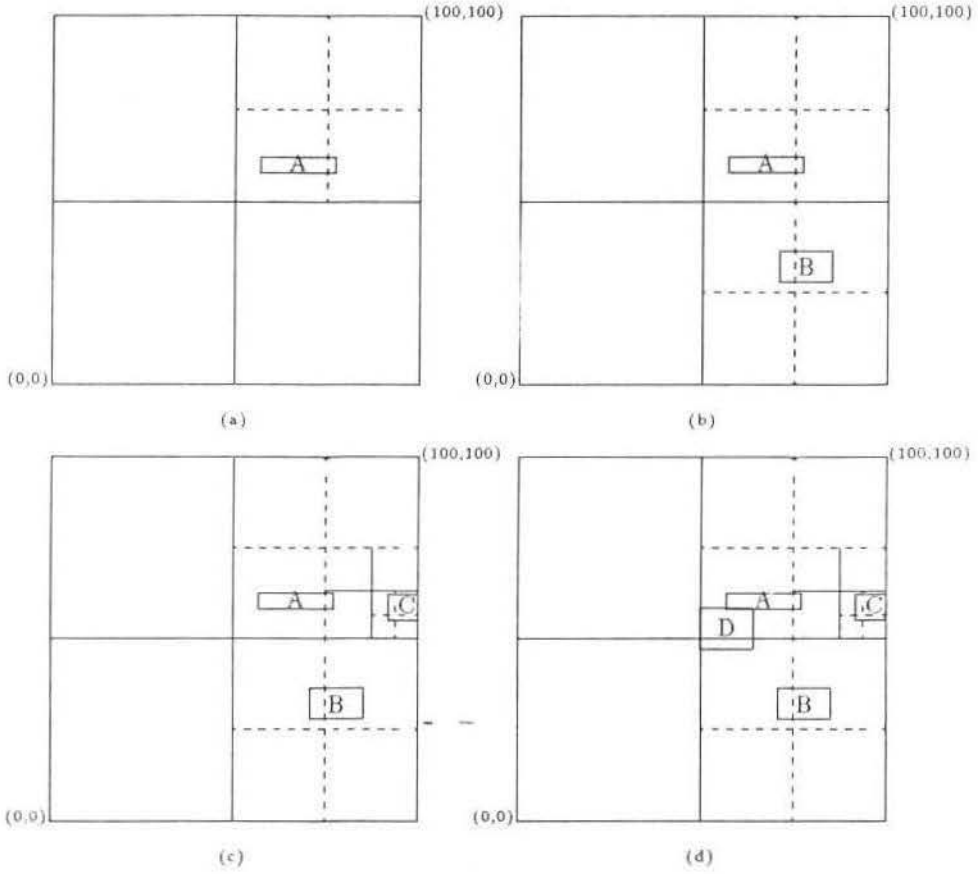


Figura 2.23: Demonstração de inserções em uma MX-CIF Quadtree através dos elementos: Cambuí (A), V. Georgina (B), J. das Palmeiras (C) e Centro (D).

nó possui subárvores vazias, este deve sofrer uma operação de *merge* (junção) através de sua eliminação e, conseqüentemente, da remoção da referência de seu nó pai. Uma vez executada uma operação de *merge*, esta pode ser propagada a todos os nós antecessores a este e, até mesmo, à MX-CIF Quadtree, caso a eliminação do retângulo deixe as referências para as árvores binárias nulas.

Para facilitar o entendimento deste processo, suponha uma operação de remoção sobre a árvore apresentada na figura 2.22. A remoção do retângulo F causa na árvore binária da figura 2.22b, a remoção da referência para o registro do tipo *rect* que representa o nó F. Como este nó possui subárvores vazias, este é eliminado com a atribuição para *null* (nulo) da subárvore direita de seu nó pai. Este nó pai passa então a possuir subárvores vazias. Como não possui referência ao registro do tipo *rect*, este também é eliminado com a atribuição para *null* (nulo) da subárvore direita do nó denominado D. Embora as subárvores de D sejam vazias, este nó não pode ser eliminado devido a referência existente para o registro do tipo *rect*. Caso o retângulo D fosse removido, essa árvore binária se tornaria vazia, propagando o *merge* ocorrido na eliminação desse nó para a MX-CIF Quadtree.

Consulta

A MX-CIF Quadtree é uma estrutura apropriada para representar coleções de pequenos retângulos. Aplicações com estas características normalmente requisitam consultas que visam detectar todos os retângulos em uma dada área, como também, todos os retângulos que sobrepõe uma dada região.

Considerações

A construção de MX-CIF Quadtree de profundidade máxima igual a n, sendo n a soma da profundidade da MX-CIF-Quadtree e da árvore binária, para N retângulos, possui como pior caso para tempo de execução $O(n.N)$.

Consultas do tipo *range queries* para determinação de intersecção não são suportadas eficientemente por esta estrutura. O problema é que não basta a busca nas subárvores relacionadas ao nó associado ao retângulo de consulta, já que os demais retângulos associados aos nós antecessores a este, na MX-CIF Quadtree, poderão também interceptar a área de pesquisa. Dessa forma, dependendo do nível da estrutura associado ao retângulo de consulta, esse processo pode tornar-se caro em conseqüência da possibilidade de se pesquisar todos os nós antecessores a este. Na figura 2.24 este problema

é ilustrado. Embora o retângulo de consulta X esteja associado ao nó Nordeste da raiz, os retângulos associados à raiz devem ser pesquisados, pois conforme ilustrado existe sobreposição.

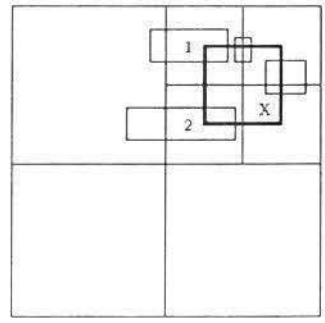


Figura 2.24: Demonstração de uma *range query* em uma MX-CIF Quadtree.

Essa característica definida acima interfere fortemente no tempo de resposta para este tipos de consultas. Soluções alternativas têm sido propostas até mesmo com a criação de outras estruturas derivadas da MX-CIF Quadtree (PR1 Quadtree e PR2 Quadtree). Porém, estas estruturas inserem outros problemas que vão desde o aumento de espaço necessário para armazenamento de seus dados ao aumento da complexidade de tempo para outras operações básicas (construção e inserção, por exemplo), tornando-se assim, também inadequadas.

2.3.4 Spatial K-d Tree

A Spatial K-d Tree (SKD Tree) proposta por Ooi [OMSD87, Ooi90] é uma variação da estrutura K-d Tree [Ben75]. A SKD Tree permite além da representação de pontos, como na K-d Tree, a representação de objetos de tamanho não-zero. A estratégia utilizada para representação de objetos espaciais de formas irregulares é a MBR. Cada objeto de dados é representado por uma tupla $(I_1.....I_K)$ onde I_i representa o intervalo fechado na dimensão I que descreve a extensão e localização do objeto nessa dimensão.

Análogo à K-d Tree, o percurso na SKD Tree é determinado por uma das dimensões do objeto de dados K-dimensional, que é identificado pelo campo

discriminador de cada nó. Esta dimensão provê a partição do espaço em dois subespaços resultantes (direito e esquerdo) indicado pelas duas subárvores desse nó.

Como na K-d Tree, objetos de dados que são pontos no espaço são totalmente incluídos em um desses subespaços, dependendo do valor da coordenada desses pontos na dimensão indicada pelo campo discriminador. Objetos de dados de tamanho não-zero podem sobrepor à outros subespaços, pois para estes objetos, a coordenada utilizada para direcionar o percurso na estrutura é a do centro do MBR. Para evitar a divisão do objeto e a duplicação de índices quando objetos sobrepõe outros subespaços, um valor extra é utilizado em cada subespaço para indicar o limite dos MBRs dos objetos naquela dimensão.

A SKD Tree é composta por dois tipos de nós:

- nós intermediários - composto por uma entrada do seguinte tipo:

  (disc, $\max_{loson}$, loson-ptr, disc-value, hison-ptr, $\min_{hison}$)

  onde,

  - **disc** indica a dimensão a ser utilizada como chave para direcionar o percurso, ou melhor, indica a dimensão em que o espaço está sendo particionado.
  - **$\max_{loson}$** indica o valor máximo na dimensão indicada por *disc* dos MBRs do subespaço esquerdo.
  - **loson-ptr** contém o endereço da raiz da subárvore esquerda.
  - **disc-value** indica a coordenada de partição do espaço na dimensão *disc*.
  - **$\min_{hison}$** indica o valor mínimo na dimensão indicada por *disc* dos MBRs do subespaço direito.
  - **hison-ptr** contém o endereço da raiz da subárvore direita.

- nós folhas - composto por uma entrada do tipo:

  (bound, min-range, page-pointer, max-range)

  onde,

  - **bound** indica uma dimensão.

- **min-range** indica o valor mínimo dos objetos na dimensão especificada por *bound*, na página referenciada por *page-pointer*.
- **page-pointer** contém o endereço de uma página de disco (*bucket*).
- **max-range** indica o valor máximo dos objetos na dimensão especificada por *bound*, na página referenciada por *page-pointer*.

A página endereçada pelo nó folha da estrutura SKD Tree deve conter entradas do tipo (MBR, id), onde MBR especifica o MBR do objeto de dados e id a referência do dado associado a esse MBR.

Na figura 2.25 é apresentado a Spatial K-d Tree correspondente aos dados da figura 2.9 com o valor de M e m iguais a 4 e 2 respectivamente.

Inserção

Para inserção de objetos de dados na SKD Tree, cada um destes será inserido no *bucket* mais apropriado de acordo com as coordenadas do centro do seu MBR. A determinação desse *bucket* é feita, recursivamente, a partir da raiz da árvore, da seguinte forma. Se o nó corrente é um nó intermediário, o percurso procede pela subárvore esquerda desse nó, se a coordenada do centro do MBR na dimensão indicada pelo campo *disc* desse nó é menor ou igual ao campo *disc-value*. Nesse caso, se o limite superior do MBR nessa dimensão for maior que o campo max_{loson}, este campo será atualizado com este valor. Caso a coordenada do centro do MBR na dimensão *disc* seja maior que o campo *disc-value*, o percurso será feito pela subárvore direita atualizando o campo min_{hison} com o menor valor entre o atual e o limite inferior do MBR.

Se o nó corrente for um nó folha tem-se duas possibilidades: se o *bucket* tem condição de armazenar o novo objeto de dados ou não. Caso o *bucket* possa armazenar o novo dado, os campos *bound*, *min-range* e *max-range* serão reavaliados. Dentre todas as dimensões, *bound* irá representar a dimensão de menor área entre *min-range* e *max-range*. Caso o *bucket* não possa armazenar o novo dado este sofrerá uma operação de *split*[11].

A operação de *split* identifica a dimensão de maior extensão e a particiona em dois subespaços. Esse nó folha é então transformado em um nó intermediário e é criado dois novos nós folhas. O *bucket* anterior é associado a uma das folhas e um novo *bucket* também é criado. Os MBR's de

[11] que significa divisão, partição.

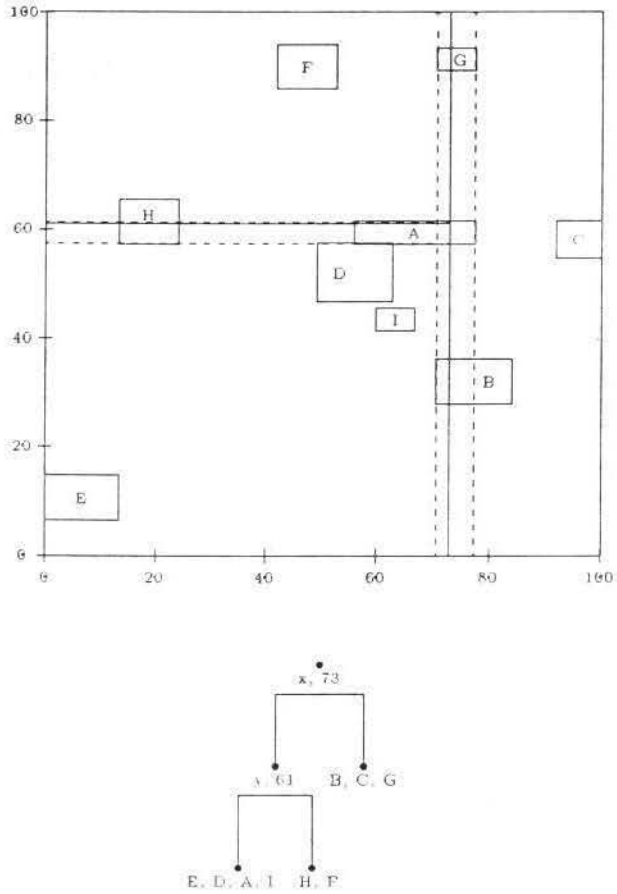


Figura 2.25: Uma SKD-Tree e os retângulos que esta representa.

todas as entradas, por meio de suas coordenadas, são classificados na ordem ascendente, em relação a dimensão escolhida, e então distribuídos nos dois *buckets*.

Na figura 2.26 é ilustrado o processo de inserção para a construção da Spatial K d-Tree apresentada em 2.25, através da seqüência dos retângulos: Cambuí (A); V. Georgina (B); J. das Palmeiras (C); Centro (D); V. Aeroporto (E); J. Sta. Genebra (F); e J. Nilópolis (G), J. Eulina (H) e Ponte Preta (I).

Remoção

Para remoção de um elemento na SKD Tree deve-se, em primeiro lugar, determinar o *bucket* que contém o elemento. Para isso, procede-se de forma semelhante à descrita na operação de inserção. Uma vez determinado o *bucket*, o elemento é removido. Se após a remoção o *bucket* permanece com menos que m elementos (igual à metade da capacidade máxima), este *bucket* sofre uma operação de *underflow*[12] com o objetivo de melhorar o aproveitamento de espaço.

Na operação de *underflow*, as entradas do *bucket* são redistribuídas com o *bucket* referenciado pelo nó vizinho. Se nesta redistribuição ocorre *overflow*, o *bucket* sofre uma operação de *split*. Contudo, se o nó folha que referencia o *bucket* a sofrer *underflow* não possuir um nó folha como vizinho, emprega-se outra estratégia. Nesses casos, esse nó folha é removido e o seu respectivo nó pai é substituído pelo nó raiz de sua outra subárvore. O *bucket* é removido e suas respectivas entradas são reinseridas a partir da subárvore indicada pelo nó que foi substituído.

As figuras 2.27 e 2.28 ilustram a operação de *underflow* através da última condição descrita acima. Suponha a operação de *underflow* no *bucket* referenciado pelo nó folha identificado como 3 (figura 2.27), ocasionando na SKD Tree apresentada na figura 2.28.

Consulta

Conforme Ooi, uma das grandes vantagens da SKD Tree é que diferente de outras estruturas cuja reorganização dos dados propicia apenas um tipo de consulta espacial, *range queries* para determinação de intersecção (R-Trees). A SKD Tree suporta, além deste tipo de consulta, *range queries*

[12] estouro do limite inferior da capacidade de armazenamento

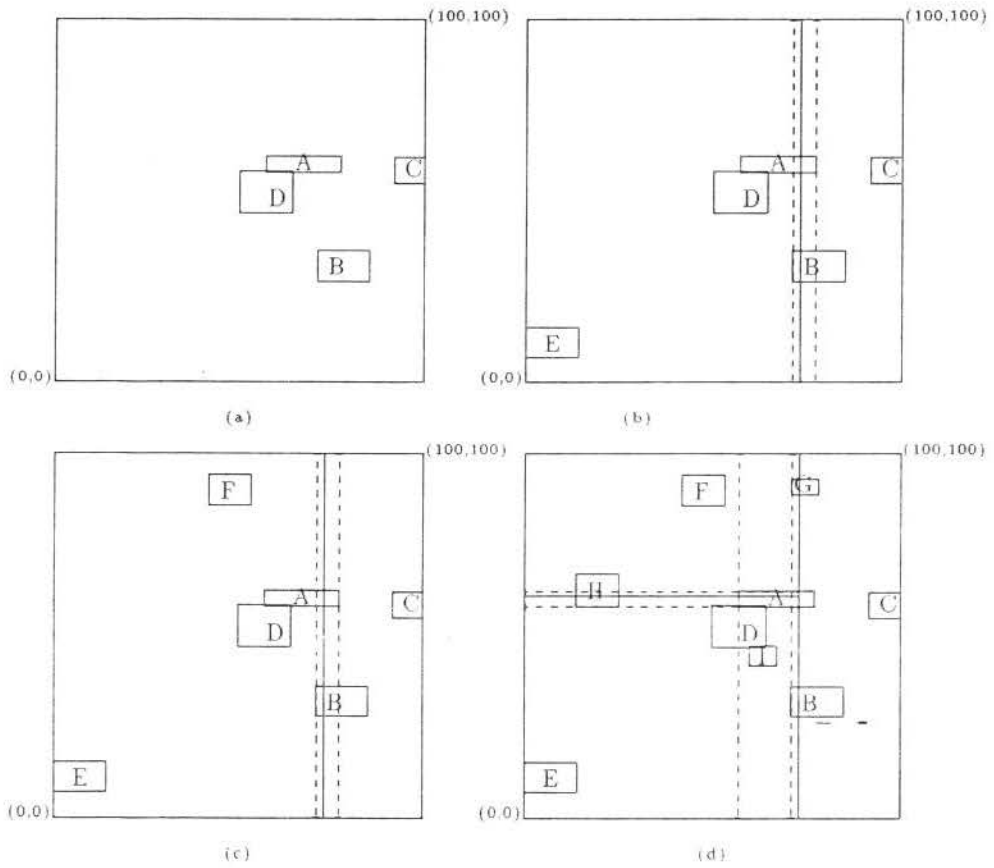


Figura 2.26: Demonstração de inserções em SKD-Tree através dos elementos: Cambuí (A), V. Georgina (B), J. das Palmeiras (C), Centro (D), V. Aeroporto (E), J. Sta. Genebra (F), J. Nilópolis (G), J. Eulina (H) e Ponte Preta (I).

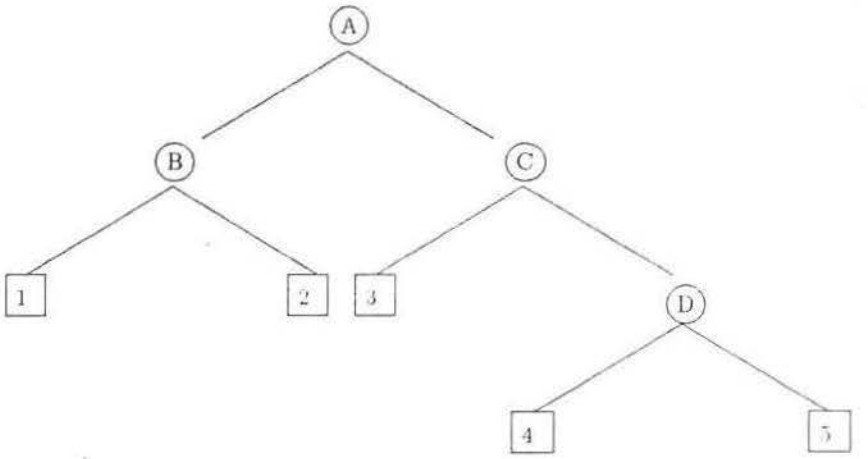


Figura 2.27: Representação de uma Spatial K-d Tree.

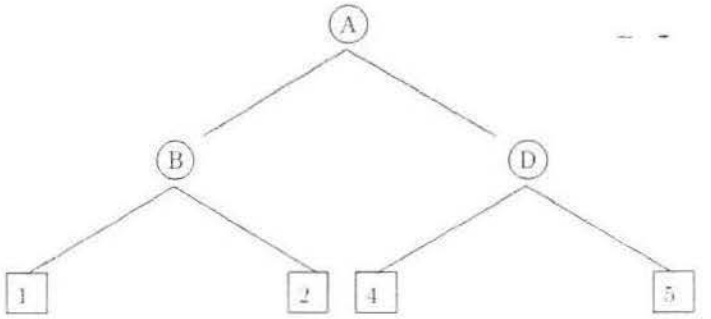


Figura 2.28: Ilustração da operação de *underflow* sob a Spatial K-d Tree da figura 2.27.

para determinação de objetos de dados contidos na área de pesquisa (*range queries* para determinação de não-inclusão).

Na realidade, o que ocorre é que na R-Tree, por exemplo, não existe nenhuma distinção entre esses dois tipos de consultas durante os teste para determinação do percurso nos nós intermediários. Já na SKD Tree, conseqüente à organização dos objetos de dados, isso não acontece. Cada nó particiona o espaço em dois, sendo tanto os objetos de dados do tipo ponto como os objetos de tamanho não-zero, através do centro do seu MBR, dispostos exclusivamente em um desses subespaços. Porém, objetos de dados de tamanho não-zero podem se estender através de outros subespaços, sendo o limite máximo dessa sobreposição indicado em cada nó (através de max_{loson} e min_{hison}). Assim, uma janela (retângulo) de consulta contida sobre essa área comum aos dois subespaços, para determinação de intersecção, necessitaria da pesquisa sobre esses subespaços, enquanto que para o outro tipo de consulta, somente um subespaço seria pesquisado. A razão disso é que os centros dos objetos do subespaço que invade o outro subespaço nunca estarão representados nessa área comum e, conseqüentemente, nem na janela de consulta, já que esta é uma condição necessária para que qualquer objeto possa satisfazer esse tipo de consulta.

A figura 2.29 ilustra esses tipos de consulta onde verifica-se que a determinação da intersecção de objetos sobre a janela W obriga a pesquisa sobre os dois subespaços, enquanto que a determinação dos objetos contidos nessa janela só requer a busca do subespaço 2.

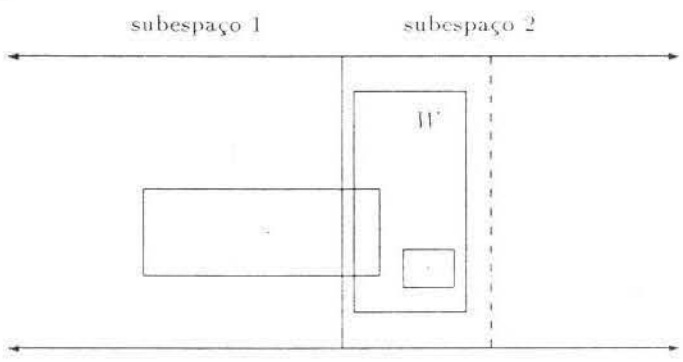


Figura 2.29: Ilustração de *range queries* sobre a SKD Tree.

Considerações

A SKD Tree incorpora as características da K-d Tree que tornam discutível a sua utilização em SGBDs. A SKD Tree é uma estrutura estática e de baixo *fan-out* por nós, e como já discutido, tais características são inadequadas para aplicações com atualização de dados que são armazenados em disco.

Contudo, Ooi [OMSD87, Ooi90] sugere a adoção da estratégia *multi-way trees* [Knu73b] na implementação da estrutura de dados, com o fim de prover uma melhor relação na distribuição dos nós nas páginas de disco. Essa melhor distribuição é obtida através do armazenamento das subárvores da estrutura de dados nas páginas de disco (figura 2.10). Essa estratégia é essencial na implementação de estruturas de dados derivadas de árvores binárias, uma vez que evita as várias *page faults*, que ocorreriam durante o percurso na estrutura se os nós estivessem armazenados desordenadamente nas páginas de disco.

Uma outra característica favorável da SKD Tree é que embora a estrutura de dados seja estática, a metodologia utilizada pela SKD Tree para remoção de um nó, seja através da operação de *merge* com um nó vizinho ou da ascenção da subárvore vizinha em um nível, provê uma estrutura de dados mais balanceada.

Segundo Ooi, o ponto mais favorável de sua estrutura de dados é o suporte a dois tipos distintos de consultas. Em seus experimentos, Ooi defende essa versatilidade como um critério essencial para melhorar o desempenho das consultas.

2.4 Derivados de Estruturas Hash

Diferente dos métodos de acesso derivados de árvores binárias, que foram adaptados para o gerenciamento de dados em disco, alguns métodos de acesso foram projetados exclusivamente para essa condição. Nos métodos de acesso apresentados na seção anterior identificou-se como um dos pontos negativos o armazenamento da estrutura de dados no disco. O fato da unidade de leitura das estruturas de dados (nós) utilizar área de armazenamento muito inferior à existente nas páginas de disco (unidade de leitura entre a memória principal e secundária), implica que o armazenamento sem nenhuma organização predefinida dos diversos nós cause várias e caras *page faults* durante o percurso nessa estrutura. Assim, diversos métodos de acesso foram projetados tendo como idéia básica o armazenamento dos dados em disco dispostos em páginas de disco (*buckets*).

O objetivo desses métodos reside na transformação de um objeto de dados espacial para um valor que indica o *bucket* associado a este objeto. Essa idéia é semelhante a usada nos métodos *hash* [Knu73b]. A diferença básica está na função de transformação, pois para os métodos de acesso a dados espaciais, esta deve prover um mapeamento multidimensional (dados no espaço) para unidimensional (páginas de disco).

Um dos primeiros métodos promissores a atender esse objetivo foi o Grid File proposto por Nievergelt [NHS84]. Nesse método de acesso, a associação dos dados aos *buckets* é feita através da divisão do espaço K-dimensional onde os dados estão dispostos em um conjunto de retângulos de K dimensões. Esse espaço subdividido é denominado de diretório (*Grid Directory*) e a cada um de seus componentes (retângulos) está associado um *bucket* que armazena os dados referentes ao espaço representado por esse retângulo. Quando um elemento de dados deve ser inserido em um *bucket* e este não tem capacidade para armazená-lo, seus elementos são redistribuídos em um novo *bucket*. Em decorrência da correspondência do diretório com as páginas de disco, essa redistribuição pode causar a possibilidade da partição desse diretório através da divisão de todos os retângulos em uma dada dimensão, fazendo com que uma página de disco esteja associada a mais de um componente de diretório (figura 2.30).

Conforme Freeston [Fre87], esta característica faz com que o número de componentes do diretório cresça exponencialmente quando a distribuição dos dados deixa de ser uniforme e, conseqüentemente, o desempenho do Grid File seja degradado em virtude do maior custo das operações executadas sobre esse diretório.

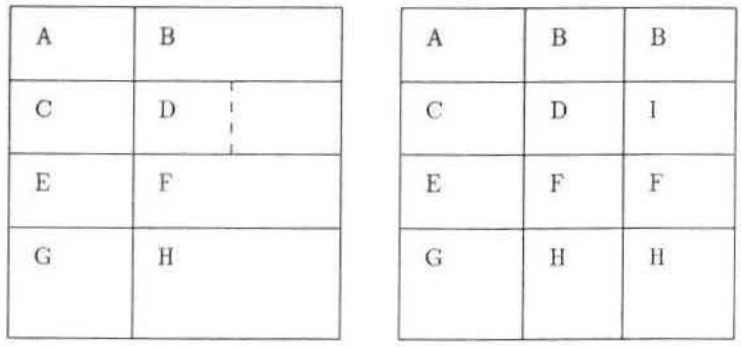


Figura 2.30: Exemplo da partição do diretório em um Grid File após o *bucket* referenciado por D sofrer uma divisão.

Hinrichs [Hin85] descreve uma implementação alternativa para o Grid File de acordo como sugerido na publicação original de Nievergelt [NHS84]. A idéia básica reside na manutenção do diretório em dois níveis: um nível mantido na memória principal com resolução menor; e um segundo nível mantido no disco. O primeiro nível de diretório, ao invés de fazer a correspondência de seus componentes aos *buckets*, faz aos demais diretórios do segundo nível. Devido à independência desses dois níveis de diretório, esta estratégia suporta uma melhor adaptação às distribuições não-uniformes, todavia, conforme [Fre87, KSSS89], a expansão do diretório ainda se faz de forma exponencial.

O DYOP Grid File proposto por Ozkarahan e Ouksel [OO87] foi desenvolvido a partir do Grid File e do trabalho de Burkhard [Bur83] sobre interpolação baseada em *hashing*. Este método de acesso evita o crescimento exponencial do Grid File através da associação de uma página de disco para cada componente do diretório. A idéia básica reside na partição hierárquica do espaço associada à página de disco que sofreu *split* (divisão) em dois subespaços iguais. Assim, diferente do Grid File, uma divisão de um *bucket* acarreta em apenas um componente a mais no diretório (figura 2.31).

Porém, conforme Freeston [Fre87] os objetos de dados de distribuição não-uniforme não são suportados eficientemente. O diretório apresentado

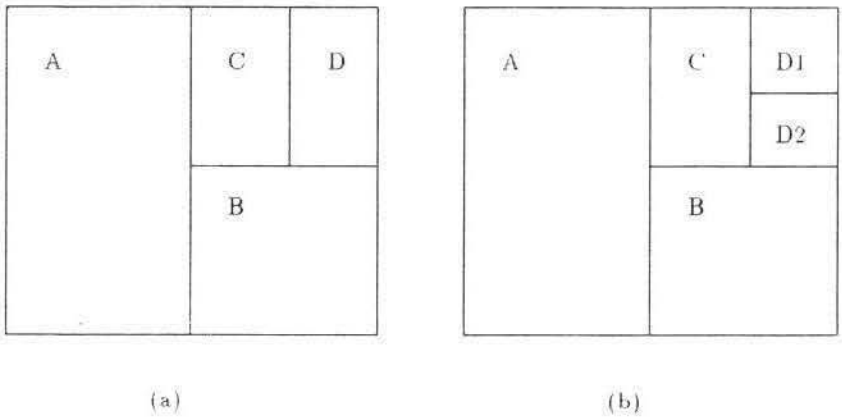


Figura 2.31: Exemplo da partição do diretório em um DYOP Grid File após o *bucket* referenciado por D (a) sofrer uma operação de *split* (b).

na figura 2.31b, por exemplo, poderia ser construído a partir da inserção de todos os dados no espaço referenciado pelo *bucket* D. Nesse caso, os demais componentes do diretório seriam vazios e representados por entradas nulas. Como conseqüência, as operações sobre o diretório teriam seu desempenho degradado pela estrutura de árvore utilizada para representá-lo. Os níveis internos da estrutura representariam a partição do espaço, e como os dados distribuídos não uniformemente ocasionam distribuição de partições não uniformes, a estrutura tenderia a ser degenerada.

Freeston [Fre87] propõe um novo tipo de Grid File denominado BANG File. Diferente dos dois Grid Files apresentados acima, o BANG FILE permite que a divisão do componente do diretório em dois subespaços gere regiões não disjuntas sob a seguinte condição: se a intersecção dos subespaços resultantes não é vazia, então um subespaço contém o outro. Essa condição permite a execução de uma rotina para prover uma melhor redistribuição desses componentes. Associado a isso, o algoritmo de partição é mais flexível do que no Grid File e DYOP File. Sempre que a inserção de um dado no *bucket* causar um estouro de sua capacidade de armazenamento, o componente do diretório associado a este *bucket* será particionado até que as regiões resultantes possuam a mesma quantidade de elementos. Essa característica provê uma melhor adaptação do BANG File quando a distribuição dos dados muda. A associação dos pontos no espaço para sua respectiva região no

diretório é feita por funções *hash*.

Um outro método de acesso denominado Buddy Hash Tree foi proposto por Seeger [SK90]. Esse método de acesso incorpora as características do método R-Tree e Grid File, contudo, é fundamentalmente diferente de cada um deles. A distinção básica para os demais métodos discutidos acima é que os espaços de dados vazios não são particionados. Semelhante ao BANG File, o método Buddy Hash Tree pode ser definido como um esquema de *hash* dinâmico com uma estrutura de árvore de diretório.

Kriegel [KSSS89] apresenta um estudo comparativo do comportamento de alguns desses métodos (Grid File, Bang File, Buddy Hash Tree) sobre sete distribuições de dados. Nesse estudo, o método Buddy Hash Tree apresenta um melhor desempenho geral para as operações de inserção e consulta. Kriegel identificou que um dos fatores importantes para o bom desempenho do Buddy Hash Tree foi o fato de não particionar o espaço de dados completamente. Kriegel inclusive observa que é interessante incorporar esta condição a outros métodos, em particular, sugere a análise do comportamento do método BANG File sob essa condição.

Nessa seção será detalhado o funcionamento do método de acesso Grid File como proposto originalmente em [NHS84]. Embora os experimentos de Kriegel não tenham apontado esse método como robusto para distribuições arbitrárias de dados, para dados distribuídos uniformemente este método foi superior aos demais. A escolha desse método a ser apresentado é conseqüência da larga citação deste na literatura sendo inclusive, algumas vezes, utilizado como medida de desempenho para os demais métodos similares.

2.4.1 Grid File

O Grid File [NHS84] é uma estrutura projetada para gerenciar espaço de armazenamento em disco em unidades de tamanho fixo denominadas *buckets*. Estes *buckets* são de quantidade ilimitada com capacidade para armazenar tantos registros quanto for o tamanho configurado por página (normalmente entre 10 e 1000).

Esta estrutura tem como principal objetivo:

- garantir o princípio de dois acessos a disco para consultas que visam determinar se um ponto está ou não armazenado na estrutura.
- permitir um processamento eficiente para *range query*.

Para isso a estrutura é composta de duas partes: *grid directory* e *buckets*. O *grid directory* é organizado da seguinte forma:

- um vetor dinâmico em k dimensões (*Grid Array*), sendo k a dimensão em que os dados são representados. Cada célula desse vetor faz referência a um *bucket* e é denominada *grid block*.
- k vetores unidimensionais denominados *scales*. Estes *scales* definem a partição do domínio de cada atributo, ou melhor, de cada dimensão. Através destes, pode-se fazer acesso ao *grid block* apropriado.

Ao *grid directory* são impostas algumas condições:

- para evitar *buckets* com baixa taxa de ocupação, alguns *grid block* podem compartilhar um *bucket*. Este conjunto é denominado de *bucket region*.
- cada *bucket region* deve possuir a forma de um retângulo k dimensional.
- existe uma correspondência de um para um entre o *grid* definido pelos *scales* e o *grid array*.

A figura 2.32 ilustra a organização do Grid File, de capacidade de dois registros por *buckets*, representando os objetos de dados da figura 2.8. O *grid array*, normalmente grande, é mantido na memória secundária. Em contrapartida, os *scales* de proporções bem menores podem ser mantidos na memória principal. Dessa forma, observa-se garantido o princípio de 2 acessos a disco. O primeiro a fim de recuperar o *grid block* armazenado no disco através dos *scales*, e o segundo para recuperar o *bucket* contendo os registros através da referência do *grid block*. O cálculo para determinar o *grid block* a ser recuperado é feito na memória principal através das partições do espaço armazenada nos *scales*.

Inserção

Para inserção de elementos na estrutura Grid File deve-se primeiro encontrar o *bucket* apropriado. Através das coordenadas do dado a ser inserido, os *scales* devem ser pesquisados com o objetivo de determinar o *grid block* que contém a referência ao *bucket* apropriado.

A figura 2.32 ilustra o caminho para inserção de um objeto de dados com coordenadas $x = 90$ e $y = 50$. Neste exemplo, o valor de x encontra-se no intervalo entre o terceiro e o quarto elemento do vetor *scale-x*, enquanto o valor de y encontra-se entre o segundo e o terceiro elemento do vetor *scale-y*. A associação destes índices nos *scales* levam ao *grid block* W.

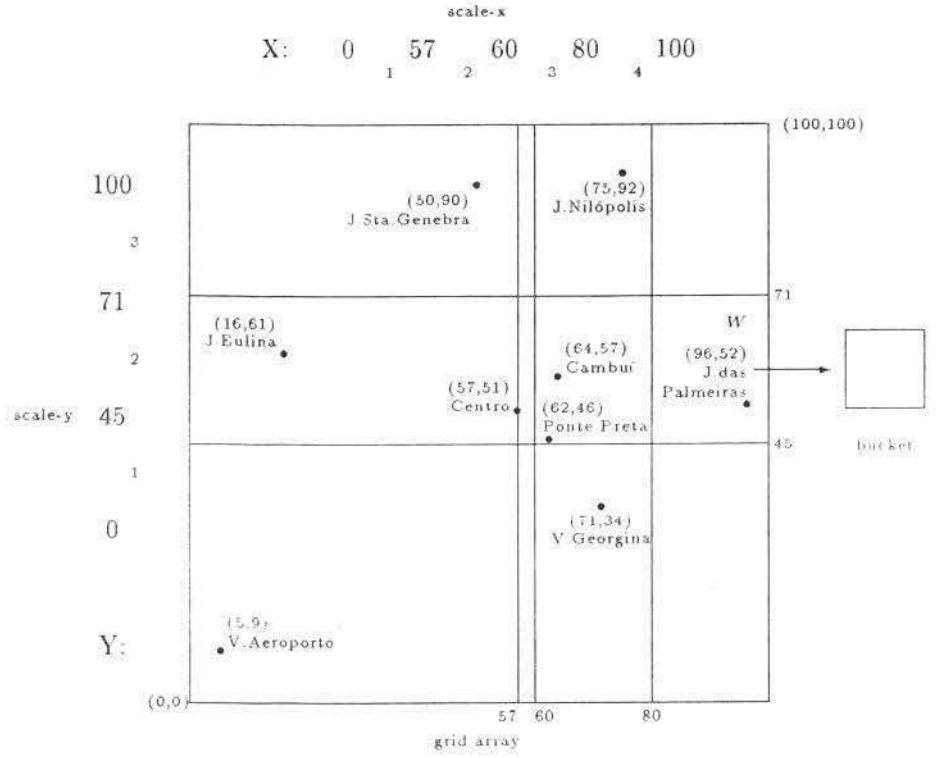


Figura 2.32: Um exemplo da estrutura Grid File.

Após a identificação do *bucket* associado ao dado a ser inserido. Este *bucket* deve armazenar o novo objeto de dados. Contudo, se o *bucket* já se encontra 100% ocupado (possui tantos registros quanto foi configurado), deve sofrer uma operação de *split* dando origem a um novo *bucket*.

Para executar a operação de *split* deve-se primeiro verificar a *bucket region* que faz referência ao *bucket* que será particionado. Se esta *bucket region* cobre somente um *grid block*, então o *grid array* deve ser estendido por um hiperplano $(k-1)$ dimensional que corta essa *bucket region* em duas. Isto é alcançado através da inserção de um novo limite em um dos *scales*, pois com o objetivo de manter a correspondência de um para um entre o *grid* definido pelos *scales* e o *grid array*, um hiperplano $(k-1)$ dimensional deve ser inserido no *grid array*. Caso a *bucket region* cubra mais de um *grid block*, então já existe um hiperplano que corta essa *bucket region* em duas.

Após a determinação das duas *bucket regions*, seja pela inserção de um plano ou pela partição de uma *bucket region* já existente, um novo *bucket* é criado com o objetivo de redistribuir os elementos armazenados no *bucket* que estava 100% ocupado. Estas duas *buckets regions* são então assinaladas para estes dois *buckets*.

A figura 2.33 ilustra a operação de *split* descrita acima ocorrida pela inserção de um elemento no *bucket A* e de outro elemento no *bucket C*. Neste exemplo são apresentadas as duas possibilidades já comentadas, com o *bucket A* referenciado por uma *bucket region* que cobre apenas um *grid block* e o *bucket C* referenciado por uma *bucket region* que cobre dois *grid blocks*.

A operação de *split* necessita da especificação de dois parâmetros: a dimensão em que a partição será feita; e a coordenada, naquela dimensão, onde se fará a partição.

Nievergelt [NHS84] propõe que a escolha da dimensão seja escalonada de forma cíclica, e sugere que para determinadas aplicações que possuam consultas parcialmente especificadas (informação de coordenadas de certas dimensões), estas dimensões mais requisitadas sejam particionadas com uma freqüência maior do que as outras, melhorando assim a precisão para as consultas. Quanto a escolha da coordenada, sugere o ponto mediano desse intervalo. Hinrichs [Hin85] apresenta outras políticas para escolha dos fatores determinantes da operação de *split*.

A figura 2.34 ilustra a seqüência inicial da construção do Grid File da figura 2.32 através da inserção dos nós: Cambuí (A); V. Georgina (B); J. das Palmeiras (C); Centro (D); V. Aeroporto (E); J. Sta. Genebra (F); e J. Nilópolis (G). A correspondência dos *grid blocks* com *buckets* é feita através

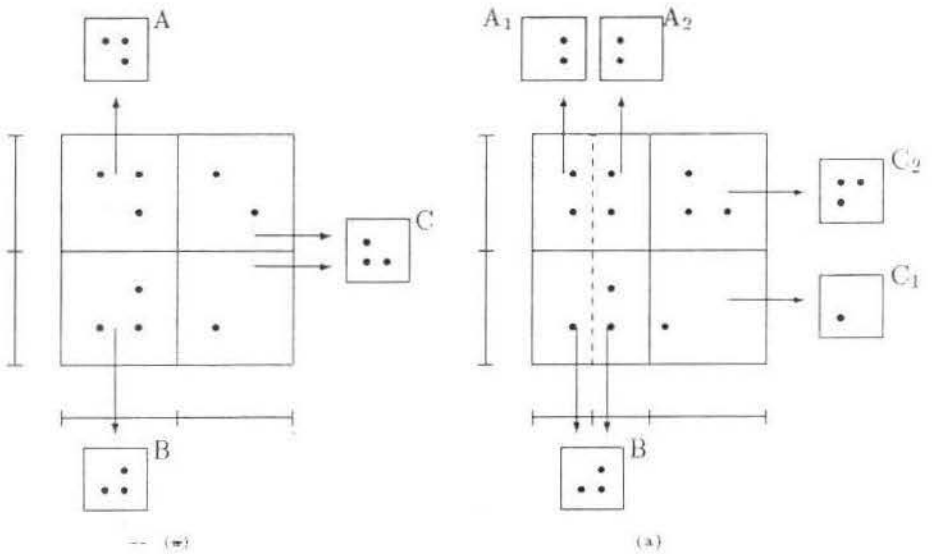


Figura 2.33: Um exemplo da operação de *split* em um Grid File com taxa de ocupação de 3 registros por *bucket*, após a inserção de um registro em A e outro em C.

dos caracteres numéricos dentro dos círculos.

Remoção

Na operação de remoção, de maneira análoga à operação inserção, deve-se primeiro identificar o *bucket* associado ao elemento a ser removido.

Após a identificação do *bucket* e remoção do elemento, se a taxa de ocupação do *bucket* (número de registros existentes / capacidade do *bucket*) for baixa, este poderá ser unido a um outro *bucket* próximo de taxa igualmente baixa. Esta operação é denominada *merge bucket*[13].

A operação de *merge* é inversa ao *split*. Assim, dois *buckets* adjacentes com baixa taxa de ocupação podem ser unidos em um, desde que essa nova *bucket region* formada por estes dois *buckets* possua a forma de um retângulo k dimensional. Se o limite do *scale* associado à dimensão em que os dois *buckets* se uniram não se faz mais necessário, este é então removido do *scale*. Deste modo, para se manter a correspondência de um para um entre o *grid* definido pelos *scales* e o *grid array*, o hiperplano associado a esse limite é removido.

A política para se efetuar um *merge bucket* é controlada por 3 decisões: que pares de *buckets* adjacentes são candidatos para a junção; quando for possível executar o *merge* entre vários eixos (dimensões), qual o mais favorável; qual a taxa de ocupação máxima a ser permitida em um *bucket* quando este sofrer a operação de *merge*.

Como atribuição para estas 3 questões acima, diferentes políticas são apresentadas. Nievergelt [NHS84] apresenta para o primeiro ponto acima duas estratégias: *buddy*[14] *system*, onde é permitido a um dado *bucket* sofrer a operação de *merge* com exatamente um vizinho em cada dimensão (com a sua respectiva região *buddy*); *neighbor system*, que permite a um dado *bucket* sofrer a operação de *merge* com seus dois vizinhos adjacentes em cada uma das k dimensões. No segundo critério, somente importante quando a estratégia de *split* favoreceu determinada dimensão, a operação de *merge* não deve desfazer o efeito do *split* fazendo mais junções nesta dimensão do que nas outras. Para o terceiro ponto é estipulado uma taxa de ocupação de 70%. Hinrichs [Hin85] apresenta outras políticas, bem como comenta e justifica os resultados obtidos com esta.

---

[13] junção.

[14] quando uma região é particionada, as duas regiões resultantes são denominadas *buddies* uma da outra.

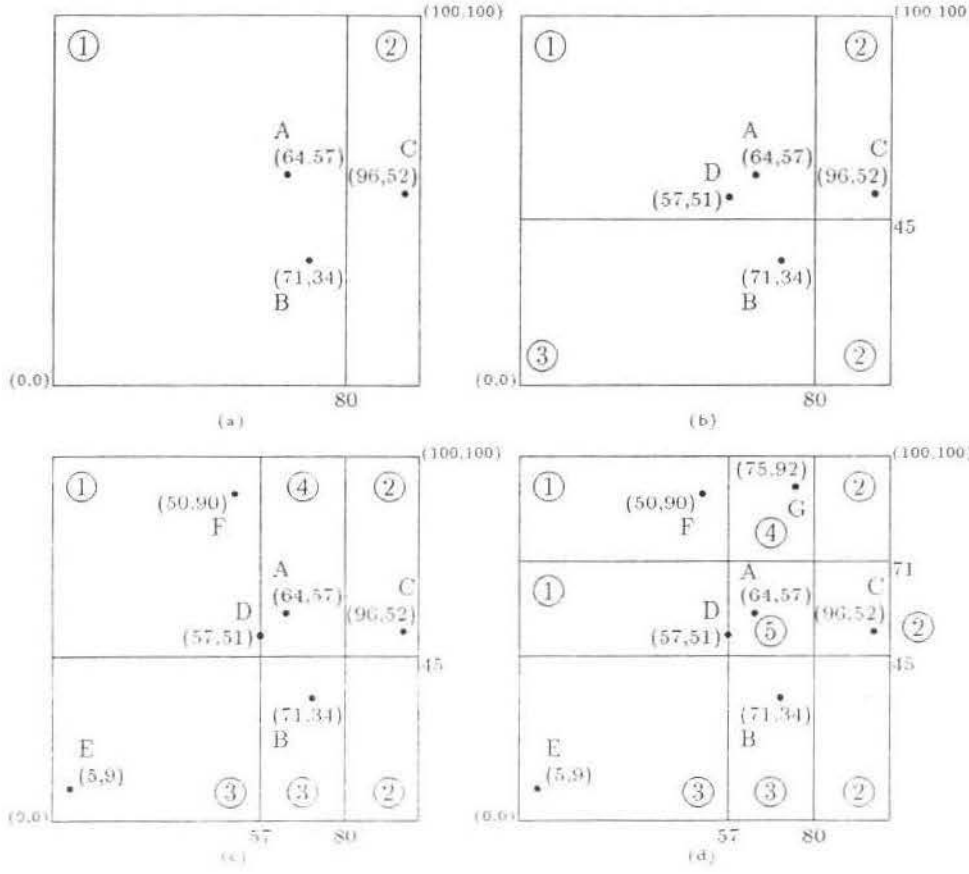


Figura 2.34: Seqüência das partições parciais do *grid* pela inserção dos elementos: Cambuí (A), V. Georgina (B), J. das Palmeiras (C), Centro (D), V. Aeroporto (E), J. Sta. Genebra (F) e J. Nilópolis (G).

Consulta

Por definição o Grid File realiza consultas do tipo *point queries* com dois acessos a disco. Através das coordenadas do ponto a ser consultado e dos *scales* armazenados na memória principal, consegue-se identificar qual o *grid block*, no *grid array*, referente à porção do espaço a ser pesquisada. Após a leitura desse *grid block* (primeiro acesso a disco) obtém-se o endereço do seu respectivo *bucket* associado. A leitura desse *bucket* para examinar os dados armazenados constitui o segundo acesso a disco.

A organização em que os dados estão mantidos nos *buckets* não é tão importante, já que a pesquisa destes é feita na memória principal. A idéia principal da estrutura Grid File é justamente como organizar o conjunto de *buckets*, ou melhor, como gerenciar o *grid directory*, responsável pela correspondência entre *grid blocks* e *buckets*.

Hinrichs e Nievergelt discutem a eficiência do Grid File na detecção de *range queries* (determinação de relacionamentos espaciais sobre uma *window* de consulta). Nievergelt [NHS84] afirma que devido às características da estrutura (associação de retângulos K-dimensionais a páginas de disco), há uma grande possibilidade de que as windows de consulta estejam em um mesmo *grid block*. Hinrichs [HN83, Hin85] afirma que mesmo que o espaço a ser consultado sobreponha o espaço representado por demais *grid blocks*, todo o esforço para detectar esta sobreposição é determinado na memória principal através dos *scales*. Dessa forma, os acessos a disco são somente necessários para recuperar o conjunto de dados que satisfazem à consulta. Assim, Hinrichs conclui que *range queries* são também suportados eficientemente uma vez que a determinação do *grid block* associado ao espaço de consulta não causam nenhum acesso a disco.

Considerações

O Grid File detecta tanto *point queries* bem sucedidas como não, com apenas dois acessos a disco. Esta é uma das grandes vantagens que apresenta sobre as demais estruturas que são baseadas em áreas de armazenamento que permitem estouro (*overflow buckets*, onde uma consulta mal sucedida normalmente demanda mais tempo do que a média gasta para consultas bem sucedidas.

O Grid File é um método de acesso para representação de pontos no espaço. Contudo, [HN83] sugere a utilização do Grid File para representação de outros objetos espaciais de formas irregulares. Nesses casos, são empre-

gadas técnicas para transformação desses objetos em pontos no espaço de dimensionalidade maior. Para dados no espaço bidimensional, por exemplo, a estratégia MBR pode ser utilizada para abstrair um objeto qualquer através de um retângulo, seguido da transformação deste em um ponto no espaço de 4-dimensões (c_x, d_x, c_y, d_y), onde c_i e d_i representam o centro e a metade do comprimento do retângulo na dimensão i.

Nievergelt [NHS84] apresenta detalhadamente as características da estrutura Grid File e discute algumas estratégias de *split* e *merge* em sua simulação. Conclui que o Grid File é uma estrutura projetada para manipular eficientemente uma coleção de registros cujos campos chaves estejam representados em um espaço de dimensionalidade modesto (menor que 10).

Hinrichs [Hin85] propõe uma forma de implementação do *Grid Directory* diferente da proposta por Nievergelt [NHS84]. De fato, devido ao *grid array* ser armazenado em disco e, conseqüentemente ocupando várias páginas de disco, as operações básicas executadas, *neighborhood* (encontrar o vizinho de uma dado *grid block* em uma dada dimensão), *split* (inserir um limite nos *scales* e conseqüentemente inserir um hiperplano (k-1) dimensional no *grid directory*) e *merge* (remover um limite nos *scales* e um hiperplano (k - 1) dimensional no *grid directory*), normalmente requisitam vários acessos a disco para serem concluídas. As diversas soluções propostas para organizar esse *Grid Directory* têm o objetivo de diminuir o número de acessos a disco na execução dessas operações. Entretanto, estas soluções não atendem eficientemente a todas as operações simultaneamente.

Dessa forma, Hinrichs propõe a idéia de dividir o *grid directory* em vários *grid directory* menores armazenados cada um destes em uma página de disco. O gerenciamento desses *grid directory* é feito por um outro *grid directory* específico, denominado *root directory*. O *root directory* possui uma resolução menor e tem como objetivo apenas direcionar qual *grid directory* deve ser recuperado no disco. A inserção de outro nível de diretório não vai de encontro ao princípio de dois acessos a disco, pois devido ao seu tamanho, este diretório é armazenado na memória principal. As operações básicas citadas anteriormente podem ser executadas eficientemente já que cada *grid directory* está armazenado em página de disco.

2.5 Derivados de *Multiway Trees*

Como já abordado, o gerenciamento de dados em disco através de árvores binárias ou de suas variantes (Quadtree por exemplo) não se faz de forma adequada. A unidade de leitura dessas estruturas (nós) ocupa uma quantidade de espaço bem inferior da existente nas páginas de disco. Dessa forma, árvores com largos *fan-out* são mais apropriadas uma vez que o percurso nas estruturas binárias causam várias *page faults*.

Contudo, se os nós forem armazenados nas páginas de disco de modo que essencialmente a estrutura da árvore mude de binária para multi-ária, conforme na figura 2.35, o percurso nessa estrutura é otimizado em função do melhor aproveitamento das páginas de disco. Esta estratégia, denominada por Knuth [Knu73b] de *multiway trees*, deu origem a uma estratégia de pesquisa baseada em índices externos desenvolvida por Bayer e McCreight [BM72] a partir de uma estrutura de dados denominada B-Tree [BM72, Com79].

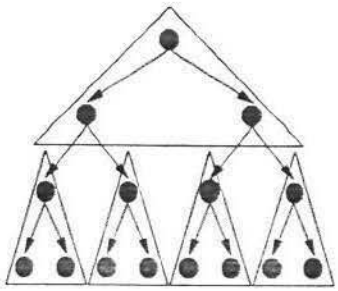


Figura 2.35: Ilustação da estratégia *multiway trees*.

Essa estrutura pode ser caracterizada como uma generalização de uma árvore binária de pesquisa, onde a partir de um nó qualquer, existem mais do que dois caminhos possíveis a serem percorridos. Embora essa característica possa ser obtida através da implementação de árvores binárias como *multiway trees* ([Ooi90] utiliza essa estratégia para Spatial K-d Tree), a B-Tree acrescenta a característica de ser uma estrutura balanceada (uma árvore é dita balanceada quando a profundidade da subárvore esquerda de cada nó nunca difere de ± 1 da profundidade de sua subárvore direita). Ao contrário das árvores binárias discutidas na seção 2.1, a B-Tree suporta inserção e

remoção de elementos sem degradar o desempenho da estrutura e, conseqüentemente, sem a necessidade de nenhuma rotina para reorganização.

As características das árvores de múltiplos caminhos (*multiway trees*), em particular a B-Tree, tornaram-na um padrão para organização de arquivos. Assim, diversos métodos de acesso a dados espaciais surgiram tendo como estrutura de dados, extensões de B-Trees.

Um dos primeiros métodos de acesso com essas características surgiu através da K-D-B Tree de Robinson [Rob81] para manipulação de pontos no espaço. Embora atualmente este método não seja utilizado em conseqüência da pouca flexibilidade na representação de dados e do baixo desempenho, quando comparado a outros métodos mais recentes [Gre89], muitas de suas características foram incorporadas em outros métodos desenvolvidos posteriormente.

Praticamente, o grande desenvolvimento de métodos de acesso derivados de *multiway trees* emergiu com a R-Tree, proposta por Guttman [Gut84], para manipulação tanto de pontos como de objetos espaciais através da estratégia MBR. A simplicidade e flexibilidade dessa estrutura motivou o desenvolvimento de outros métodos variantes como a R^+-Tree de Sellis [SRF87] e a R^+-Tree de Beckmann [BKSS90].

Uma outra tendência no desenvolvimento de métodos derivados de B-Tree tem se iniciado com a Cell Tree, proposta por Gunther [Gun88, Gun89]. Nas aplicações onde os objetos de dados não são representados pelo seu próprio intervalo (figura 2.36), como em aplicações de robótica e visão, a representação desses através de MBRs requer uma maior complexidade nas rotinas de consulta em conseqüência da necessidade de evitar a determinação de relacionamentos espaciais entre *dead space*. Conforme Gunther88 [Gun88], a Cell Tree tem se mostrado um método de acesso eficiente para estas aplicações, em decorrência da maior precisão na representação dos objetos de dados, por meio da estratégia *region decomposition* (em particular, a metodologia utilizada pela Cell Tree decompõe os objetos de dados em conjuntos de poliedros regulares convexos).

Nessa seção serão apresentados os métodos R-Tree e suas variantes, como também o método de acesso Cell Tree.

2.5.1 R-Trees

O método de acesso R-Tree [Gut84] tem a finalidade de representar objetos por intervalos em várias dimensões. A idéia dessa estrutura é de permitir a recuperação de objetos eficientemente, de acordo com as suas localizações

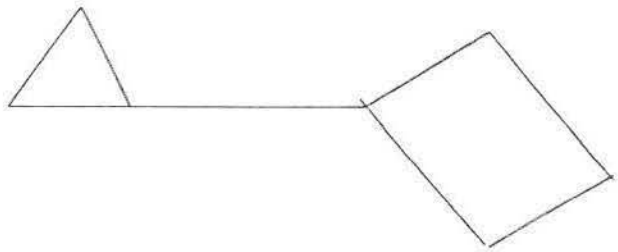


Figura 2.36: Ilustração da representação de dados complexos através da estratégia *region decomposition*.

espaciais.

A R-Tree é uma estrutura hierárquica e dinâmica, derivada de uma B^+-Tree [Com79]. Dessa forma, esta estrutura possui dois tipos de nós a serem descritos abaixo:

- nós folhas - composto por um conjunto de entradas do tipo

  (*rect*, *tuple-id*)

  onde:

  - **tuple-id** é o identificador do objeto usado para referir-se a uma tupla no Banco de Dados.
  - **rect** representa o retângulo de k dimensões cujos limites contém o objeto de dados referido por *tuple-id*.

- nós intermediários - composto por um conjunto de entradas do tipo

  (*rect*, *child-pointer*)

  onde:

  - **child-pointer** contém o apontador para a subárvore hierarquicamente subordinada a esta entrada.
  - **rect** representa o menor retângulo de k dimensões que contém todos os retângulos de todas as entradas do nó inferior a este hierarquicamente, ou seja, do nó para o qual essa entrada faz referência através de *child-pointer*.

As regras básicas para formação de uma R-Tree são semelhantes as regras para formação de uma B^+-Tree. Assim, seja M o número máximo de entradas por nó e $m = M/2$ o número mínimo, tem-se as seguintes restrições:

1. cada nó folha possui entre m e M apontadores para registros, a menos que seja raiz.
2. para cada nó folha da forma (*rect*, *tuple-id*), *rect* é o menor retângulo que espacialmente contém o objeto de dados multidimensional representado por *tuple-id*.
3. cada nó intermediário possui entre m e M filhos, a menos que seja raiz.
4. para cada nó intermediário da forma (*rect*, *child-pointer*), *rect* é o menor retângulo que espacialmente contém os retângulos do nó filho.
5. o nó raiz possui pelo menos dois filhos, a menos que seja folha.
6. todas folhas aparecem no mesmo nível.

Na figura 2.37 é apresentado a R-Tree correspondente aos dados da figura 2.9 com o valor de M e m iguais a 4 e 2 respectivamente. Através desta figura, pode-se verificar que a R-Tree permite a sobreposição dos retângulos dos nós intermediários. Essa característica, que será discutida posteriormente, é indesejável visto que degrada o desempenho da estrutura para consultas.

Antes de prosseguir a apresentação da estrutura, é necessário definir dois conceitos associado com a técnica de representação utilizada por esse método de acesso, que também serão utilizadas na apresentação dos demais métodos dessa seção.

- o MBR de um nó corresponde ao menor retângulo cujos limites contém todos os retângulos das entradas desse nó.
- o MBR de uma entrada corresponde ao retângulo representado pelo campo *rect* dessa entrada

Inserção

A inserção de novos elementos de dados em uma R-Tree é semelhante a inserção em uma B^+-Tree, no aspecto que: os registros são inseridos nas folhas da estrutura; se o nó folha à armazenar o novo elemento já possuir M

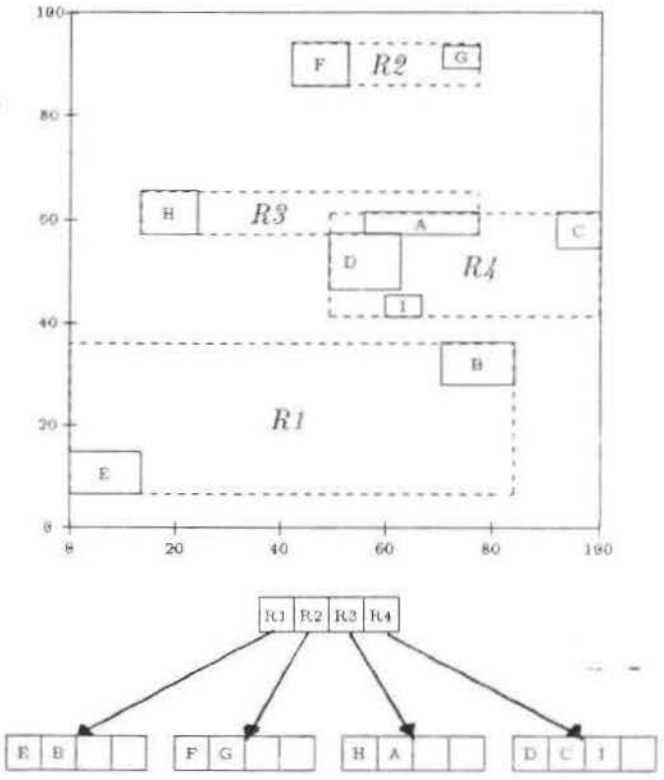


Figura 2.37: Uma R-Tree e os retângulos que esta representa.

entradas, este ocasionará um *overflow*[15]; nós que sofrem *overflow* devem ser particionados (operação de *split*); e uma vez ocorrido uma operação de *split* em um nó folha, esta poderá ser propagada até a raiz.

Assim, para a inserção de um dado na estrutura deve-se percorrer por esta até que se chegue a um nó folha onde o novo registro será inserido. Esse percurso é governado da seguinte forma. Em um dado nó intermediário, com o objetivo de determinar qual das entradas a que será utilizada para direcionar o percurso, escolhe-se aquela cujo respectivo MBR seja o que sofrer o menor aumento de sua área, caso o dado seja inserido em um dos nós folhas subordinados a esta entrada.

Após encontrado o nó folha, o novo objeto de dados deve ser inserido nesse nó. A partir desse ponto, analogamente a uma B^+-Tree, tem-se duas possibilidades: se o nó possui capacidade para armazenar o novo dado (existem menos que $M - 1$ entradas) ou se o nó não possui capacidade para armazená-lo (existem M entradas). No primeiro caso, quando o nó possui menos que $M - 1$ entradas, o dado é armazenado neste nó. Essa inserção pode causar uma alteração no MBR desse nó, fazendo com que seja necessário percorrer o caminho inverso ao utilizado para se chegar até este, atualizando os MBRs das entradas envolvidas nesse percurso. Esse processo é necessário, pois como o MBR de cada entrada reflete o MBR do nó sucessor a esta, uma vez que a inserção de um novo elemento causou alteração no MBR do seu respectivo nó folha, esta alteração deve ser propagada aos níveis antecessores.

A segunda possibilidade, em que o nó já possui M elementos, a inserção do novo dado ocasionará uma operação *split* desse nó. Essa operação é responsável por criar um novo nó e redistribuir os $M+1$ elementos entre esses dois. Após essa redistribuição, o MBR desses dois nós deve ser calculado. Analogamente ao caso anterior, também é necessário percorrer em direção a raiz atualizando o MBR das entradas envolvidas no percurso. Ao mesmo tempo que se faz o percurso de sentido folha-raiz, é feita a avaliação sobre a propagação da operação de *split* gerada nas folhas. Com a criação de um novo nó, é necessário criar no nível anterior a este uma nova entrada que será a entrada pai desse novo nó folha. A inserção dessa nova entrada em um nó intermediário é tratada de forma semelhante a inserção de uma entrada na folha da estrutura.

A redistribuição dos $M + 1$ elementos nos dois nós, ocorrida na operação de *split*, é de grande importância para o refinamento da estrutura R-Tree.

[15] estouro da capacidade máxima de armazenamento

Como será apresentado, dois fatores relacionados com o MBR dos nós podem degradar o desempenho da estrutura, *coverage*[16] e *overlap*[17]. Uma boa política para a operação de *split* pode atender da melhor forma possível a estes fatores: tornar zero o *overlap*; e minimizar ao máximo o *coverage*. Embora a situação ideal fosse o atendimento desses dois fatores simultaneamente, em algumas condições, esse objetivo torna-se inalcançável. A figura 2.38 ilustra duas possibilidades para redistribuição dos nós da figura (a): através da figura (b), onde o *coverage* dos nós resultantes foi reduzido, porém, com *overlap*; e através da figura (c), onde não foi gerado *overlap* dos nós, porém, o *coverage* destes permaneceu extenso. Dessa forma, para solucionar esse impasse torna-se necessário o estabelecimento de prioridades entre esses objetivos. Guttman [Gut84] propõe três algoritmos diferentes para a operação de *split* tendo como prioridade a diminuição do *coverage* (estratégia apresentada na figura 2.38b).

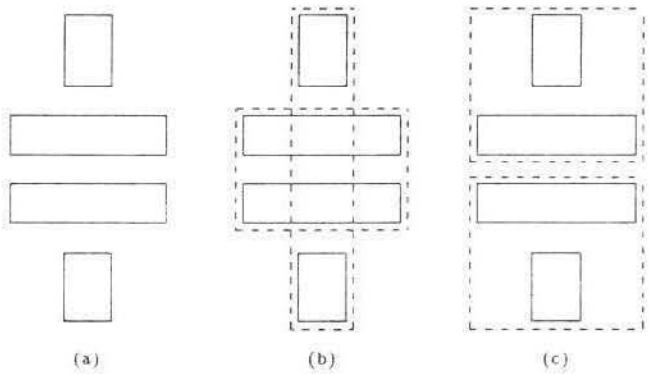


Figura 2.38: Apresentação de duas políticas para a operação de *split*, visando diminuir o *coverage* (b), e o *overlap* (c).

Na figura 2.39 é ilustrado o processo de inserção para a construção da R-Tree apresentada em 2.37, através da seqüência dos retângulos: Cambuí (A), V. Georgina (B), J. das Palmeiras (C), Centro (D), V. Aeroporto (E), J. Sta. Genebra (F), J. Nilópolis (G), J. Eulina (H) e Ponte Preta (I).

[16] *coverage* refere-se a área total representada por um ou mais MBRs.
[17] *overlap* refere-se a área comum representada por dois ou mais MBRs.

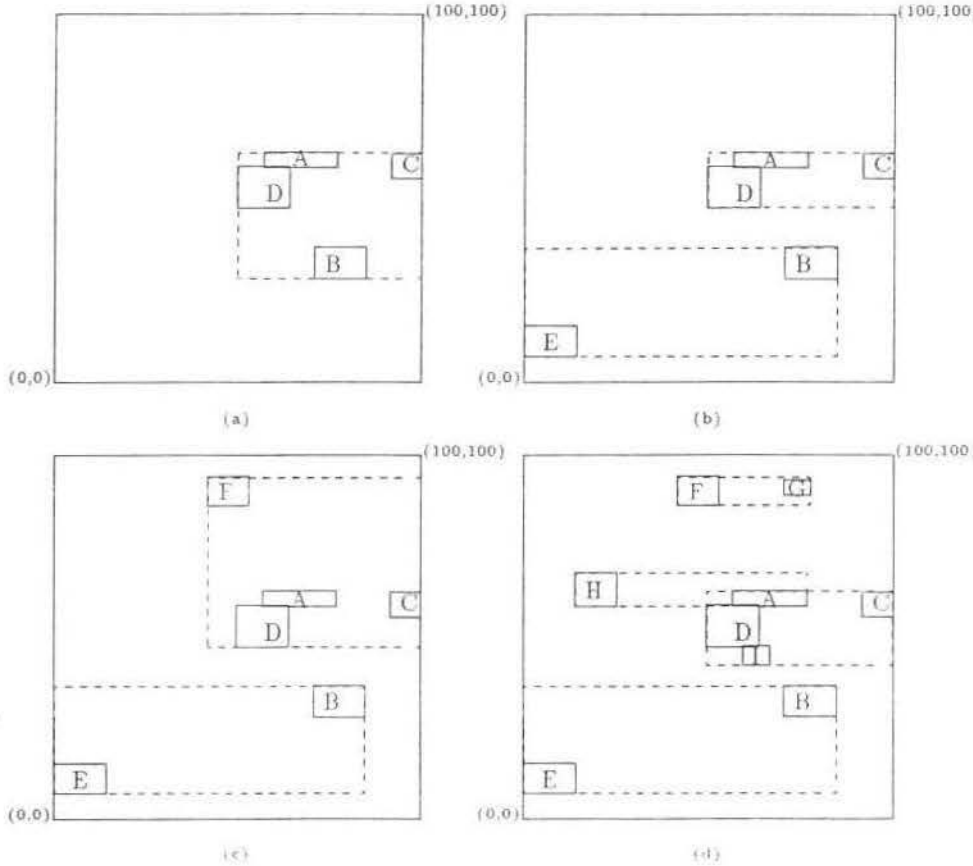


Figura 2.39: Demonstração de inserções em uma R-Tree através dos elementos: Cambuí (A), V. Georgina (B), J. das Palmeiras (C), Centro (D), V. Aeroporto (E), J. Sta. Genebra (F), J. Nilópolis (G), J. Eulina (H) e Ponte Preta (I).

Remoção

Para a remoção de um retângulo em uma R-Tree deve-se primeiro determinar em qual nó folha se encontra o dado a ser removido. O percurso na estrutura para se encontrar tal elemento é feito da seguinte forma. A partir do nó raiz, todas as entradas são verificadas até encontrar dentre estas aquelas cujo respectivo MBR sobrepõe área correspondente ao retângulo a ser removido. Para cada entrada que satisfizer essa condição, sua respectiva subárvore será percorrida, de maneira análoga à apresentada para inserção, até encontrar ou não em um nó folha da estrutura o objeto de dados a ser removido.

Uma vez encontrado o nó que contém o retângulo a ser removido, esta entrada é eliminada resultando em duas possibilidades: nó com m ou mais entradas; ou nó com $m-1$ entradas. A primeira possibilidade, em que o nó possui m ou mais entradas, deve-se calcular o novo MBR desse nó e de maneira análoga a inserção sem *split*, percorrer a estrutura no sentido folha-raiz pelo mesmo caminho utilizado para se chegar à folha, alterando, se necessário, o MBR das entradas envolvidas nesse percurso.

Na segunda possibilidade, onde o nó possui $m-1$ entradas, um *underflow* desse nó é gerado. Essa operação de *underflow* é responsável por eliminar esse nó alocando as $m-1$ entradas restantes a um conjunto U. A eliminação desse nó folha implica na remoção da entrada referente a este em um nível anterior. A remoção dessa entrada no nó intermediário é executada da mesma forma que a remoção em um nó folha, tornando assim esse processo recursivo. A propagação do *underflow* através do percurso de sentido folha-raiz é feita ao mesmo tempo em que o MBR das entradas envolvidas nesse percurso são recalculadas. Após terminado esse processo, todas as entradas armazenadas no conjunto U, definido anteriormente, são reinseridas.

O algoritmo de remoção de um elemento em uma R-Tree difere do algoritmo em uma B^+-Tree, à medida que a operação de *underflow* não causa a redistribuição das entradas desse nó com um outro nó vizinho. Na R-Tree, as entradas de nós vizinhos não implicam que seus objetos de dados sejam adjacentes. Dessa forma, a redistribuição pode ocasionar com que os MBRs desses nós aumentem de tamanho propagando essas áreas aos nós antecessores e, por fim, comprometendo o desempenho da R-Tree.

A idéia de reinserção de Guttman possui dois fatores favoráveis: primeiro, o fato do algoritmo de inserção causar um refinamento na estrutura; segundo, porque as páginas a serem utilizadas no processo de reinserção provavelmente serão as mesmas que as visitadas para se encontrar o nó removido, e por já estarem na memória, não causarão demais acessos a disco.

Consulta

As características da R-Tree quanto à forma que representa seus dados a tornam apropriada para detecção do tipo *range queries* (quais os objetos que sobrepõem uma dada área, quais os objetos que possuem área comum a uma dada área).

Entretanto, a determinação desses relacionamentos pode ser degradada em virtude da possibilidade de sobreposição dos MBRs das entradas de um nó intermediário. Essa área comum às demais entradas, *overlap*, é indesejável para R-Tree, pois tanto para consultas como remoções, se o elemento a ser pesquisado ou removido estiver nessa área, a partir desse nó intermediário, tem-se que percorrer todas as subárvores cujos respectivos MBRs sobreponham essa área. Na figura 2.40 é ilustrada uma consulta de uma área denominada *J*. Verifica-se neste exemplo, que a partir da raiz há a necessidade de se percorrer todas as suas subárvores com o objetivo de se concluir a consulta.

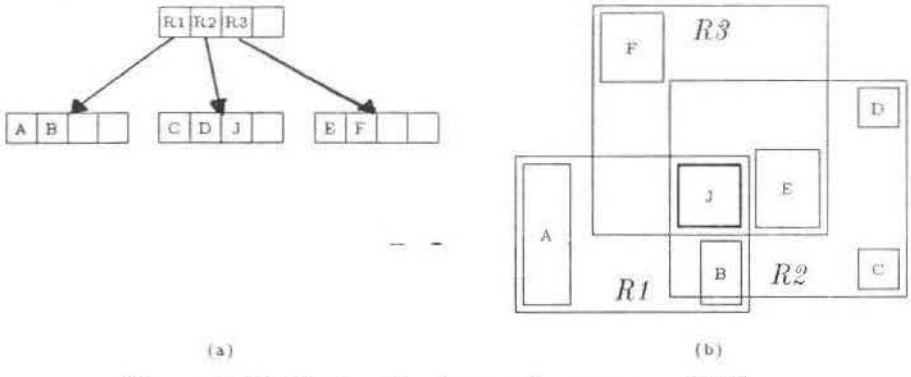


Figura 2.40: Ilustração do *overlap* em uma R-Tree.

Em conseqüência da estrutura de dados ser dinâmica torna-se impossível evitar as sobreposições dos MBRs das entradas nos nós intermediários. Porém, uma metodologia correta para inserção e uma política apropriada para a operação de *split* pode vir a diminuir a quantidade e a área onde existam sobreposições (Roussopoulos e Leifker [RL85] propõem uma técnica diferente para a construção de uma R-Tree se esta for utilizada em um ambiente estático).

Considerações

A profundidade de uma R-Tree com N objetos de dados é no máximo igual a $\lceil \log_m N \rceil - 1$. Esse valor de m, parâmetro da estrutura, refere-se ao número mínimo de entradas por nó em uma R-Tree. Verifica-se que quanto maior for o valor de m menor será a profundidade da estrutura, em contrapartida, maior será o custo para se consultar todas as entradas. Na realidade, essa conseqüência é irrelevante, pois uma vez recuperado um nó, todas as informações contidas neste estarão na memória principal onde o acesso é muito mais rápido do que o acesso a disco.

Com o objetivo de otimizar os acessos a disco a estrutura é configurada a ter tantas entradas quanto for possível armazenar em uma página de disco.

Inserções em R-Trees, desde que não ocasionem *splits*, podem ser satisfeitas em $O(\log_m N)$. Dentre os algoritmos sugeridos por Guttman [Gut84] para a operação de *split*, o que demanda menor complexidade de tempo, aloca os $M + 1$ registros nos dois nós em tempo proporcional a M (O(M)).

Remoções de elementos em uma R-Tree são mais difíceis de serem analisadas. O primeiro passo dessa operação envolve uma consulta com o objetivo de encontrar o elemento a ser removido. Esta consulta pode tornar-se custosa se a área correspondente ao elemento a ser removido é comum aos diversos retângulos associados a um mesmo nó intermediário na estrutura.

Outro fator decisivo para o bom desempenho da R-Tree, além do discutido acima sobre a sobreposição de MBRs dos nós intermediários (*overlap*), é a extensão do *coverage*. Quanto maior for a extensão maior será a possibilidade de haver *overlap*, como também, *dead space* (área sem informação). *Dead space* pode permitir a recuperação de objetos de dados através da determinação de relacionamentos espaciais que na realidade são inexistentes.

2.5.2 R^+-Tree

A R^+-Tree, por Sellis, Roussopoulos e Faloutsos [SRF87, FSRM, Sel], é uma variação da estrutura R-Tree apresentada na subseção anterior. Essa estrutura foi desenvolvida com base na R-Tree tentando aperfeiçoar os pontos, que nessa estrutura, são cruciais para o seu bom desempenho. Zero *overlap* e mínimo *coverage* são condições ideais para o excelente funcionamento de um R-Tree, porém, esses resultados só foram alcançados através da estratégia estática de Roussopoulos [RL85] para a representação de pontos em ambiente estático. Roussopoulos mostra ainda, que para a representação de retângulos, zero *overlap* foi obtido apenas com a divisão de um retângulo de

dados em sub-retângulos. Esse foi o ponto de partida para o desenvolvimento da R^+-Tree.

A idéia principal da R^+-Tree consiste em permitir que os objetos de dados, retângulos, possam ser representados na estrutura por um conjunto de sub-retângulos disjuntos cuja união formem o retângulo original. Assim, é possível manter a estrutura de modo que o MBR de nenhuma entrada sobreponha os demais MBR dessa mesma entrada.

Os nós da R^+-Tree preserva a mesma estrutura dos nós da R-Tree:

- nós folhas - composto por um conjunto de entradas do tipo

  (*rect*, *tuple-id*)

  onde:

  - **tuple-id** é o identificador do objeto usado para referir-se a uma tupla no Banco de Dados.
  - **rect** representa o retângulo de k dimensões cujos limites contém o objeto de dados referido por *tuple-id*.

- nós intermediários - composto por um conjunto de entradas do tipo

  (*rect*, *child-pointer*)

  onde:

  - **child-pointer** contém o apontador para a subárvore hierarquicamente subordinada a esta entrada.
  - **rect** representa o menor retângulo de k dimensões que contém todos os retângulos de todas as entradas do nó inferior a este hierarquicamente, ou seja, do nó para o qual essa entrada faz referência através de *child-pointer*.

As regras para construção de uma R^+-Tree diferem das regras para construção de uma R-Tree. Seja M o número máximo de entradas por nó, a construção da R^+-Tree segue às seguintes restrições:

1. Para cada entrada (*rect*, *child-pointer*) de um nó intermediário, a subárvore relacionada com essa entrada contém um retângulo R, se e somente se, *rect* sobrepõe totalmente o retângulo R. A única exceção é quando R é um retângulo de um nó folha, nesse caso, *rect* pode apenas sobrepor parcialmente o retângulo R.

2. Para quaisquer duas entradas ($rect_1$, *child-pointer*$_1$) e ($rect_2$, *child-pointer*$_2$) de um nó intermediário, o *overlap* entre $rect_1$ e $rect_2$ é zero.

3. A raiz tem pelo menos dois filhos, a não ser que seja uma folha.

4. Todas as folhas estão em um mesmo nível.

A figura 2.41 apresenta a R$^+$-Tree correspondente aos dados da figura 2.9 com o valor de M igual a 4. Nessa figura é possível verificar a diferença entre a R-Tree e a R$^+$-Tree, onde nesta, a sobreposição dos retângulos dos nós intermediários não é permitida.

Inserção

Inserções de elementos em R$^+$-Tree tornam-se mais complexas do que inserções em R-Trees. A R$^+$-Tree tem como restrição, e principal objetivo, evitar que os retângulos associados as entradas dos nós intermediários de um mesmo nível se sobreponham. Dessa forma, a partir do nó raiz, todas as entradas cujos respectivos retângulos sobreponham o objeto de dado a ser inserido devem ser percorridas em direção a suas folhas. O retângulo correspondente ao objeto de dados passa então a ser representado por um conjunto de sub-retângulos que são inseridos nessas folhas.

Para a inserção de objetos de dados nas folhas da estrutura procede-se de maneira análoga a R-Tree. Se o nó possui menos que M entradas, o elemento é inserido, caso contrário uma operação de *split* é gerada. A operação de *split*, necessária por redistribuir as $M + 1$ entradas em dois nós, é feita através de uma partição horizontal ou vertical do MBR correspondente a essas entradas. Em decorrência da necessidade de que o MBR desses dois nós sejam disjuntos, essa política de partição, comparada à da R-Tree, torna-se a melhor opção, pois uma vez determinado a dimensão e a posição em que esse espaço será cortado, tem-se como conseqüência duas sub-regiões disjuntas.

A única conseqüência dessa estratégia adotada para a operação de *split*, é que se a linha de partição aplicada a um dado nível I cortar algum retângulo em seu nível inferior, este também deverá sofrer uma operação de *split*. A possibilidade de propagação do *split* para os níveis inferiores também difere da estratégia utilizada para a R-Tree, onde a operação de *split* é somente propagada aos níveis superiores. Para ilustrar essa possibilidade, na figura 2.42 é apresentada uma R$^+$-Tree em dois níveis intermediários. Na figura 2.42a é definido uma partição para o nível I e na figura 2.42b é apresentado os

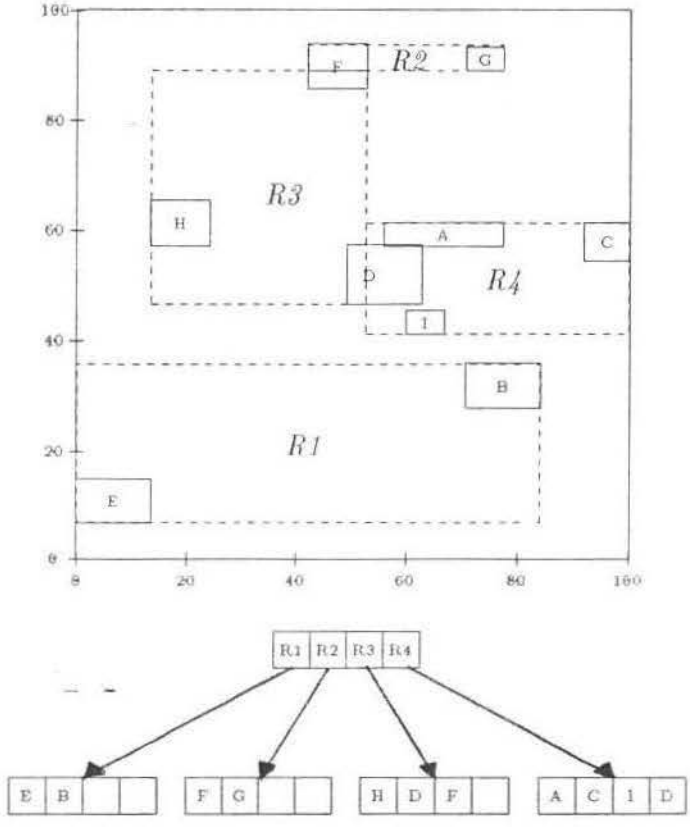


Figura 2.41: Uma R+-Tree e os retângulos que esta representa.

nós resultantes dessa partição. Verifica-se que a figura 2.42b não corresponde mais a uma R+-Tree. O fato da linha de partição interceptar o MBR de um nível inferior obriga a propagação do *split* para este nível com o objetivo de evitar a inconsistência da estrutura.

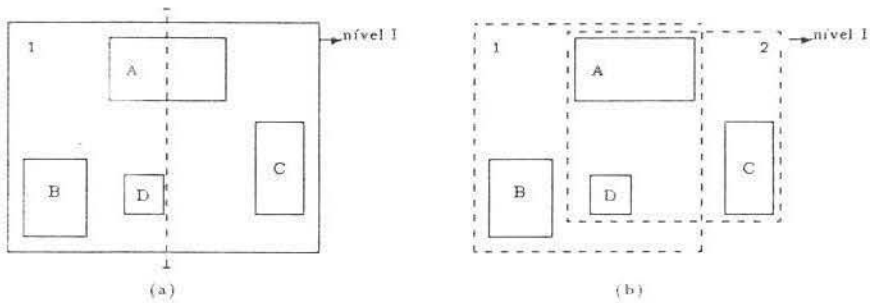


Figura 2.42: Ilustração em (b) a necessidade da propagação do *split* ocorrido em (a).

Como na R-Tree, após a inserção é necessário proceder o percurso inverso para se chegar nas folhas, com o objetivo de avaliar a necessidade de ajustar o MBR das entradas envolvidas na inserção do novo elemento.

A figura 2.43 ilustra o processo de inserção para a construção da R-Tree apresentada em 2.41, através da seqüência dos retângulos: Cambuí (A), V. Georgina (B), J. das Palmeiras (C), Centro (D), V. Aeroporto (E), J. Sta. Genebra (F), J. Nilópolis (G), J. Eulina (H) e Ponte Preta (I).

Remoção

A remoção de elementos em uma R+-Tree é análoga à da R-Tree. O elemento deve ser localizado nas folhas da estrutura para que a sua respectiva entrada seja removida.

Como na R-tree, o processo para remoção de um elemento pode causar diversos percursos pela estrutura, porém com objetivos diferentes. Na R-Tree, esse processo pode ocorrer se o MBR do dado a ser eliminado estiver representado pela área comum aos diversos MBRs dos nós intermediários, na R+-Tree, ocorre se o MBR do dado a ser eliminado sobrepor o MBR de diversos nós intermediários. Foi mostrado que a inserção de um dado na R+-Tree sob essas condições ocasiona a inserção deste nas diversas folhas da estrutura, justificando assim, a necessidade dos diversos percursos para

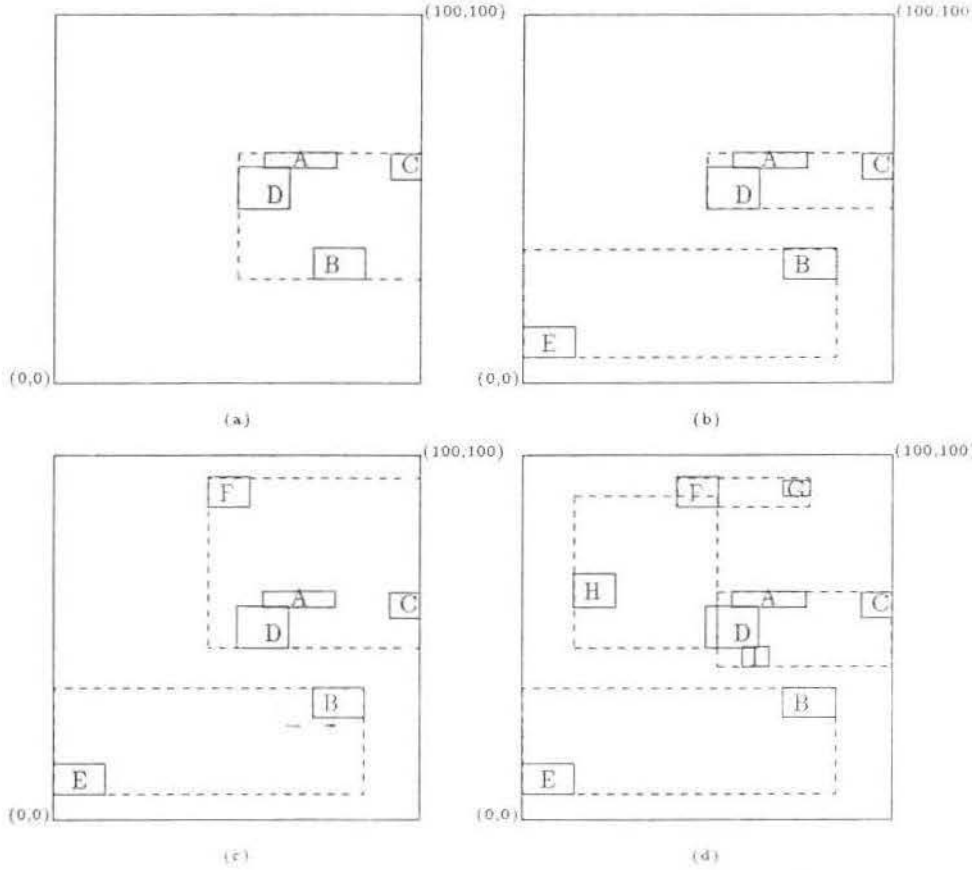


Figura 2.43: Demonstração de inserções em uma R^+-Tree através dos elementos: Cambuí (A), V. Georgina (B), J. das Palmeiras (C), Centro (D), V. Aeroporto (E), J. Sta. Genebra (F), J. Nilópolis (G), J. Eulina (H) e Ponte Preta (I).

remoção desse dado. Na figura 2.44 é ilustrada a discussão acima para eliminação do retângulo W na R$^+$-Tree (2.44a) e na R-Tree (2.44b).

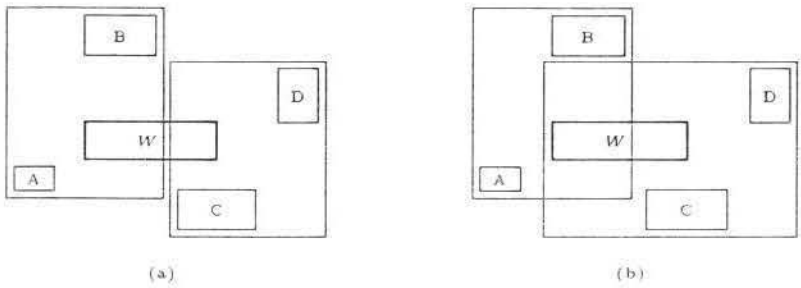


Figura 2.44: Ilustração da possibilidade de diversos percursos nos métodos R$^+$-Tree (a) e R-Tree (b), para se eliminar o objeto W.

Após a determinação das entradas referentes ao dado a ser removido, estas serão eliminadas de seus respectivos nós folhas. A operação de *underflow* só ocorrerá se, após a remoção destas entradas, algum nó folha se tornar vazio. Nesses casos, essa operação é propagada para o níveis superiores com o objetivo de se retirar a referência para essa entrada.

A remoção de qualquer entrada em um nó folha força o percurso inverso ao utilizado para chegar a esse nó, de sentido folha-raiz, a fim de avaliar e alterar, caso necessário, o MBR das entradas envolvidas nesse percurso.

Consulta

A R$^+$-Tree é apropriada para os mesmos tipos de consultas que a R-Tree. A diferença entre estas estruturas existe apenas na forma em que são tratadas as sobreposições dos nós intermediários de um mesmo nível, que na R$^+$-Tree não são permitidas. Essa característica da R$^+$-Tree torna o seu processo de consulta distinto da R-Tree, no sentido que qualquer subconjunto da área de pesquisa não estará representado pela área comum ao MBR das diversas entradas em um mesmo nó intermediário, não necessitando assim, diversos percursos para pesquisar uma mesma área.

O processo de consulta pode ser definido da seguinte forma. A partir da raiz, toda entrada cujo MBR sobrepor a área de pesquisa irá particionar essa área através da intersecção desse MBR para com essa área, ocasionando assim, um subconjunto da área de pesquisa. Todas as entradas que estiverem

nessa condição serão percorridas dessa mesma forma, até que se encontre nas suas respectivas folhas, algum objeto de dado que satisfaça o predicado da consulta. A figura 2.45a ilustra a partição da área de consulta Q em 3 subconjuntos denominados A, B, e C que, respectivamente, sobrepõem as entradas 1, 2, e 3 de um determinado nó intermediário. Na figura 2.45b é ilustrada a repetição desse processo para o nó filho da entrada 1, com a definição de um novo subconjunto do espaço de pesquisa.

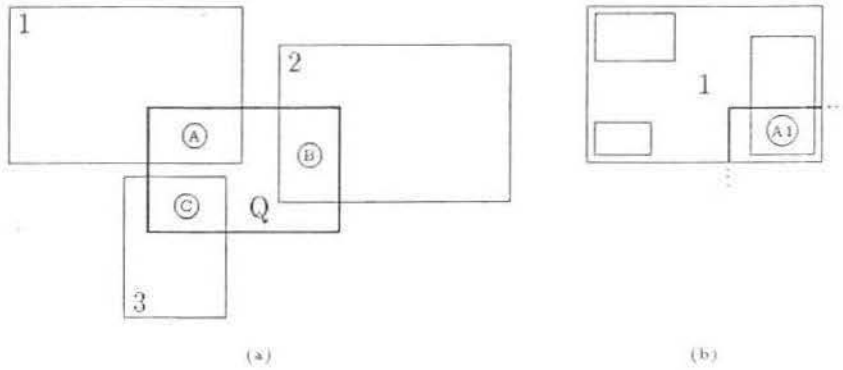


Figura 2.45: Ilustração de uma *window query* em uma R$^+$-Tree.

Considerações

É difícil determinar a profundidade máxima de uma R$^+$-Tree, pois dependendo da disposição dos dados, pode-se ter entradas referentes a um mesmo nó replicadas nas diversas folhas da estrutura. Na prática, a redundância das entradas em alguns nós não aumenta em grandes proporções a profundidade da estrutura, quando comparada a R-Tree, em virtude destas serem distribuídas logaritmicamente pela estrutura.

O algoritmo de inserção na R$^+$-Tree é mais caro que o da R-Tree. O procedimento é idêntico para ambas as estruturas, desde o percurso de sentido raiz-folha até o de sentido inverso para se atualizar os MBRs. Porém, enquanto que para R-Tree esse procedimento é utilizado uma única vez para cada elemento inserido, para R$^+$-Tree este procedimento pode ser repetido em razão do número de entradas a serem associadas a este dado. Acrescido a esse problema, tem-se o maior custo da operação de *split*, que pode ser propagado também em direção das folhas.

A remoção para a R$^+$-Tree também pode requerer diversos percursos caso existam diversas entradas associadas ao elemento a ser removido. Contudo, as operações de *underflow* são menos constantes, pois não existe um limite inferior para capacidade dos nós em uma R$^+$-Tree. Essa característica ocasiona uma pior utilização de espaço, comparado a R-Tree. Entretanto, estudos mostram que, em média, esta diferença na utilização de espaço permanece em torno de 10%. Para se melhorar essa utilização são sugeridos algoritmos para reorganizar a estrutura, alcançando assim, melhores resultados no desempenho da estrutura.

2.5.3 R*-Tree

Guttman [Gut84] identificou dois critérios determinantes para o bom desempenho da estrutura R-Tree, *coverage* e *overlap*. Segundo Guttman, estes dois critérios poderiam ser utilizados na redistribuição das entradas durante a operação de *split*, através da diminuição do *coverage* e da ausência de *overlap* dos MBRs dos nós resultantes. Contudo, como apresentado na figura 2.38, a tentativa de alcançar estes dois critérios simultaneamente nem sempre é possível. Guttman não propôs soluções para otimizar esses dois critérios em conjunto, adotando como critério único a diminuição do *coverage*.

Essa estratégia de Guttman fez emergir discussões sobre qual fator, dentre os apresentados acima, deveria ser otimizado. Além desses dois critérios, discussões surgiram em torno de dois novos: *margin* (soma do comprimento dos lados do MBR) e *storage* (referente a quantidade de espaço utilizado pela estrutura). A tentativa de combinar todos esses critérios, como estratégia de otimização da estrutura, resultou na utilização de numerosos experimentos, sobre um método padronizado, com uma variedade de dados e consultas. O resultado destes experimentos concluiu em uma estrutura variante da R-Tree, denominada R*-Tree, por N. Beckmann, H. P. Kriegel, R. Schneider e B. Seeger [BKSS90].

Esses dois métodos de acesso, R-Tree e R*-Tree, diferem basicamente pelos critérios utilizados para otimização da estrutura. Ao invés de apenas tentar diminuir ao máximo a dimensão do *coverage*, como na R-Tree, a R*-Tree também tenta eliminar o *overlap*, diminuir a *margin* e melhorar a utilização de espaço, ponderando cada um destes critérios na melhor combinação determinada em seus experimentos. Assim, a estrutura dos nós de uma R*-Tree permanece idêntica ao da R-Tree:

- nós folhas - composto por um conjunto de entradas do tipo

(*rect*, *tuple-id*)

onde:

- **tuple-id** é o identificador do objeto usado para referir-se a uma tupla no Banco de Dados.
- **rect** representa o retângulo de k dimensões cujos limites contém o objeto de dados referido por *tuple-id*.

- nós intermediários - composto por um conjunto de entradas do tipo

(*rect*, *child-pointer*)

onde:

- **child-pointer** contém o apontador para a subárvore hierarquicamente subordinada a esta entrada.
- **rect** representa o menor retângulo de k dimensões que contém todos os retângulos de todas as entradas do nó inferior a este hierarquicamente, ou seja, do nó para o qual essa entrada faz referência através de *child-pointer*.

Para construção da estrutura tem-se as seguintes restrições. Seja M o número máximo de entradas por nó e $2 \leq m \leq M/2$ o número mínimo:

1. cada nó folha possui entre m e M apontadores para registros, a menos que seja raiz.
2. para cada nó folha da forma (*rect*, *tuple-id*), *rect* é o menor retângulo que espacialmente contém o objeto de dados multidimensional representado por *tuple-id*.
3. cada nó intermediário possui entre m e M filhos, a menos que seja raiz.
4. para cada nó intermediário da forma (*rect*, *child-pointer*), *rect* é o menor retângulo que espacialmente contém os retângulos do nó filho.
5. o nó raiz possui pelo menos dois filhos, a menos que seja folha.
6. todas folhas aparecem no mesmo nível.

Na figura 2.46 é apresentado a R*-Tree correspondente aos dados da figura 2.9 com o valor de M e m iguais a 4 e 2 respectivamente.

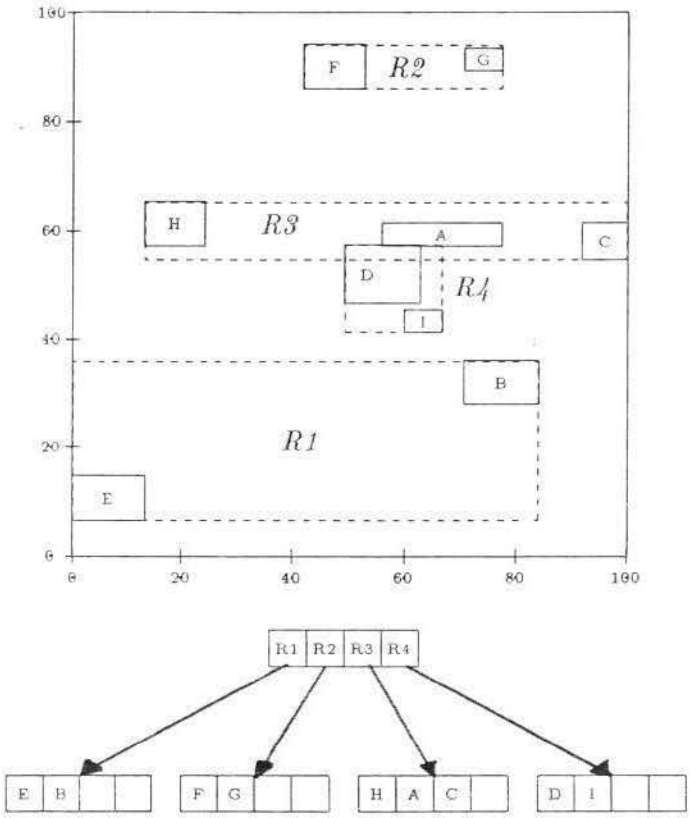


Figura 2.46: Uma R*-Tree e os retângulos que esta representa.

Inserção

O método de acesso R-Tree e suas variantes, R^+-Tree e R^*-Tree, são estruturas dinâmicas, não necessitando assim de reorganizações entre inserções e remoções intercaladas. Portanto, todas as estratégias para melhorar o desempenho na recuperação de elementos devem ser aplicadas durante a inserção de novos elementos de dados. Na rotina de inserção, essas estratégias são utilizadas em dois pontos importantes: no momento da decisão de qual subárvore conterá o novo elemento em suas folhas (percurso); e no momento de reorganizar as $M + 1$ entradas em dois nós, quando a inserção causa um *overflow* (*split*).

Todo novo elemento a ser inserido na R^*-Tree será armazenado nas folhas da estrutura. Diferente da R-Tree, onde este percurso de sentido raiz-folha é governado, exclusivamente, pela tentativa de diminuir ao máximo a área remanescente das entradas após inserido o novo elemento, na R^*-Tree o critério de sobreposição (*overlap*) também é avaliado para esse percurso. Em particular, será avaliado o *overlap* das entradas de um nó, como definido abaixo:

$$overlap\ (E_k) = \sum_{i=1, i \neq k}^{p} area(E_k.\text{rect} \cap E_i.\text{rect}),\ 1 \leq k \leq p$$

onde,

- $E_1, \ldots, E_p$ são entradas de um determinado nó
- *area* é uma função que calcula a área de um retângulo.

Assim, a partir da raiz, o processo é repetido em cada nó da seguinte forma. Se os filhos do nó corrente forem nós folhas, a entrada a ser utilizada no percurso será aquela cujo respectivo *overlap* aumentar menos com a inserção do novo dado. Igualdades serão resolvidas pela escolha da entrada cujo MBR necessitar menor aumento de área para incluir esse dado. Se os filhos do nó corrente não forem nós folhas, a escolha é semelhante a da R-Tree, a entrada a direcionar o percurso será aquela cujo respectivo MBR necessite de um menor aumento de área para armazenar o novo dado. Nesse último caso, igualdades são resolvidas pela escolha do MBR de menor área.

A operação de *split* na R^*-Tree, ocorrida quando a inserção de um elemento na folha da estrutura causa um *overflow*, também difere da operação na R-Tree. Para essa operação, todos os critérios de otimização citados anteriormente são avaliados com o objetivo de redistribuir as $M + 1$ entradas nos dois nós. A combinação desses critérios, determinada através dos experimentos, visam melhorar o desempenho das consultas.

As $M + 1$ entradas são avaliadas da seguinte maneira. Para cada eixo ou dimensão, as entradas são classificadas de acordo com seus valores inferiores, menor limite do retângulo naquela dimensão, e depois pelos seus valores superiores. A esse conjunto final são feitas $M - 2m + 2$ distribuições em dois grupos. O primeiro grupo conterá as primeiras $(m - 1) + k$ entradas, e o segundo grupo as restantes, onde $k = 1, ..., (M - 2m + 2)$. O eixo escolhido para divisão (*split axis*) será o que possuir a menor soma dos *margin-values* de suas distribuições.

Uma vez escolhido o eixo, a determinação da distribuição a ser utilizada será dependente do cálculo de *overlap*. A distribuição que possuir menor *overlap* será escolhida. Em caso de igualdade a que possuir menor área será a mais apropriada.

Os valores de *overlap*, *margin* e *area* para cada distribuição, são definidos da seguinte forma:

- $area\text{-}value = area[bb(primeirogrupo)] + area[bb(segundogrupo)]$
- $margin\text{-}value = margin[bb(primeirogrupo)] + margin[bb(segundogrupo)]$
- $overlap\text{-}value = area[bb(primeirogrupo)] \cap area[bb(segundogrupo)]$

onde,

— *bb* representa o MBR de um conjunto de retângulos.

— *margin* é uma função que calcula a soma do comprimento dos lados de um retângulo e

— *area* é uma função que calcula a área de um retângulo.

O critério de otimização de espaço está relacionado com o valor m (número mínimo de entradas por nó), e influenciará no número de distribuições. Os experimentos realizados apresentam como valor de m ideal, 40% do valor de M.

Ainda para inserção em R*-Tree, Beckmann [BKSS90] propôs uma estratégia semelhante à adotada para remoções em R-Tree. Guttman verificou que devido à inserção de elementos ser não determinística (i.e. diferentes seqüências de inserções constroem árvores diferentes), entradas referentes a elementos de dados inseridos anteriormente comprometem o desempenho da estrutura. Guttman utilizou-se deste critério para adotar a reinserção de entradas dos nós que sofreram *underflow*. Beckmann, porém, estendeu a estratégia de reinserção também para ser executada durante a inserção de um elemento qualquer. Em seus experimentos, comprovaram que a remoção

e depois inserção de dados antigos, inseridos durante a construção inicial da estrutura, melhoraram o desempenho das consultas de 20% a 50%. Assim, a estratégia sugerida compartilharia de inserção e *split*, quando houvesse *overflow*, da seguinte forma. Se a inserção de um elemento qualquer causa um *overflow* e este é o primeiro ocorrido naquele nível, as $M + 1$ entradas serão classificadas em ordem decrescente da distância entre o centro dos retângulos e o centro do MBR desse nó. P entradas iniciais serão removidas e submetidas a uma reinseção. Caso o *overflow* não seja o primeiro ocorrido nesse nível, durante a inserção de um único elemento, a operação de *split*, como descrita anteriormente, será utilizada.

Na figura 2.47 é ilustrado o processo de inserção para a construção da R*-Tree apresentada em 2.46, através da seqüência dos retângulos: Cambuí (A), V. Georgina (B), J. das Palmeiras (C), Centro (D), V. Aeroporto (E), J. Sta. Genebra (F), J. Nilópolis (G), J. Eulina (H) e Ponte Preta (I).

Remoção / Consulta

A R*-Tree surgiu através de experimentos realizados sobre a R-Tree com diferentes critérios de otimização. A diferença básica destas estruturas reside na inserção de elementos, pois devido às estruturas serem dinâmicas e, conseqüentemente, sem estratégias de reorganização, é o único ponto onde os critérios de otimização podem ser utilizados para melhorar o desempenho.

Portanto, pode-se afirmar que as metodologias para remoção e consulta de elementos em ambas as estruturas são idênticas. Quanto à remoção de elementos na R*-Tree, a única distinção existe na detecção de *underflow*, pois enquanto que na R-Tree esta operação é motivada quando um nó possui menos que $M/2$ entradas, na R*-Tree depende de um parâmetro da estrutura que designa o número mínimo de entradas por nó. Nos experimentos realizados por Beckmann [BKSS90], foram testados valores de 20%, 30%, 40% e 45% do número máximo de entradas por nó, apresentando como o parâmetro determinante do melhor desempenho, o valor de 40%.

Considerações

A R*-Tree surgiu a partir dos experimentos realizados na R-Tree com o objetivo de propor novos critérios de otimização ao único sugerido por Guttman [Gut84], diminuição dos MBRs dos nós envolvidos na operação de *split* (diminuição do *coverage*).

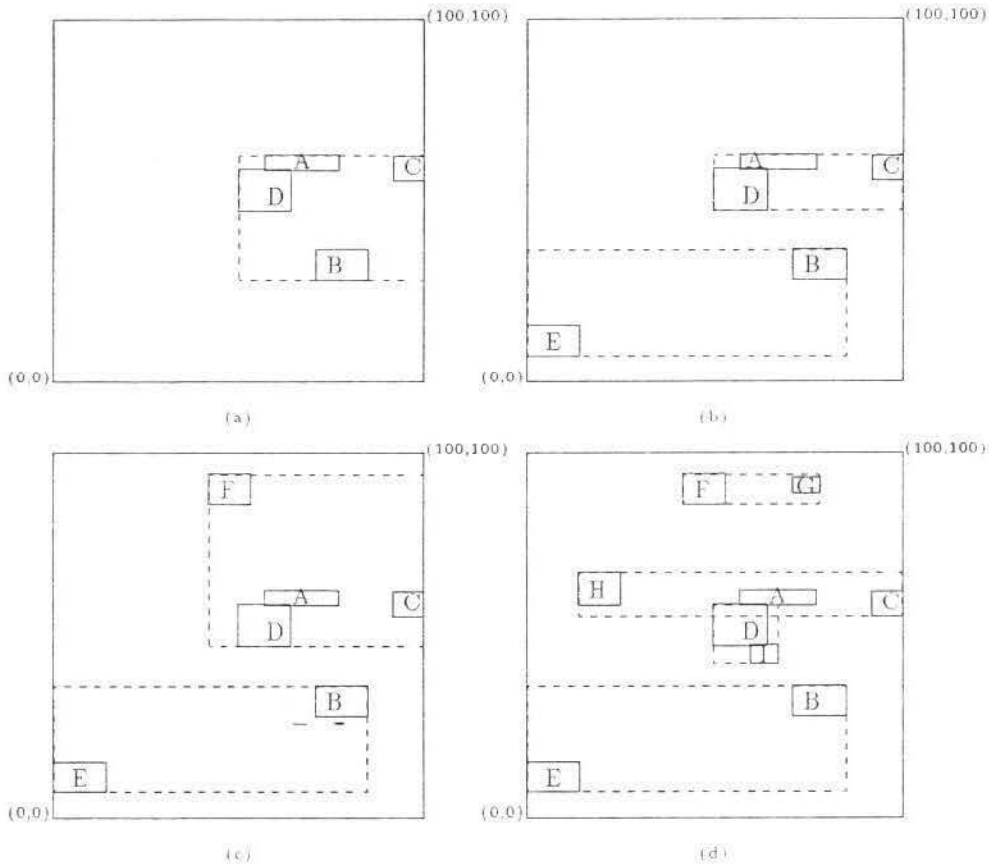


Figura 2.47: Demonstração de inserções em uma R*-Tree através dos elementos: Cambuí (A), V. Georgina (B), J. das Palmeiras (C), Centro (D), V. Aeroporto (E), J. Sta. Genebra (F), J. Nilópolis (G), J. Eulina (H) e Ponte Preta (I).

Como parte dos experimentos, Beckmann propôs uma comparação do desempenho da R*-Tree para com as estruturas R-Tree, proposta por Guttman [Gut84] e a variante proposta por Greene [Gre89]. Guttman apresentou três algoritmos para redistribuição das entradas, na operação de *split*, com distintas complexidade de tempo: exponencial, quadrático e linear. Beckmann utilizou em seus experimentos os dois últimos algoritmos que, por Guttman, concluiram em distribuições que tornaram o desempenho da estrutura semelhantes.

Beckmann utilizou o *benchmark* proposto por Kriegel [KSSS89], composto por seis arquivos de dados com diferentes distribuições e três tipos de consultas: *point queries*, *range queries* para determinação de intersecção e *range queries* para determinação de objetos de dados que estão contidas no retângulo de consulta.

Os resultados dos experimentos concluiram que há superioridade da R*-Tree sobre a R-Tree em todas as análises. Em particular, a R*-Tree apresentou maior ganho no desempenho para range queries de retângulos de consulta menores, o que segundo Bechmann, comprova a melhor preservação de ordem na R*-Tree (i.e., retângulos próximos são mais prováveis de serem armazenados na mesma página).

Como as estruturas R-Tree e suas variantes são também propostas para o gerenciamento de pontos no espaço, através da associação de um retângulo de extensão zero em todas as dimensões, Beckmann também realizou estudos comparativos sobre estas estruturas com o *benchmark* utilizado para métodos de acesso a pontos [KSSS89]. Nestes experimentos, o ganho da R*-Tree em relação às demais variantes foi superior ao ganho obtido nos experimentos com retângulos como objetos de dados. Inclusive, ao contrário do que se esperava, a R*-Tree apresentou melhores resultados do que os obtidos pelo método de acesso a pontos Grid File [NHS84].

Enfim, a R*-Tree mostrou-se a estrutura mais apropriada tanto para pontos como objetos de tamanho não-zero, do que as estruturas R-Tree e suas outras variantes testadas.

2.5.4 Cell Tree

A estrutura Cell Tree proposta por Gunther [Gun88, Gun89] é uma variação da B$^+$-Tree [Com79], que tem como objetivo gerenciar células no Banco de Dados de acordo com a localização destas no espaço.

Diferente dos demais métodos de acesso descrito nesse capítulo, a Cell Tree utiliza a estratégia *region decomposition* para representar seus obje-

tos de dados. Em particular, este método é apropriado para representação de objetos de dados de formas irregulares, através da representação destes em poliedros regulares mais simples. Esta metodologia, já comentada, compromete-se com uma maior precisão na representação do objeto de dados do que a estratégia MBR.

O método de acesso Cell Tree é composto de uma estrutura de índices dinâmica representada através de uma árvore de profundidade balanceada. Os dois tipos de nós existentes são organizados como descrito abaixo:

- nós folhas - composto por um conjunto de entradas do tipo

$$(id, Z)$$

onde:

  - **id** é o identificador do objeto usado para referir-se a uma tupla no Banco de Dados.
  - **Z** representa uma célula contida na soma dos poliedros p_i convexos que representam o objeto de dados referido por *id*.

- nós intermediários - composto por um conjunto de entradas do tipo

$$(child\text{-}pointer, P, C)$$

onde:

  - **child-pointer** contém o apontador para a subárvore hierarquicamente subordinada a esta entrada.
  - **P** é um poliedro convexo não necessariamente fechado. Todas as células no Banco de Dados que são subconjuntos de P estão na subárvore referenciada por *child-pointer*.
  - **C** é um subconjunto convexo de P que contém todas as células p armazenadas na subárvore referenciada por *child-pointer*. C provê uma localização mais precisa dessas células, melhorando assim a eficiência das consultas.

Seja m o parâmetro que especifica o número mínimo de entradas em um nó intermediário, a Cell Tree deve satisfazer as seguintes propriedades:

1. o nó raiz possui pelo menos duas subárvores a menos que seja um nó folha.

2. cada nó intermediário possui pelo menos m entradas a menos que seja a raiz da estrutura.

3. para cada entrada (*child-pointer*, P, C) de um nó intermediário, a subárvore referenciada por *child-pointer* contém uma célula p somente se $p \subseteq P$.

4. para cada entrada (*child-pointer*, P, C) de um nó intermediário, o campo $C \subseteq P$ é um poliedro convexo que pode ser especificado como a intersecção de P com no máximo k *halfspaces* no espaço E^d. Para cada célula p armazenada na subárvore referenciada por *child-pointer*, $p \subseteq C$.

5. para cada nó intermediário N, os poliedros Ps das entradas de N formam um particionamento binário do espaço, semelhante à *binary space partitioning* tree, ocupado pelo poliedro P da entrada pai do nó N.

6. todas as folhas estão no mesmo nível.

7. quase todos os nós requerem não mais do que uma página de disco de espaço de armazenamento.

Pela propriedade 5 é definido que o campo P de duas entradas em um mesmo nível não podem se sobrepor. Outra característica que pode ser identificada é a ausência de um limite máximo para o número de entradas por nó. O limite utilizado é a capacidade de armazenamento de uma página de disco. Contudo, após sucessivas inserções, um nó pode exceder este limite ocasionando uma operação de *overflow*. Diferente dos demais métodos apresentados, nem sempre o *split* pode ser executado, fazendo com que esses nós necessitem de páginas de *overflow*.

Inserção

Para inserção de um objeto de dados, este objeto deve primeiro ser decomposto em um conjunto de poliedros convexos de forma que a soma destes poliedros formem o objeto original.

Seja D um objeto de dados representado pelo conjunto de poliedros convexos D_i, tais que $\sum D_i = D$. Cada *cell* D_i é inserida na estrutura recursivamente, a partir da raiz, da seguinte forma. Se o nó corrente é um nó

intermediário, então cada entrada cujo campo P sobrepõe D_i será percorrida através de sua subárvore indicada por *child-pointer*. O campo C dessa entrada é estendido com o objetivo de conter a intersecção de D_i e P. Se o nó corrente é um nó folha, D_i é armazenado em uma entrada desse nó. Caso o nó não possa armazená-lo, este sofre uma operação de *split* [Gun89].

Verifica-se no processo de inserção, que cada *cell* D_i do objeto de dados original D pode causar a criação de mais do que uma entrada em nós folhas distintos.

Remoção

A remoção de um objeto de dados D é feita através da remoção de todas as entradas associadas a este objeto de dados armazenadas nos nós folhas da estrutura.

Assim, após a decomposição do objeto de dados D nos poliedros convexos regulares D_is, cada *cell* D_i é submetida a uma rotina de remoção recursiva, a partir da raiz, da seguinte forma. Se o nó corrente for um nó intermediário, cada entrada cujo respectivo campo P sobrepõe D_i será percorrida através da sua subárvore referenciada por *child-pointer*. Se o nó corrente for um nó folha, a entrada referente ao objeto de dados D, se existente, será removida. Nesse caso, o campo C da entrada pai desse nó, que representa o espaço utilizado por todas as suas entradas, será alterado caso a remoção do objeto de dados D cause uma contração desse espaço.

Se a remoção das entradas nos nós folhas tornarem alguns destes nós vazios, estes são eliminados e sua eliminação propagada aos nós superiores da estrutura. Nós intermediários com menos que m entradas são removidos e suas entradas reinseridas.

Consulta

As características de representação dos dados da Cell Tree permite *range queries* sobre espaços de consulta de forma arbitrária.

Gunther [Gun88, Gun89] apresentam *range queries* para determinação de objetos de dados que sobrepõem o espaço de consulta, embora a variação para outros tipos de predicados seja facilmente compreendida. O primeiro passo do algoritmo executa a decomposição do espaço de consulta S em um conjunto de poliedros regulares S_is, tais que $\sum S_i = S$. A partir da raiz da estrutura, o algoritmo descende recursivamente até as folhas de forma semelhante ao algoritmo da R-Tree. Cada entrada em um nó intermediário

cujo campo C sobrepõe uma *cell* S_i qualquer será percorrida pela subárvore referenciada pelo campo *child-pointer* dessa entrada. Ao se chegar a um nó folha, cada entrada é submetida ao predicado de consulta com o objetivo de confirmar ou não a sobreposição do objeto de dados representado pelo campo Z dessa entrada com a *cell* S_i.

Considerações

O método de acesso Cell Tree apresenta como grande inovação em relação aos demais métodos de acesso derivados de *multiway trees*, a representação de objetos de dados através da estratégia *region decomposition*.

Essa representação permite o suporte de objetos espaciais mais complexos de forma mais precisa do que a estratégia MBR. Em particular, conforme Gunther [Gun88, Gun89], a Cell Tree é apropriada para representação de objetos de formas arbitrárias, especialmente objetos de dados que não são formados pelos seus próprios intervalos.

Contudo, essa estratégia de representação introduz certa complexidade. A inserção de objetos de dados no Banco de Dados pode causar a criação de hiperplanos caso qualquer face de algumas das *cells* que representam esse objeto de dados não faça adjacência a nenhum dos hiperplanos já existentes. Embora a criação de hiperplanos possa ser manipulada sem perda da eficiência, a remoção de objetos de dados podem tornar hiperplanos redundantes. Estes hiperplanos devem ser removidos, porém, o custo de tal processo impede a atualização dos hiperplanos a cada remoção de um objeto de dados.

Além da complexidade de tempo existente na estratégia para decomposição do objeto de dados, conforme Gunther [Gun88], a determinação do predicado de consulta sobre as *cells* requer um custo maior do que a determinação sobre os MBRs.

Capítulo 3

Implementação

Na seção anterior foram apresentados alguns dos métodos de acesso a dados espaciais mais citados na literatura. Buscou-se determinar, por meio dos estudos destes métodos, os mais apropriados a serem utilizados pelos SGBDs. Todavia, esta determinação não pode ser restrita ao estudo analítico de suas estruturas de dados, com a identificação das melhores relações de desempenho, flexibilidade e simplicidade. Além de alguns métodos de acesso possuírem características que tornam difícil a sua representação em um modelo matemático, como por exemplo os diversos parâmetros existentes na rotina de *split*, certas peculiaridades existentes em implementações de SGBDs podem interferir no desempenho de cada um desses métodos de formas diferentes, como utilização de *buffer pool*.

3.1 Definição do Critério de Análise

A análise desses métodos, seja analítica ou não-analítica, envolve duas complexidades: complexidade de tempo e complexidade de espaço. Estas estão associadas à quantidade de tempo e a de espaço utilizadas pelo método de acesso durante a execução de cada uma de suas operações básicas inserção, remoção e consulta.

Nos experimentos atuais deu-se ênfase à complexidade de tempo, porque foi identificado que um dos problemas mais críticos no projeto de SGBDs é o de prover sistemas que atinjam um bom nível de satisfação dos usuários. Em outras palavras, sistemas que possam atender as diversas requisições de operações do usuário de forma mais rápida possível.

O critério escolhido para análise da complexidade de tempo foi a quanti-

dade de acessos a disco. Esse critério torna-se o mais importante, reunindo inúmeros esforços dos métodos de acesso, na tentativa de minimizá-lo, em virtude das informações para recuperação estarem armazenadas no disco e da quantidade de tempo demandado na execução dessas operações possuir grandeza maior comparada as demais instruções dos algoritmos.

O tempo requerido pelos algoritmos, independentemente do gerado pelas instruções de leitura e gravação a disco, não foi analisado. A complexidade de espaço, por sua vez, foi analisada com menor importância, considerando-se o alto grau de desenvolvimento do hardware através de memórias cada vez maiores e mais baratas.

3.2 Métodos Escolhidos

Optou-se, neste estudo, pela análise da implementação da classe de métodos derivados de *multiway trees*. Em particular, foram implementados os métodos R-Tree e suas variantes. Os métodos K-D-B Tree e Cell Tree, também apresentados como derivados de *multiway trees*, não foram implementados. O método K-D-B Tree, além de não suportar os diversos tipos de dados espaciais, conforme os experimentos de Greene [Gre89], apresentou desempenho inferior em relação aos métodos variantes de R-Trees. O método Cell Tree, apesar de mais robusto, permitindo uma representação mais eficiente de tipos de dados espaciais mais complexos, embute essa complexidade no funcionamento das operações básicas da estrutura (inserção, remoção e consulta) e afeta o seu desempenho. A grande vantagem da estrutura R-Tree e suas variantes é a simplicidade com que representa os objetos de dados e, conseqüentemente, dos algoritmos de consulta mais simples para determinação dos relacionamentos espaciais entre esses objetos.

Os métodos baseados em árvores binárias, apesar de terem sido os primeiros a propor gerenciamento de dados espaciais, apresentam características que tornam discutível a sua utilização como métodos de acesso a dados espaciais. Aqui identificou-se como problemas cruciais dessa classe de métodos: o fato de possuírem um *fan-out* muito pequeno em relação à capacidade de armazenamento das páginas de disco e o fato das estruturas de dados utilizadas por esses métodos não serem balanceadas.

Diversas estratégias têm sido propostas para atender a esses problemas, com o plano de tornar essa classe de métodos de acesso mais apropriada ao gerenciamento de dados em disco. Assim, para o primeiro problema surgem estratégias, como a adotada por Ooi [OMSD87] na implementação da Spatial

K-d Tree (representação das subárvores da estrutura em páginas de disco), denominada por Knuth [Knu73b] de *multiway trees*, buscando-se aproximar a estruturação dos métodos de acesso à estruturação das árvores de múltiplos caminhos. Essa estratégia ocasiona melhor aproveitamento das páginas de disco, possibilitando a diminuição das *page faults* durante o percurso na estrutura.

Quanto ao fato das estruturas de dados não serem balanceadas, as soluções para contorno dos problemas têm sido menos interessantes. Já que as estratégias para manter a estrutura balanceada durante todo o tempo são obtidas através de algoritmos muito caros, as soluções propostas visam prover o balanceamento da estrutura através de duas alternativas: aprimoramentos no algoritmo de construção da estrutura, quando os dados são conhecidos a priori; rotinas de balanceamento estáticas a serem executadas com determinada periodicidade.

Embora a primeira alternativa justifique a utilização desses métodos em ambiente estático, não os tornam flexíveis, pois ambientes com grande atualização de dados não são suportados eficientemente. A segunda alternativa torna-se mais crítica em um ambiente de multiprocessamento, por causa da interrupção de todas as aplicações que utilizam esses métodos a cada execução da rotina de rebalanceamento.

Fazendo uma análise da aplicação ou não desses métodos, o que se encontra é uma controvérsia. Alguns autores descartam totalmente a sua utilização [SRF87, KSSS89, FSRM], outros desenvolvem pesquisas sobre sua utilidade [HJ]. Nesta análise o autor da dissertação concorda com os fatos dos métodos serem impróprios, mas com uma ressalva sobre os métodos Quadtree e suas variantes, em particular às características de suas estruturas de dados. Não obstante apresentarem um desempenho questionável como métodos de acesso, são adequadas à representação de dados espaciais, podendo, inclusive, preservar suas características (localização e extensão no espaço) e abstraí-las através da representação dos dados como conjunto de pontos, nesse último caso apropriado para representação de imagens.

A outra classe de métodos de acesso, derivados de estruturas *hash*, apresenta por definição um bom desempenho para *point queries*, como através do princípio de dois acessos a disco no Grid File, e um desempenho esperado como satisfatório para *range queries*. O ponto desfavorável dessa classe é que as operações de inserção e remoção, quando ocasionam *split* e *merge*, respectivamente, geram inúmeros acessos a disco em conseqüência da necessidade do *Grid Directory* em se expandir e contrair como respostas às operações. Ademais, essa classe de métodos de acesso provê suporte a tipos restritos

de dados, i.e. pontos no espaço. Ainda que Hinrichs [HN83] sugira uma maior flexibilidade desses métodos através da transformação de retângulos d-dimensionais em pontos $2d$-dimensionais, a maior complexidade das rotinas de consulta e o aumento no custo das operações de inserção e remoção, devido a maior dimensão do *grid directory*, tornam a estratégia imprópria.

Mesmo sendo um método pouco flexível, os métodos derivados de estruturas *hash* tornam-se mais apropriados ao uso em Banco de Dados do que os métodos derivados de árvores binárias, em virtude do comprometimento com um melhor desempenho. Nesses casos, para que o SGBD possa prover suporte à diversidade de tipos de dados, outros métodos devem ser implementados.

As características dos métodos derivados de estruturas *hash*, em particular o Grid File, apresentado no capítulo anterior, tornam interessante a análise de sua implementação. Entretanto, um estudo realizado por Beckmann [BKSS90], sobre o desempenho desse método em relação a outro derivado de *multiway trees*, resultou na superioridade desse último. As características dos métodos derivados de *multiway trees* e a flexibilidade do método testado no suporte aos diversos tipos de dados espaciais (pontos e regiões no espaço) fizeram, neste estudo, descartar a implementação do Grid File.

Analogamente às aplicações convencionais, onde os métodos derivados de *multiway trees* apresentam um melhor desempenho e um maior suporte aos diversos tipos de consultas do que árvores binárias e estruturas *hash*, nas aplicações espaciais essa classe de métodos também mostra-se mais favorável. Todavia, diferente das aplicações convencionais onde árvores tipo B e B^+ tornaram-se padrões, existindo uma definição precisa sobre quais condições de dados e consulta cada um desses métodos é mais apropriado, nas aplicações espaciais não existe um consenso sobre os métodos R-Tree e suas variantes [SRF87, Gre89].

3.3 Descrição

Para facilitar a avaliação dos métodos implementados através do critério de análise em questão nesse trabalho, quantidade de acesso a disco, os métodos de acesso foram implementados em dois níveis de abstração: módulo de abstração do disco e módulo de métodos de acesso.

O módulo de abstração do disco é responsável pela abstração lógica do sistema de arquivos fornecido pelo sistema operacional em páginas lógicas

de disco.

O módulo de métodos de acesso corresponde à implementação dos métodos de acesso.

A seguir será descrita a implementação desses módulos que foram elaborados em linguagem C, executando sob máquinas SUN (Sparc Station) com sistema operacional SUN OS 4.01 (derivado do UNIX BSD).

3.3.1 Módulo de Abstração do Disco

O módulo de abstração do disco faz a abstração das informações em um determinado arquivo, em páginas lógicas de disco. Esse módulo foi criado para permitir o controle do pesquisador sobre as requisições de leitura e gravação dos métodos de acesso, porquanto todas essas operações serão feitas por funções fornecidas pelo módulo. O propósito desse controle é oferecer as condições necessárias à avaliação dos métodos de acesso através do critério de análise (quantidade de acesso a disco).

Dessa forma, verifica-se como função desse módulo, além da comunicação entre os métodos de acesso e as informações armazenadas no disco, a contabilização da quantidade de acessos a disco solicitadas pelo método de acesso.

Como a implementação do presente material visa aproximar-se das reais condições em que os métodos de acesso são implementados nos SGBDs, esse módulo incorpora ainda um gerenciador de *buffer pool*. O *buffer pool* contém na memória principal as m (parâmetro desse módulo) páginas mais recentemente utilizadas, a fim de evitar que a cada requisição de leitura ou escrita dessas páginas sejam geradas operações de leitura ou escrita no disco.

Quanto à quantidade de páginas a ser utilizada pelo *buffer pool*, este estudo propõe um valor entre 10% e 20% da média da quantidade de páginas utilizadas pelos diversos métodos e arquivos; como política a ser utilizada para troca de páginas, sempre que uma nova for requisitada, propõe-se a política LRU (*least recently used*) onde a página menos recentemente utilizada dará lugar à mais nova.

Associada à utilização do *buffer pool*, foi gerada uma cadeia de referência contendo a seqüência das requisições das páginas de disco através dos seus números e operações, que a requisitaram (leitura ou escrita). Esta seqüência obtida para cada método de acesso possibilitará, através de uma análise, a determinação do tamanho do *buffer pool* e a política para substituição das páginas mais adequadas a cada método.

A comunicação entre o método de acesso e as informações armazenadas no disco é obtida através de seis funções básicas:

- *open_blockfile* - abre/cria um arquivo e inicializa as variáveis necessárias ao gerenciamento das páginas de disco.
- *alloc_block* - retorna o endereço de uma página disponível a ser utilizada.
- *read_block* - lê uma página específica no *buffer pool*/disco.
- *write_block* - escreve em uma página específica armazenada no *buffer pool*.
- *dispose_block* - coloca em disponibilidade a área de armazenamento representada por uma página. Nesse caso essa página é colocada na lista de espaços disponíveis para ser posteriormente requisitada através de uma função *alloc_block*.
- *close_blockfile* - fecha o arquivo.

A contabilização da quantidade de acessos a disco solicitada pelo método de acesso é obtida através das funções *read_block* e *write_block*. Nessas funções, a requisição de leitura e escrita das páginas de disco é diferenciada em leitura e escrita no *buffer pool* e no disco.

3.3.2 Módulo de Métodos de Acesso

Esse módulo refere-se à implementação de cada método de acesso, sendo ela dividida em submódulos de acordo com a meta de cada um desses:

- submódulo de inicialização - inicializa o método de acesso através da abertura e fechamento do arquivo de índices.
- submódulo de inserção - provê os algoritmos para inserção dos dados e de tratamento de *split*.
- submódulo de remoção - provê os algoritmos para remoção dos dados e de tratamento de *underflow*.
- submódulo de consulta - provê os algoritmos de consulta.
- submódulo de biblioteca - contém funções básicas comuns aos demais submódulos.

Os algoritmos referentes às funções básicas (inserção, remoção e consulta) foram implementados através de recursão. Embora alguns autores não utilizem essa estratégia, devido ao custo das sucessivas chamadas recursivas às funções [Gre89], optou-se aqui pela sua utilização em troca da maior simplicidade.

Associado ao conjunto de variáveis necessárias para o gerenciamento de cada método de acesso, foi alocado espaço de armazenamento referente a três páginas de disco. Essas páginas têm a função de prover maior simplicidade e eficiência no gerenciamento dos métodos de acesso de acordo com suas atribuições. Uma página conterá a raiz da estrutura de índices, pois esta será requisitada constantemente a cada execução de uma operação básica. Uma outra página conterá o nó corrente da estrutura de índices, enquanto uma terceira será utilizada durante a operação de *split* de cada nó.

Ainda que a implementação dos métodos de acesso tenha tido como base os artigos onde os mesmos foram apresentados, por meio dos algoritmos sugeridos foram encontradas algumas condições especiais, não discutidas nesses artigos, que impulsionaram a tomada de decisões particulares. A seguir serão apresentadas essas particularidades existentes na implementação da presente pesquisa restrita a cada método de acesso: R-Tree, como proposto originalmente por Guttman [Gut84]; R-Tree proposto por Greene [Gre89] através da estratégia diferente adotada à rotina de *split*; R^+-Tree como proposto por Sellis [SRF87, Sel]; e R^*-Tree proposto por Beckmann [BKSS90].

R-Tree (Guttman)

A R-tree foi implementada de acordo com os algoritmos sugeridos por Guttman [Gut84]. Entretanto, a rotina de inserção, como relatada por Guttman, não é capaz de atender à reinserção de entradas ocorrida como conseqüência do tratamento de *underflow*. Quando a remoção de um elemento em um nó intermediário deixa-o com menos que m entradas, sendo m o número mínimo de entradas por nó, esse nó é removido e suas entradas reinseridas na estrutura, no mesmo nível do nó removido. Assim, com a finalidade de utilizar o código da rotina de inserção, já que a rotina de Guttman só é capaz de atender a reinserção de entradas nos nós folhas, foi utilizada uma variável a mais como parâmetro que indica o nível na estrutura em que o elemento deve ser inserido. Dessa forma o código da rotina de inserção foi modificado para interromper as sucessivas chamadas recursivas sempre que o nível passado como parâmetro é atingido. Para a inserção de novos elementos de dados, o nível passado como parâmetro é o da profundidade da estrutura.

Uma outra peculiaridade da implementação deste estudo foi a escolha da rotina de *split*. Guttman utilizou como critério único para otimização da R-Tree a diminuição do *coverage*. Então, a estratégia de *split* teve como objetivo a redistribuição das $M + 1$ entradas nos dois nós, de forma que a área do MBR desses fosse a menor possível. Três algoritmos com complexidade de tempo distintos foram propostos por Guttman à rotina de *split*: exponencial, quadrático e linear. Mesmo identificando a melhor relação de custo/benefício aos dois últimos, este estudo optou pelo algoritmo quadrático que, conforme os experimentos de Beckmann [BKSS90], contribui em um desempenho melhor para as demais operações básicas do que o algoritmo linear.

Apesar de Guttman ter definido o número mínimo de entradas permitido por nó a ser igual a metade do número máximo, aqui foi empregado o valor sugerido por Bechmann [BKSS90], que realizou diversos experimentos para identificar o melhor valor a esse parâmetro, concluindo na quantidade de 40% do número máximo de entradas.

A tabela 3.1 apresenta a descrição da implementação da R-Tree através da quantidade de linhas de código em programa fonte.

Submódulos	Tamanho do Código
Inicialização	111
Inserção	340
Remoção	331
Consulta	194
Biblioteca	265

Figura 3.1: Descrição do Método de acesso R-Tree em linhas de código fonte.

R-Tree (Greene)

Greene [Gre89] apresentou um estudo sobre a implementação e desempenho de alguns métodos de acesso. Em particular, a R-Tree, como proposta por Greene, diverge da proposta de implementação de Guttman [Gut84]. A diferença básica dessas implementações reside na política adotada para *split*. Neste estudo houve opção pela implementação desse método de acesso, denominado aqui R-Tree Greene, objetivando enriquecer o presente estudo sobre R-Tree e suas variantes.

A rotina de *split*, como sugerido por Greene, é simples. Após a identificação dos retângulos mais distantes dentre os $M + 1$ (sendo M o número máximo de entradas por nó) a serem redistribuídos, é feita a escolha do eixo de partição. O eixo escolhido será o que possuir a maior separação entre esses dois retângulos. A separação será determinada pela distância desses retângulos dividida pelo comprimento do MBR dos $M + 1$ elementos nesse eixo (valor normalizado). Posterior à determinação do eixo, os retângulos são classificados pelo seu valor inferior nessa dimensão. Os primeiros $(M + 1)/2$ elementos serão alocados a um nó, os $(M + 1)/2$ últimos elementos serão alocados ao outro nó e o $((M + 1)/2 + 1)$-ésimo elemento, caso $M + 1$ seja ímpar, será alocado ao nó que necessitar menor aumento de seu MBR para representá-lo.

Tanto Greene [Gre89] como Beckmann [BKSS90], que também analisa a implementação da R-Tree como sugerida por Greene, não especificam se a distância a ser calculada entre os retângulos, durante a determinação do eixo de partição, é interna, como ilustrado na figura 3.2a, ou externa, como na figura 3.2b.

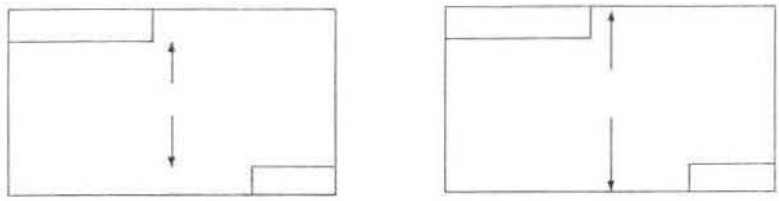


Figura 3.2: Ilustração do cálculo da distância entre os retângulos: interna (a) e externa (b).

Ao ser verificado, em alguns destes experimentos iniciais, que o cálculo a partir da distância externa possibilitou uma maior freqüência de valores normalizados iguais a 1, conseqüentemente algumas igualdades, cujo procedimento para tratamento não foi elucidado por Greene ou Beckmann, foram encontradas diante da determinação desses valores entre os demais eixos (figura 3.3).

Há um entendimento que o cálculo a partir da distância externa propicia a sobreposição dos MBR dos nós resultantes, pois a extensão dos

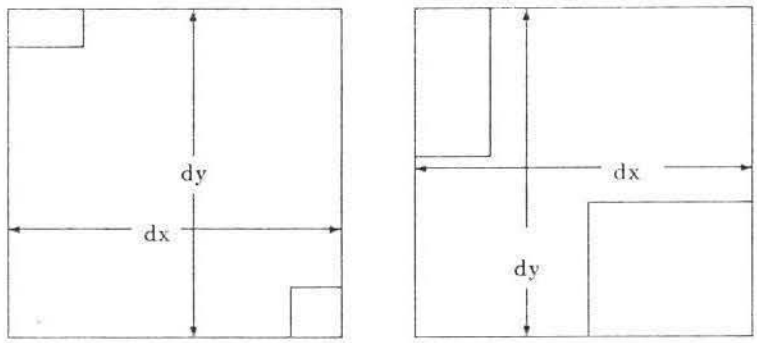


Figura 3.3: Nesses exemplos, o valor da separação normalizada tanto no eixo x como y é igual a 1.

retângulos utilizados para o cálculo não são levados em consideração (3.3a *versus* 3.3b). Assim, na presente implementação adotou-se pelo cálculo a partir da distância interna dos retângulos.

A tabela 3.4 apresenta a descrição da implementação da R-Tree Greene através da quantidade de linhas de código em programa fonte.

Submódulos	Tamanho do Código
Inicialização	113
Inserção	322
Remoção	331
Consulta	194
Biblioteca	265

Figura 3.4: Descrição do Método de acesso R-Tree em linhas de código fonte.

R^+-Tree

A presente implementação da R^+-Tree diverge significativamente da proposta por Sellis, Faloutsos e Rossoupoulos [SRF87]. A implementação como

relatada em [SRF87] é incompleta. Em particular, a rotina de inserção não apresenta o procedimento para as condições em que o MBR do elemento a ser inserido intercepta parcialmente a algumas entradas (figura 3.5a) ou não intercepta nenhuma (figura 3.5b).

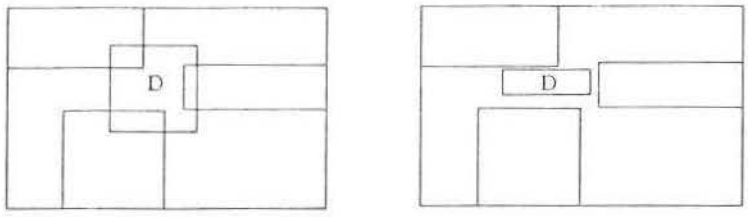


Figura 3.5: Esse exemplo ilustra condições especiais de inserção de elementos que não são suportadas no artigo original de Sellis.

Mesmo que essas questões tenham sido esclarecidas posteriormente por Sellis e Faloutsos [Sel, FSRM], aqui foi escolhida a estratégia de implementação sugerida por Sellis [Sel], a partir do estudo sobre diversas alternativas de implementação realizado por Epstein e Hartman [EH88].

A estratégia de Sellis [SRF87], revisada por Faloutsos [FSRM], requer algoritmos complexos na rotina de inserção, para atender as condições especiais citadas acima (Greene [Gre89], em seus experimentos concluiu, inclusive que a vantagem da R^+-Tree em relação à R-Tree, para determinadas condições, não justifica a sua implementação em virtude dessa complexidade). A estratégia sugerida por Epstein [EH88], por sua vez, é caracterizada pela simplicidade, tornando a implementação da R^+-Tree facilmente obtida a partir de extensões da implementação de uma R-Tree.

A idéia básica da estratégia de implementação sugerida por Epstein [EH88] foi associar a cada entrada de um nó em uma R^+-Tree um outro retângulo k-dimensional, denominado MaxRect, sendo k a dimensão do espaço onde os objetos estão dispostos. Esse novo retângulo, diferente do já existente, descreve o espaço no qual cada entrada é responsável. Esse campo MaxRect possui as seguintes propriedades:

- todos os MaxRects de um determinado nó intermediário dividem completamente e sem sobreposição o espaço representado pelo MaxRect da entrada pai desse nó.

- caso o nó intermediário seja raiz, a união dos MaxRects representam o espaço total onde os dados são dispostos.

Na prática, essas propriedades tornam a inserção de elementos mais simples. Como esse novo retângulo será utilizado para direcionar o percurso na estrutura durante a inserção de elementos, as condições especiais levantadas anteriormente jamais existirão. A partir da raiz, durante o percurso o MBR do elemento a ser inserido será sempre interceptado totalmente pelos campos MaxRect que compõem as entradas do nó corrente. Para cada entrada que satisfaça a essa condição, a sua respectiva subárvore será percorrida de forma semelhante, até que se chegue às folhas da estrutura. A figura 3.6 ilustra as propriedades acima relacionadas com o campo MaxRect. Em (a) tem-se a representação do nó raiz apenas pelos MaxRects de suas entradas correspondentes aos dados da figura 2.41, enquanto que em (b) tem-se a representação dos nós filhos da raiz.

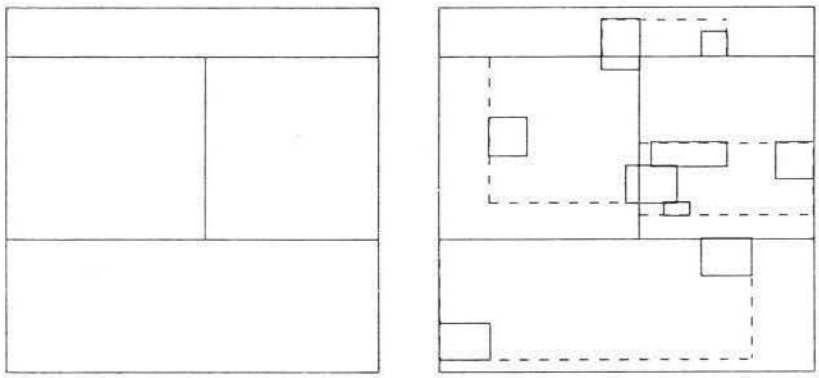


Figura 3.6: Ilustração das propriedades do campo MaxRect.

Ainda que seguindo como base da presente implementação o artigo [EH88], algumas particularidades de certos algoritmos não foram apresentadas, obrigando o autor do assunto em questão a tomar decisões próprias.

Na rotina de *split*, dentre as diversas estratégias sugeridas para redistribuição das entradas em dois nós, houve opção pela utilizada nos experimentos de Epstein [EH88], que realoca as entradas tendo como critério de otimização a diminuição do *coverage* dos nós resultantes. Entretanto foram

encontrados casos especiais em que essa rotina de *split* não conseguia fazer a partição. Suponha na figura 3.7a, o MBR de quatro elementos (A, B, C e D) que devem ser redistribuídos em dois nós com capacidade máxima de três entradas e com fator de redistribuição mínimo para nó igual a dois. As figuras (b), (c), (d) e (e) ilustram a varredura que determina a melhor partição para o eixo x, no sentido da esquerda para direita e da direita para esquerda, e para o eixo y, de cima para baixo e de baixo para cima, respectivamente.

Pela definição do procedimento de *split*, essa varredura é interrompida tão logo o número de retângulos se iguala ao fator de redistribuição mínima.

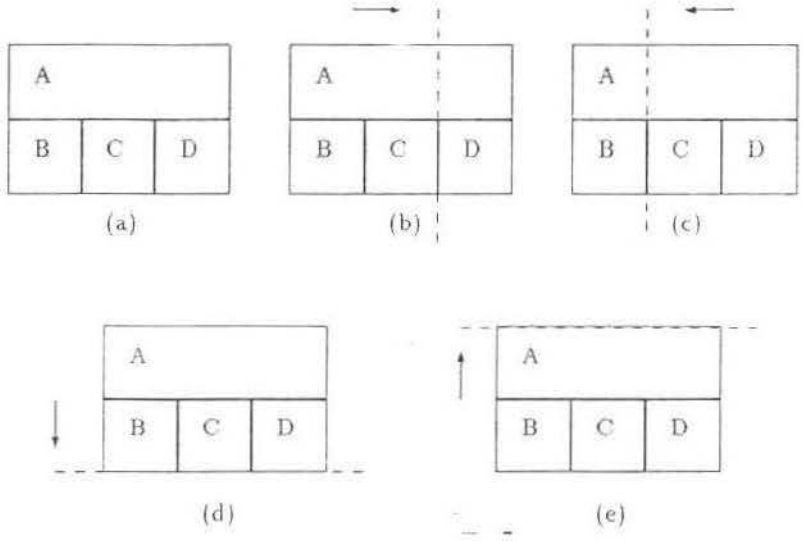


Figura 3.7: Ilustração do procedimento de *split*.

A impossibilidade da determinação da linha de partição é justificada, nesse caso específico, pelo fato dos retângulos serem adjacentes e, conseqüentemente, da linha de partição interceptar a área representada pela adjacência desses retângulos. Daí as figuras (b), (c), (d) e (e) geram respectivamente nós com quatro (A, B, C e D) e três (A, C, D) retângulos, três (A, B e C) e quatro (A, B, C e D) retângulos, quatro (A, B, C e D) e três (B, C e D) retângulos e quatro (A, B, C e D) e quatro (A, B, C e D) retângulos.

A solução aqui adotada para contornar essa situação é empírica, variando o fator de redistribuição (de 1 ao número máximo de entradas por nó - 1) até

encontrar uma partição que seja possível. No caso extremo da estratégia não funcionar, foi analisada a possibilidade de partição sobre uma linha gerada aleatoriamente.

Em teoria, identificou-se um caso extremo onde nenhuma das soluções alternativas consegue proceder uma operação de *split*. Ainda com os parâmetros do exemplo anterior, essa condição pode ser obtida com a inserção de quatro elementos iguais ou de tamanhos e localizações muito semelhantes. Na prática, contudo, espera-se que a maior capacidade no número de entradas máxima permitida por nó ocasionem a pouca probabilidade de que condições extremas como essa possam ocorrer. Todavia, essa possibilidade existe e deve ser identificada pelo algoritmo de *split*.

Na rotina de remoção a atual pesquisa identificou algumas condições que degradam o desempenho da estrutura e que são conseqüências da estratégia adotada para implementação. Pela estratégia de Sellis e Faloutsos [SRF87, FSRM], a remoção de uma entrada em um nó, tornando-o vazio, obriga a remoção da respectiva entrada pai desse nó. Pela estratégia de implementação empregada nem sempre a entrada pai desse nó pode ser removida.

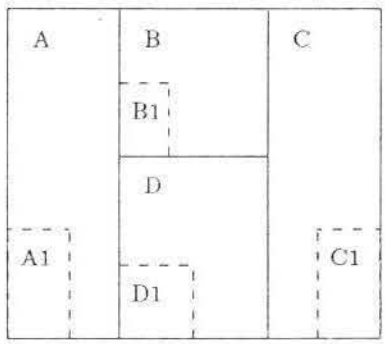


Figura 3.8: Ilustração do procedimento de remoção.

Na figura 3.8 ilustrou-se a representação de um nó através dos MaxRects A, B, C e D. Suponha que os nós filhos dessas entradas sejam representados por entradas únicas $A1$, $B1$, $C1$ e $D1$, respectivamente. A remoção da entrada $C1$ deve causar a remoção de sua entrada pai. Todavia, para que as propriedades dos MaxRects permaneçam válidas, a área representada por C deve ser representada por algumas das demais entradas desse nó, A, B ou

D. Epstein e Hartman [EH88] não discutem como esse procedimento deve ser feito. Como estratégia mais simples, adotou-se que apenas uma única das entradas restantes deve ter seu campo MaxRect estendido, de forma a conter a área representada pela entrada removida. Porém essa estratégia nem sempre é possível. No exemplo da figura 3.8, a expansão do MaxRect, seja através de A, B ou D, gera inconsistência nas suas propriedades, dado que para essas três possibilidades a intersecção das entradas torna-se não disjunta. A solução adotada aqui para essa situação é manter a entrada. Assim, o nó continua sendo representado pelos quatro MaxRects, como apresentado na figura 3.8. Essa estratégia, entretanto, possui alguns inconvenientes:

- a existência de uma entrada obriga a existência de um nó filho a essa. Logo, no caso da figura 3.8, o nó filho da entrada C não pode ser liberado, obrigando-o a ser representado mesmo que sem entradas.

- analogamente, o outro campo dessa entrada, que representa o MBR dos nós filhos a essa, deve ser representado. Como os nós filhos a essa entrada não existem, optou-se pela associação a esse campo de um retângulo de extensão nula com valor igual ao limite inferior do MaxRect dessa entrada. Tal estratégia pode degradar as consultas. Ainda tendo como exemplo o nó da figura 3.8, após a remoção da entrada $C1$, o MaxRect da entrada pai desse nó ainda é representado pela união de A, B, C e D, contudo o outro retângulo dessa entrada pai passa a ser representado pelo retângulo tracejado da figura 3.9a, quando deveria ser representado pelo da figura 3.9b.

- suponha, ainda, através da figura 3.8, a remoção das entradas $A1$, $B1$ e $D1$. A representação desse nó passa então a ser como apresentado na figura 3.10. Entretanto, após essas remoções, tanto os nós representados por $A1$ e $C1$ deveriam ser liberados, como também os nós que contêm as entradas pai desses (representado por A e C).

Para evitar a deterioração da estrutura, que pode ocorrer a partir de sucessivas remoções, como apresentado na figura 3.10, identificou-se a possibilidade da utilização de rotinas específicas para reorganização. Porém não serão utilizadas nestes experimentos.

A tabela 3.11 apresenta a descrição da implementação da R$^+$-Tree através da quantidade de linhas de código em programa fonte.

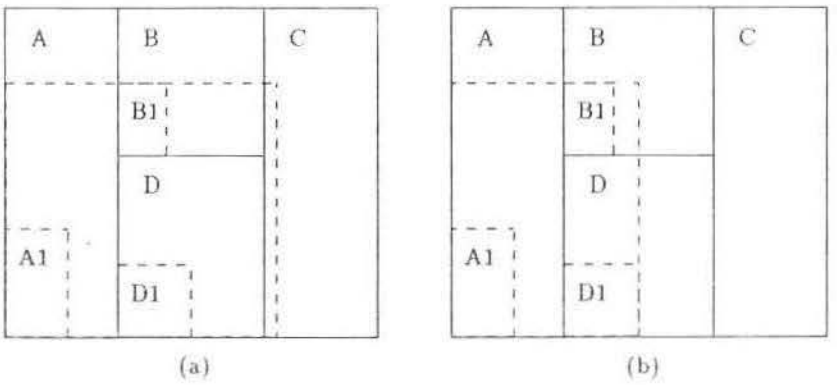


Figura 3.9: Representação do MBR da entrada pai desse nó após a remoção da entrada $C1$.

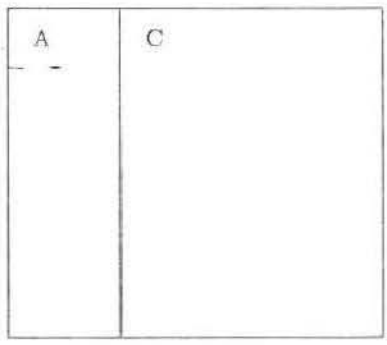


Figura 3.10: Representação de um nó após a remoção de todos os seus elementos.

Submódulos	Tamanho do Código
Inicialização	122
Inserção	640
Remoção	367
Consulta	194
Biblioteca	358

Figura 3.11: Descrição do Método de acesso R^+-Tree em linhas de código fonte.

R*-Tree

A R*-Tree foi implementada de acordo com os algoritmos sugeridos por Beckmann, Kriegel, Schneider e Seeger [BKSS90], tendo como base a implementação da R-Tree.

Na rotina de inserção, a R*-Tree diverge da R-Tree pela forma em que decide qual entrada, em um determinado nó, será utilizada para prosseguir o percurso, e pela estratégia utilizada para o *split*. Durante o percurso na estrutura, a escolha das entradas, que possuem nós folhas como filhos, é determinada pela que possui o retângulo que necessita aumentar ao mínimo a sua área de sobreposição com as demais, caso o dado seja inserido em sua subárvore. Para entradas que possuem nós intermediários como filhos, o procedimento para escolha é idêntico ao da R-Tree.

A estratégia de *split* difere tanto das propostas por Guttman como por Greene. A R*-Tree faz uso da incorporação de novos conceitos baseados na redução da área, *margin* e sobreposição dos MBRs. Conforme Beckmann [BKSS90], desde que estes três valores sejam reduzidos, a R*-Tree torna-se mais robusta para distribuições de dados adversas.

O ponto duvidoso na implementação do presente estudo, a partir da proposta por Beckmann [BKSS90], é sobre o tratamento para reinserção de entradas. Beckmann realizou experimentos para confirmar se a reinserção de elementos inseridos anteriormente na estrutura, conforme sugerido por Guttman para tratamento de *underfilled nodes*[1], melhoravam o seu desempenho. Como resultado, Beckmann determinou como melhora do desempenho valores entre 20% e 50%, dependendo dos tipos de consultas realizadas.

[1] nós que permaneceram com menos entradas do que a capacidade mínima permitida.

Comprovada a melhora do desempenho, Beckmann sugeriu a utilização da estratégia de reinserção de entradas não apenas durante o tratamento de *underfilled nodes*, mas também durante a inserção de elementos. Assim, sempre que a inserção de um elemento causar um *overflow* de um nó, sendo esse nó não raiz e esse *overflow* o primeiro ocorrido naquele nível durante a inserção desse elemento, ao invés de se utilizar a rotina de *split*, para o tratamento do *overflow*, deve ser utilizada a rotina de reinserção. Essa rotina classifica em ordem decrescente as $M+1$ entradas (sendo M o número máximo de entradas permitida por nó) de acordo com a distância do centro do seu MBR para com o centro do MBR de cada entrada (identificado pelo campo *rect*). As primeiras p entradas, sendo sugerido como valor de p 30% do valor de M, são removidas e reinseridas na estrutura, e as demais são mantidas no nó que sofreu *overflow*.

O questionamento deste autor sobre a estratégia de reinserção de entradas durante a inserção de elementos é devido ao artigo que a apresenta não ser preciso sobre a sua utilização ser indispensável ou não. Embora seja esperado um melhor desempenho para consultas, essa estratégia aumenta o custo da inserção de elementos tanto em *cpu*[2], pois a rotina de inserção é chamada mais vezes, quanto em quantidade de acessos a disco (cerca de 4%, em média, conforme Beckmann [BKSS90]). Aqui houve decisão pela análise da R*-Tree através dessas duas condições: sem utilização de reinserção para rotina de inserção de elementos, denominada neste trabalho por R*-Tree; e com a utilização de reinserção, denominada R*-Tree R.

Ainda houve a adoção de particularidades na implementação da R*-Tree R.

A especificação da rotina de reinserção, contida em [BKSS90], é incompleta. Além de não identificar o eixo, dimensão em que os elementos devem ser classificados, induz que a reinserção das entradas seja feita logo após a sua remoção, antes que o novo MBR desse nó seja propagado aos níveis superiores da estrutura. Assim, optou-se para escolha da dimensão em que os elementos são classificados, aquela cujo nó resultante, após a remoção das p entradas, possuir menor MBR. Quanto à reinserção das entradas, como sugerido, não surge efeito. Em virtude do novo MBR não ter sido propagado aos níveis superiores da estrutura, o percurso para reinserção de cada entrada provavelmente levará ao nó onde estavam armazenadas. Dessa forma, preferiu-se aqui a reinserção, feita tão logo os MBRs das diversas entradas

[2] medido através da quantidade de tempo gasto durante o processamento do algoritmo na unidade central de processamento

estejam consistentes, ao final do procedimento de remoção.

A estratégia de reinserção de entradas durante a inserção de elementos foi estendida também à rotina de remoção, já que o tratamento dos *underfilled nodes* compartilha essa rotina.

A tabela 3.12 apresenta a descrição da implementação da R*-Tree e R*-Tree R através da quantidade de linhas de código em programa fonte.

Submódulos	Tamanho do Código	
	R*-Tree	R*-Tree R
Inicialização	111	111
Inserção	493	686
Remoção	331	355
Consulta	195	195
Biblioteca	265	265

Figura 3.12: Descrição do Método de acesso R*-Tree e R*-Tree R em linhas de código fonte.

Capítulo 4

Experimentos e Análise dos Resultados

Ao contrário dos métodos de acesso utilizados nos modelos convencionais, não existe ainda um consenso sobre quais condições de dados e consultas proporcionam melhor desempenho para cada método de acesso a dados espaciais [GB90]. Logo, a escolha dos métodos mais apropriados a serem utilizados pelos SGBDs não pode ser feita com segurança, dado que não se sabe quão robusto cada método vai se comportar à diversidade de aplicações espaciais.

Para auxiliar essa escolha é proposto neste trabalho a realização de experimentos com o fim de se determinar as condições mais favoráveis de desempenho de cada método de acesso. Este trabalho critica as estratégias utilizadas nos experimentos realizados por alguns autores [Gut84, SRF87, KSSS89, Gre89], pois os seus conjuntos de dados, por serem tendenciosos, resultam em conclusões parciais. Com a definição de um conjunto de parâmetros que influenciam diretamente e peculiarmente no desempenho de cada método de acesso, propõe-se uma estratégia alternativa para realização dos experimentos, que permite uma conclusão mais completa sobre o desempenho de cada método de acesso.

4.1 Motivação

No capítulo anterior foi justificada necessidade da implementação dos métodos de acesso, dentre outros fatores, conseqüentes ao funcionamento de alguns métodos possuírem características que tornam difícil a sua representação em um modelo matemático, o que compromete o estudo analítico

sobre as estruturas de dados utilizadas por esses métodos.

Espera-se, a começar da realização de experimentos sobre os métodos de acesso, determinar as condições ideais de desempenho de cada um desses. Entretanto, o que se percebe, através dos relatos de desempenho sobre os métodos de acesso, são divergências. Restrito aos métodos de acesso implementados, esses relatos, em algumas condições, chegam a ser contraditórios. Sellis [SRF87] e Greene [Gre89], por exemplo, pelo estudo comparativo da R-Tree e R^+-Tree, concluem resultados opostos, mesmo sob condições de dados semelhantes.

Identificou-se, porém, que essas divergências encontradas são resultantes das condições de dados e consultas distintas às quais esses experimentos foram submetidos. Nesta análise sobre os diversos métodos de acesso encontrou-se um conjunto de fatores relacionados com as características de dados e consultas, denominados determinantes de desempenho, que, quando combinados diferentemente resultaram em alterações distintas de desempenho.

4.1.1 Fatores Determinantes de Desempenho

Os fatores determinantes de desempenho foram classificados da seguinte forma:

- relacionados aos dados
- relacionados às consultas

Quanto à classe de fatores relacionados aos dados, foram encontradas quatro características que interferem no desempenho dos métodos de acesso: tipo, tamanho, distribuição e dinamismo.

Os tipos de dados referem-se as formas geométricas dos dados: pontos ou sólidos no espaço. Conforme discutido no capítulo 2, alguns métodos de acesso são apropriados para o suporte a um tipo de dado específico. A tentativa de prover flexibilidade por meio do suporte aos demais tipos de dados, envolve problemas que são refletidos no desempenho. O Grid File, como proposto por Hinrichs [HN83], é um exemplo característico de como a adequação ao suporte a outros tipos de dados pode interferir no desempenho. Nesse caso, as rotinas de consulta são significativamente mais complexas e requerem uma grandeza de tempo maior durante a sua execução.

Determinou-se, também, a relação de tamanho dos objetos com o desempenho dos métodos de acesso. Observou-se que conjuntos de dados com

objetos grandes tendem a gerar sobreposições. Como essa característica é suportada pelos diversos métodos de acesso de forma variada, espera-se em alterações de desempenho distintas por parte destes. No estudo comparativo da R-Tree e da R^+-Tree, realizado por Greene [Gre89], identificou-se a superioridade de desempenho da R^+-Tree para pequenos objetos de dados, e da R-Tree para objetos grandes.

Quanto à distribuição dos dados, além de também interferir na sobreposição dos objetos, podem degenerar as estruturas de dados utilizadas pelos métodos de acesso e, conseqüentemente, comprometer o seu desempenho. Os métodos Quadtree e K-d Tree, por exemplo, não mantém sua estrutura de dados balanceada, assim, semelhante às árvores binárias, determinadas distribuições de dados podem degradar o tempo médio das consultas de logarítmico a linear.

O último fator identificado relacionado aos dados refere-se ao nível de atualização do conjunto de dados, i.e., a dinâmica dos dados. Além das operações de consulta, os métodos de acesso devem prover um suporte eficiente às operações de inserção e remoção de elementos, como também, manter o seu desempenho geral satisfatório após as sucessivas atualizações. O Grid File, por exemplo, apresenta um desempenho para *point queries* estimado em 2 acessos a disco. Todavia, as operações de inserção e remoção, por esse critério de análise, podem requisitar inúmeros acessos a disco. Já os métodos Quadtree e K-d Tree, que não possuem estruturas de dados balanceadas, apresentam a possibilidade das sucessivas remoções de elementos degenerarem a estrutura de dados, afetando, assim, o desempenho do método de acesso.

Quanto ao fator relacionado às consultas, identificou-se que nem todos os possíveis tipos de consultas são suportados eficientemente por cada método de acesso. Logo, os tipos de consultas surgem como fator determinante de desempenho dos métodos de acesso.

4.2 Definição dos Experimentos

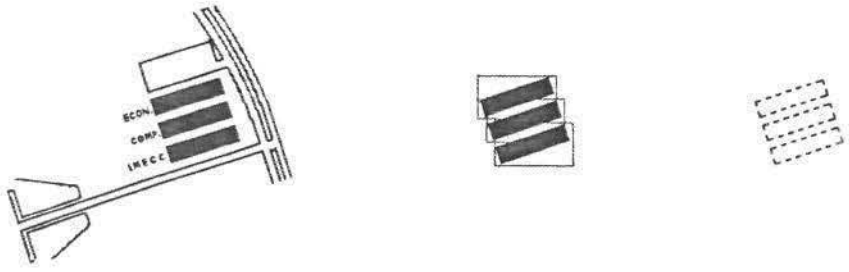
Como conseqüência da determinação desses fatores (determinantes de desempenho) torna-se interessante compreender como os métodos de acesso se comportam a sua variação, para que se possa determinar as condições ideais de desempenho de cada método de acesso e, deste modo, propor os métodos apropriados aos SGBDs.

Sabe-se que, apenas como requisito, além do gerenciamento eficiente so-

bre grande quantidade de dados armazenados em disco, estes métodos devem ser flexíveis, de modo a prover suporte eficiente à diversidade de aplicações espaciais existentes. Esse requisito motiva a estratégia utilizada por alguns autores [Gut84] de se determinar os métodos de acesso apropriados a partir da escolha de aplicações específicas, da caracterização dos tipos de dados e consultas utilizados nessas aplicações, e da posterior análise do comportamento dos diversos métodos sob essas condições.

Entretanto essa estratégia traz alguns inconvenientes. Em primeiro lugar, nem todas as possíveis condições de dados e consultas que interferem no desempenho dos métodos de acesso podem ser obtidas a partir de uma aplicação específica. As diversas aplicações espaciais possuem características de dados e consultas muito particulares. Aplicações VLSI/CAD, por exemplo, são caracterizadas por dados cujas extensões são pequenas em relação ao espaço em que estão dispostos. Aplicações geográficas, por sua vez, caracterizam-se por um volume de dados menor comparado aos da aplicação VLSI/CAD, sendo a extensão destes próxima da extensão do espaço em que estão representados. Logo, a análise dos métodos, restrita a uma determinada aplicação, pode fazer com que algumas possíveis condições de dados e consultas não sejam analisadas.

Outro inconveniente é que nem mesmo a determinação do método mais apropriado para uma aplicação específica pode ser obtida com segurança. Mesmo após o levantamento dos dados, há diferentes formas em que estes podem ser modelados para os métodos de acesso. Estas diversas formas de modelagem, além de serem dificilmente predeterminadas, podem resultar em conjuntos de dados com características completamente distintas. Suponha, por exemplo, uma aplicação geográfica onde se tenha que representar o conjunto de dados denominados de *ECON.*, *COMP.* e *IMECC*, ilustrados na figura 4.1. Utilizando a estratégia MBR para representar esses elementos, tem-se o conjunto ilustrado na figura 4.1b. Na tentativa de diminuir o *dead space* gerado por essa representação, pode-se decompor cada objeto de dados em um conjunto de MBRs menores ou em um conjunto de pontos (figura 4.1c). Observa-se que essas três estratégias geram, a contar de um mesmo conjunto de dados, elementos com características distintas a serem representados pelos métodos de acesso. Na primeira estratégia, comparada às demais, tem-se uma menor quantidade de MBRs representando uma maior quantidade de área, enquanto que, na segunda e na terceira estratégias, as relações vão mudando gradativamente, a quantidade de MBRs aumentam e a área representada por eles diminuem. Essas características fizeram descartar a utilização da referida estratégia em virtude de sua pouca eficiência.



(a) (b) (c)

Figura 4.1: Estratégias para modelagem dos dados.

Outra estratégia mais interessante foi sugerida por Kriegel [KSSS89], que ao invés de identificar possíveis aplicações e condições de dados, realizou seus experimentos sobre seis diferentes arquivos de dados e vários tipos de consulta. Notou-se que essa estratégia pode determinar, de acordo com os parâmetros utilizados para gerar os arquivos de dados, as condições ideais de desempenho de cada método de acesso.

Conhecendo-se estas condições ideais, a determinação dos métodos mais apropriados a serem utilizados pelos SGBDs é facilmente obtida, estabelecendo-se, inclusive, as melhores condições para utilização de cada um desses (tipos de dados, distribuição, tamanho, consultas e níveis de atualização). Com essa estratégia pode-se ainda determinar os métodos mais apropriados para uma aplicação específica, bastando para tal classificá-las quanto aos parâmetros utilizados nos experimentos.

O ponto desfavorável da estratégia de Kriegel, entretanto, foi a forma pela qual chegou-se à definição desses arquivos de dados e consultas. Nenhum relato foi apresentado justificando o porquê da escolha das condições de cada arquivo. Embora sendo um subconjunto dos parâmetros aqui definidos como determinantes de desempenho (relacionado aos dados e consultas), não foi utilizada essa estratégia, já que poderia-se incorrer na mesma ineficiência da anterior. Algumas condições especiais não seriam analisadas e, conseqüentemente, as conclusões sobre o desempenho dos métodos seriam incompletas.

4.2.1 Estratégia Adotada

Adotou-se como estratégia utilizada nos presentes experimentos uma solução alternativa da proposta de Kriegel. O autor deste trabalho propõe arquivos de dados e consultas para análise dos métodos de acesso, mas restringe que sua elaboração seja feita a partir dos fatores determinantes de desempenho.

Os fatores relacionados aos dados e consultas foram usados com o propósito de gerar todas as condições possíveis que podem interferir no desempenho de cada método de acesso.

A seguir, serão descritos os valores que foram atribuídos a cada fator determinante de desempenho para determinação dos arquivos de dados e consultas. A escolha desses valores foi feita através da representação de alguns mais significativos, pois seria impossível cobrir todas as possibilidades.

Tipos dos dados

Os tipos de dados referem-se às formas dos dados espaciais. Ainda que sejam as mais variadas possíveis, a partir de pontos ou sólidos no espaço, em conseqüência da estratégia de representação utilizada pelos métodos implementados, podem ser restringidas a duas: pontos, representado pelo MBR de extensão nula em todas as dimensões; e retângulos, dado que qualquer sólido no espaço será sempre representado por essa forma geométrica.

Tamanho dos dados

Para analisar o comportamento dos métodos de acesso quanto à extensão dos objetos, essa condição foi variada de modo a obter arquivos distintos: com objetos pequenos, com objetos grandes e com objetos pequenos e grandes.

A variação da extensão dos objetos foi feita apenas para tipos de dados que são retângulos, por não fazer sentido para pontos.

Foram definidos como objetos pequenos aqueles cujas extensões em cada dimensão corresponde no máximo a 2% do espaço total naquela dimensão, e os objetos grandes, como limite máximo de extensão 50% do total.

Distribuição dos dados

A análise dos métodos de acesso, quanto a esse parâmetro, torna-se difícil porque é impossível prever todos os possíveis tipos de distribuição. Por isso foram restritas a esta análise dois tipos de distribuição: uniforme e não-uniforme. O autor deste trabalho entende que a análise dos métodos

de acesso, quanto a essas distribuições, pode representar uma tendência à análise sobre demais formas.

Para distribuição uniforme, os dados foram gerados por meio de funções geradoras de números uniformes aleatórios (funções randômicas), mapeadas para o intervalo inteiro de 0 a 1.000. Os objetos de dados do tipo ponto foram criados a partir da geração de dois números; os de tipo retângulo, a partir da geração de quatro números, sendo os dois primeiros gerados para representar o centro do MBR nos eixos x e y, respectivamente; e os demais, para representar a distância do centro do MBR à sua extremidade nos eixos x e y. Estes dois últimos valores foram mapeados no intervalo de 0 a 10 e 0 a 250, de acordo com o tamanho dos dados gerados.

A criação de pontos para a distribuição não-uniforme foi feita a partir da determinação dos pontos referentes ao contorno de polígonos quadriláteros. Estes polígonos foram construídos com a geração de quatro pares de números aleatórios referentes aos seus vértices. Os retângulos pequenos sob distribuição não-uniforme foram gerados segundo a curva senóide, enquanto os retângulos grandes através da curva co-senóide.

Dinâmica dos dados

Para determinar a eficiência com que os métodos de acesso suportam inserção, remoção e atualização de elementos, foi observado, de acordo com o critério de análise (quantidade de acessos a disco), o desempenho dos métodos de acesso pela seguinte freqüência:

- da inserção sucessiva de todos os elementos de dados (10.000 elementos).
- da remoção aleatória e sucessiva de 5.000 elementos.
- da intercalação de 2.000 operações de inserção e remoção de elementos, mantendo-se ao final 5.000 elementos no arquivo.

Tipos de consultas

Com a tarefa de determinar os tipos de consultas suportadas eficientemente por cada método de acesso, foram analisados três tipos de consulta:

- *point queries* - dado um ponto P no espaço bidimensional, determine todos os retângulos R no arquivo tal que $P \in R$.

- *range queries* para determinação de intersecção - dado um retângulo S no espaço bidimensional, determine todos os retângulos R no arquivo tal que $S \cap R \neq \emptyset$.

- *range queries* para determinação de inclusão - dado um retângulo S no espaço bidimensional, determine todos os retângulos R no arquivo tal que $S \subseteq R$.

Consultas do tipo *range queries* para determinação de não-inclusão[1] deixarão de ser executadas, uma vez que apresentarão resultados semelhantes às consultas para determinação de intersecção. Embora essas consultas tenham o objetivo de determinar relacionamentos diferentes, o procedimento para determinação do relacionamento de não-inclusão difere do procedimento para determinação de intersecção apenas nos nós que são folhas da estrutura e, portanto, não requisitam demais acessos a disco.

Esses três tipos de consultas foram submetidas independentemente a cada método de acesso sob cada conjunto de dados utilizado.

Point queries foram criadas a partir da geração de 200 números aleatórios e uniformes representando as coordenadas nos eixos x e y respectivamente.

Para a criação dos retângulos referentes às consultas do tipo *range queries* foram gerados tamanhos e formatos variados. Para o formato dos retângulos definiu-se o formato quadrado, em que a extensão nas duas dimensões é idêntica; e o formato linear, em que a extensão em uma dimensão é três vezes maior que a da outra. Quanto ao tamanho dos retângulos, houve variação da extensão em cada dimensão nas faixas de 0% a 2%, 2% a 10%, 10% a 20%, 20% a 40% e 40% a 60% da extensão total naquela dimensão. O centro dos retângulos foi gerado por funções aleatórias e uniformes, sendo as suas extensões mapeadas de acordo com as faixas descritas acima. A cada faixa de tamanho e formato, foram criados 20 retângulos, totalizando 200 *range queries* por arquivo.

Apesar de consultas para determinação de inclusão não serem muito apropriadas para arquivos de dados formados exclusivamente por pontos, estas foram submetidas para completar os relatos de desempenho.

As consultas foram realizadas após o término de cada fase definida durante a discussão do parâmetro de dinâmica dos dados, com o objetivo também de complementar a sua análise.

[1] onde dado um retângulo S no espaço bidimensional, determina todos os retângulos R no arquivo tal que $S \supseteq R$.

4.2.2 Dados e Consultas

Tendo como base os valores atribuídos aos fatores determinantes de desempenho, determinou-se 14 arquivos de dados diferentes e 2 arquivos de consultas.

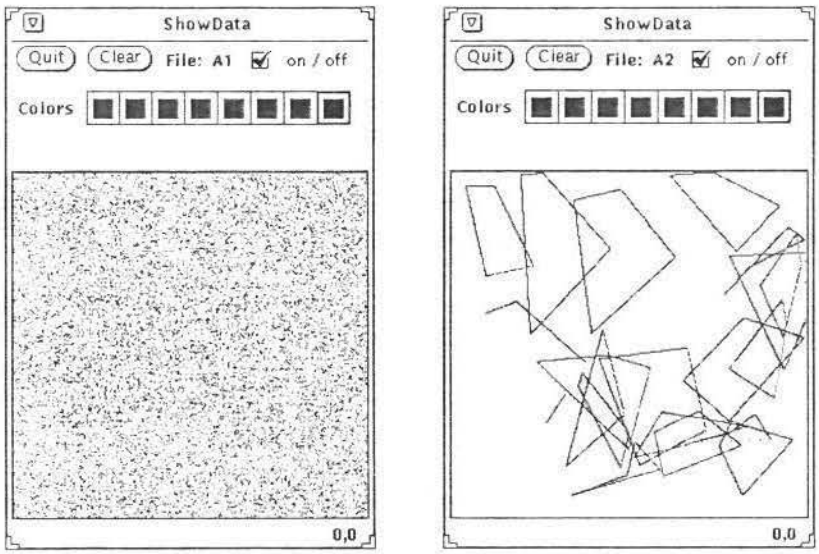
Os arquivos formados por dois ou mais tipos de dados foram criados a partir dos arquivos mais simples compostos de pontos, retângulos pequenos e retângulos grandes.

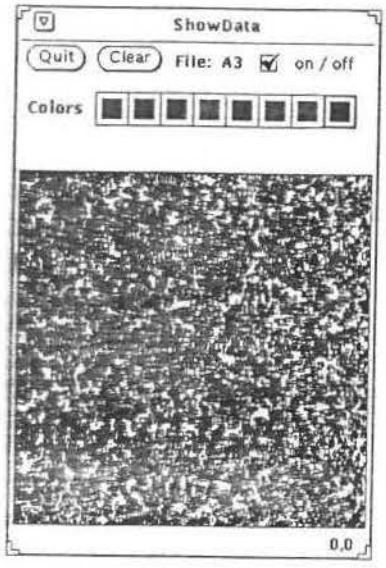
A seguir, serão identificados os arquivos de dados a serem utilizados nos experimentos:

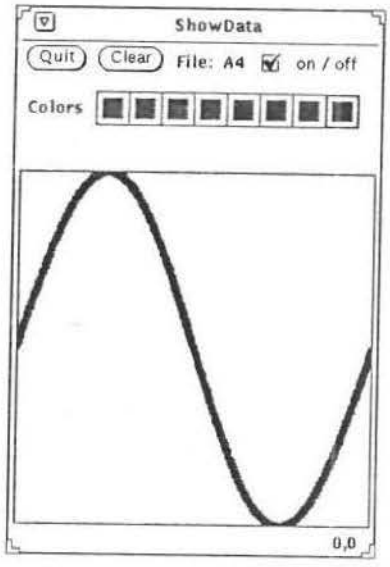
- (A1) - composto de 10.000 objetos de dados representados por pontos sob distribuição uniforme.
- (A2) - composto de 10.000 objetos de dados representados por pontos sob distribuição não-uniforme.
- (A3) - composto de 10.000 objetos de dados representados por retângulos pequenos sob distribuição uniforme.
- (A4) - composto de 10.000 objetos de dados representados por retângulos pequenos sob distribuição não-uniforme.
- (A5) - composto de 10.000 objetos de dados representados por retângulos grandes sob distribuição uniforme.
- (A6) - composto de 10.000 objetos de dados representados por retângulos grandes sob distribuição não-uniforme.
- (A7) - composto de 10.000 objetos de dados representados por retângulos pequenos e grandes sob distribuição uniforme.
- (A8) - composto de 10.000 objetos de dados representados por retângulos pequenos e grandes sob distribuição não-uniforme.
- (A9) - composto de 10.000 objetos de dados representados por pontos e retângulos pequenos sob distribuição uniforme.
- (A10) - composto de 10.000 objetos de dados representados por pontos e retângulos pequenos sob distribuição não-uniforme.
- (A11) - composto de 10.000 objetos de dados representados por pontos e retângulos grandes sob distribuição uniforme.

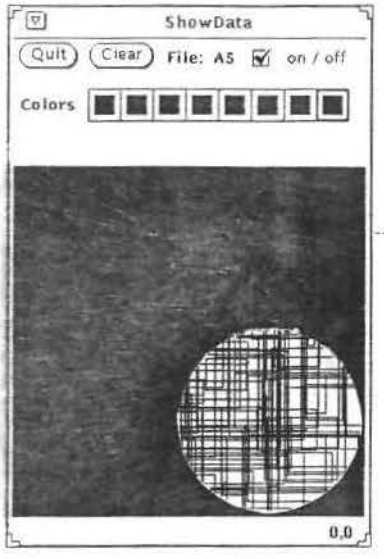
- (A12) - composto de 10.000 objetos de dados representados por pontos e retângulos grandes sob distribuição não-uniforme.
- (A13) - composto de 10.000 objetos de dados representados por pontos, retângulos pequenos e grandes sob distribuição uniforme.
- (A14) - composto de 10.000 objetos de dados representados por pontos, retângulos pequenos e grandes sob distribuição não-uniforme.

A figura 4.2 ilustra o conjunto de dados formado por esses arquivos. A ilustração dos dados referentes às consultas é apresentada na figura 4.3.

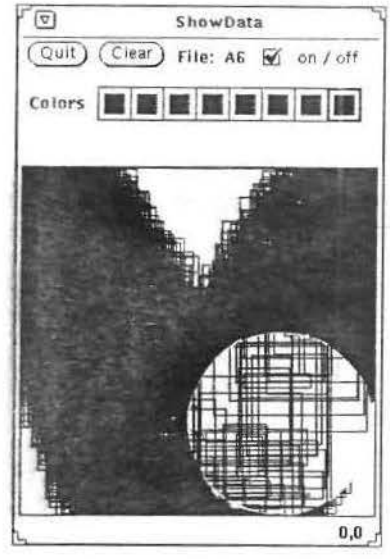




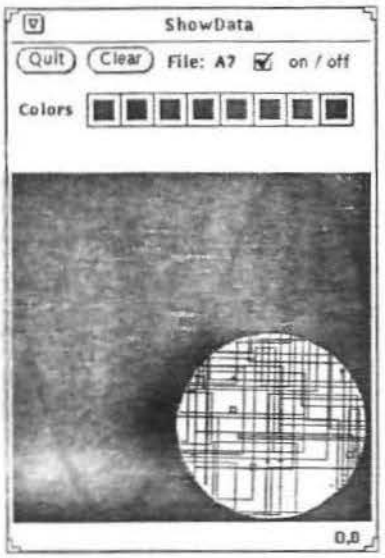




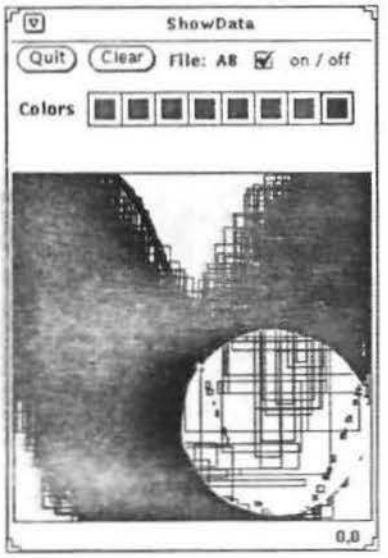
* o círculo ilustra a instância do arquivo na inserção do ducentésimo objeto.



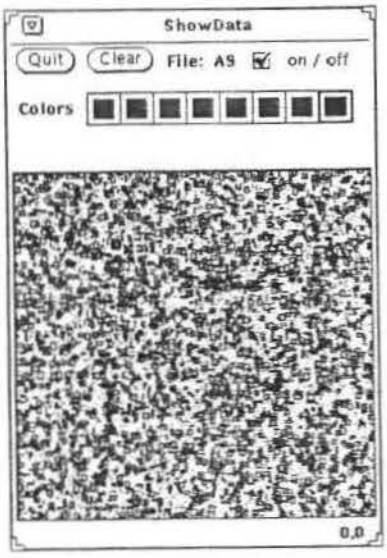
* o círculo ilustra a instância do arquivo na inserção do ducentésimo objeto.

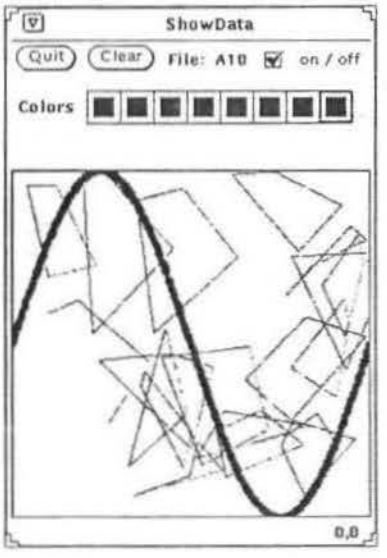


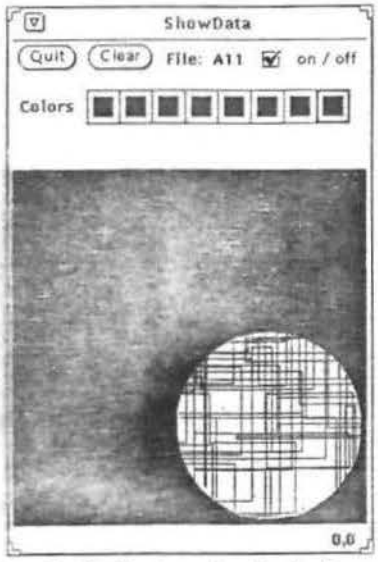
* o círculo ilustra a instância do arquivo na inserção do ducentésimo objeto.



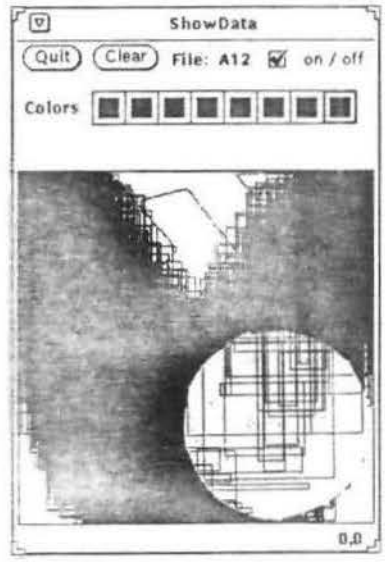
* o círculo ilustra a instância do arquivo na inserção do ducentésimo objeto.



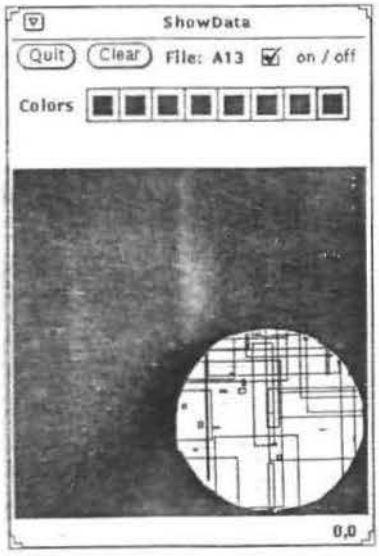




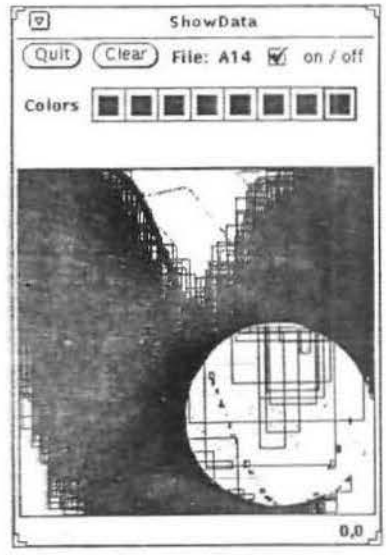
* o círculo ilustra a instância do arquivo na inserção do ducentésimo objeto.



* o círculo ilustra a instância do arquivo na inserção do ducentésimo objeto.



* o círculo ilustra a instância do arquivo na inserção do ducentésimo objeto.



* o círculo ilustra a instância do arquivo na inserção do ducentésimo objeto.

Figura 4.2: Ilustração dos arquivos de dados.

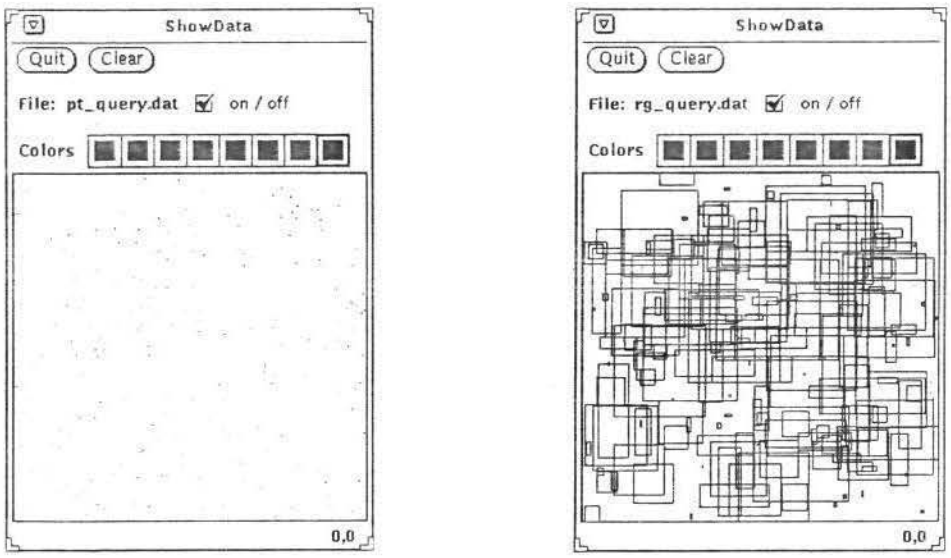


Figura 4.3: Ilustração dos arquivos de consulta.

4.2.3 Parâmetros da Implementação

Para definição dos parâmetros necessários por parte dos métodos de acesso, utilizou-se os sugeridos na sua especificação original. Assim, sendo o tamanho da página de disco igual a 1.024 bytes, foi aqui usado como parâmetro para os valores de M (o número máximo de entradas permitido por nó), m (número mínimo de entradas permitido por nó) e p (quantidade de elementos

Parâmetros	R-Tree	R-Tree Greene	R^+-Tree	R*-Tree	R*-Tree R
M	50	50	28	50	50
m	20	20	1	20	20
p	-	-	-	-	15

Figura 4.4: Definição dos parâmetros utilizados nos métodos de acesso.

a serem reinseridos)[2], os apresentados na tabela 4.4.

Ainda foi expresso como parâmetro para a quantidade de páginas úteis pelo *buffer pool*, o espaço referente a 50 páginas de disco.

4.3 Apresentação dos Resultados

Os resultados obtidos nos experimentos são apresentados de acordo com as fases em que estes foram realizados:

- fase1
  - inserção de 10.000 objetos de dados.
  - realização de *point queries*.
  - realização de *range queries* para determinação de intersecção.
  - realização de *range queries* para determinação de inclusão.
- fase2
  - remoção de 5.000 objetos de dados.
  - realização de *point queries*.
  - realização de *range queries* para determinação de intersecção.
  - realização de *range queries* para determinação de inclusão.
- fase3
  - inserção e remoção totalizando 2.000 objetos de dados.
  - realização de *point queries*.
  - realização de *range queries* para determinação de intersecção.

[2] apenas para o métodos R*-Tree R.

– realização de *range queries* para determinação de inclusão.

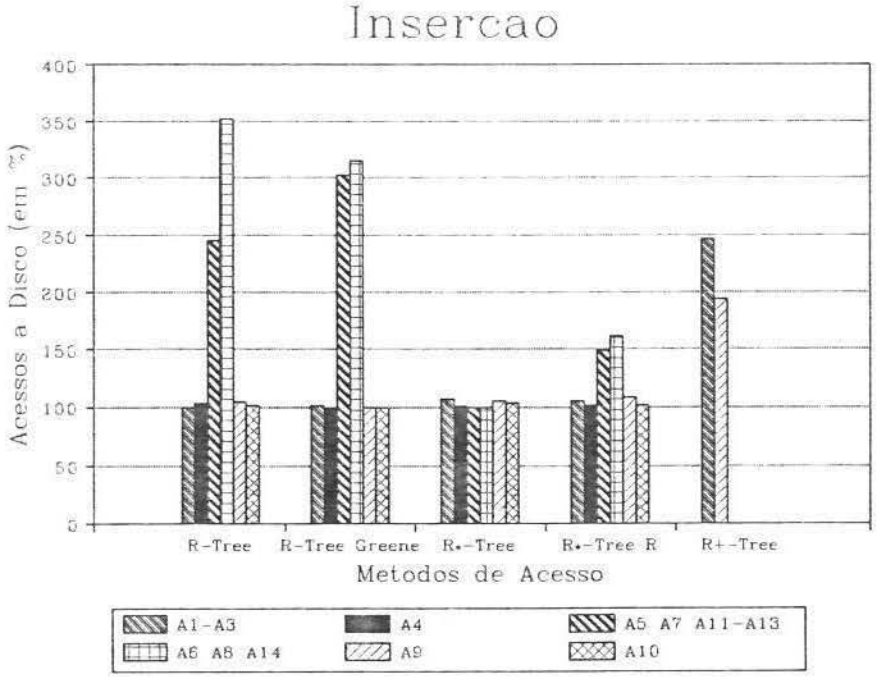
Através da análise do desempenho dos diversos arquivos sob todos os métodos de acesso, verificou-se que muitos arquivos possuíam comportamentos semelhantes em relação aos métodos de acesso testados. Essa característica possibilitou a representação dos resultados através de gráficos mais simples.

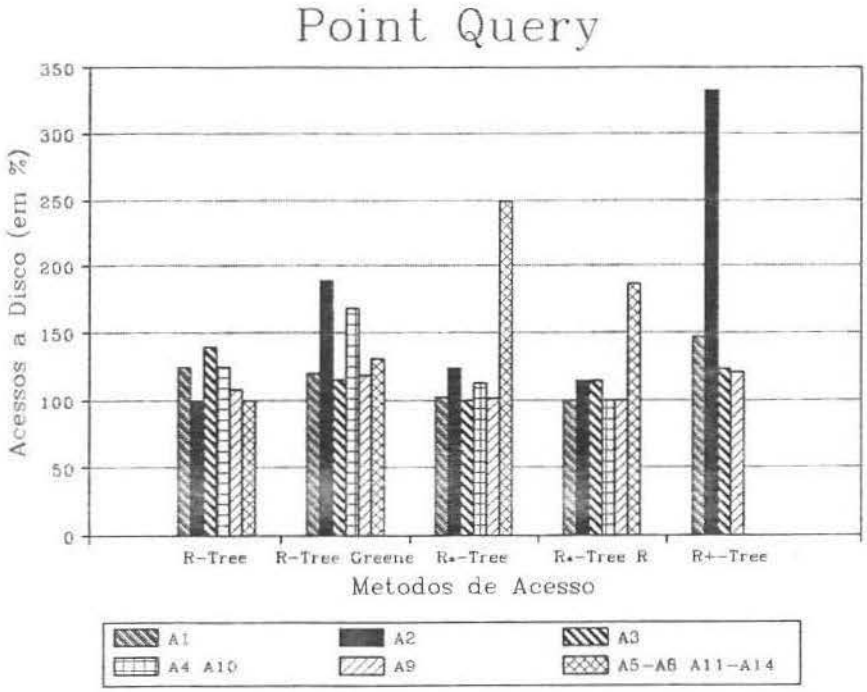
Todos os arquivos que apresentaram comportamentos semelhantes foram representados por um único. Seus resultados de desempenho referentes aos métodos de acesso foram apresentados de acordo com a média desses valores para cada método de acesso.

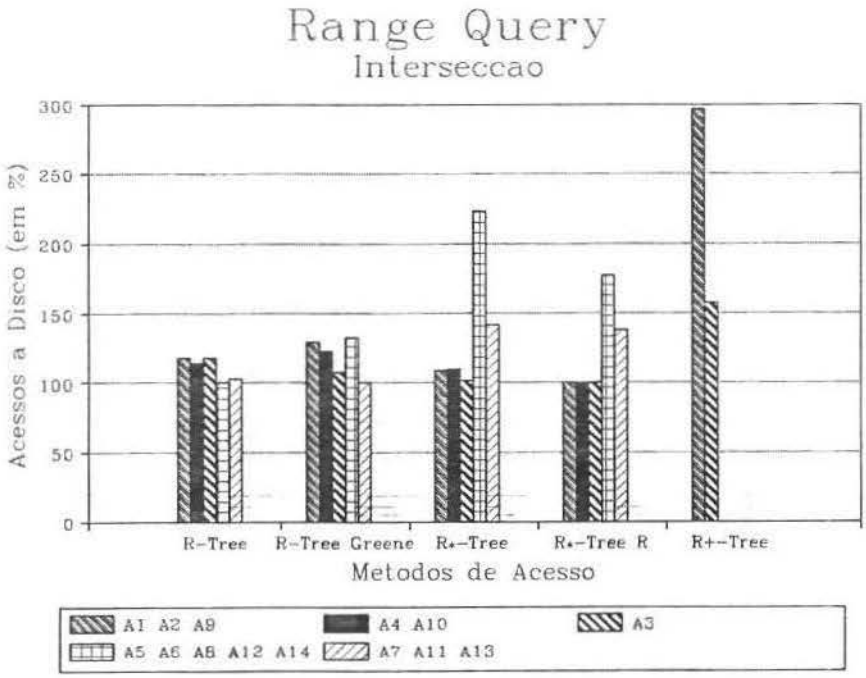
Os resultados obtidos para cada arquivo, referentes à quantidade de acessos a disco requisitados pelos métodos, foram normalizados em função do método que requisitou a menor quantidade. A esse método foi atribuído o valor de 100% e aos demais foi calculado a variação em porcentagem a partir desse valor.

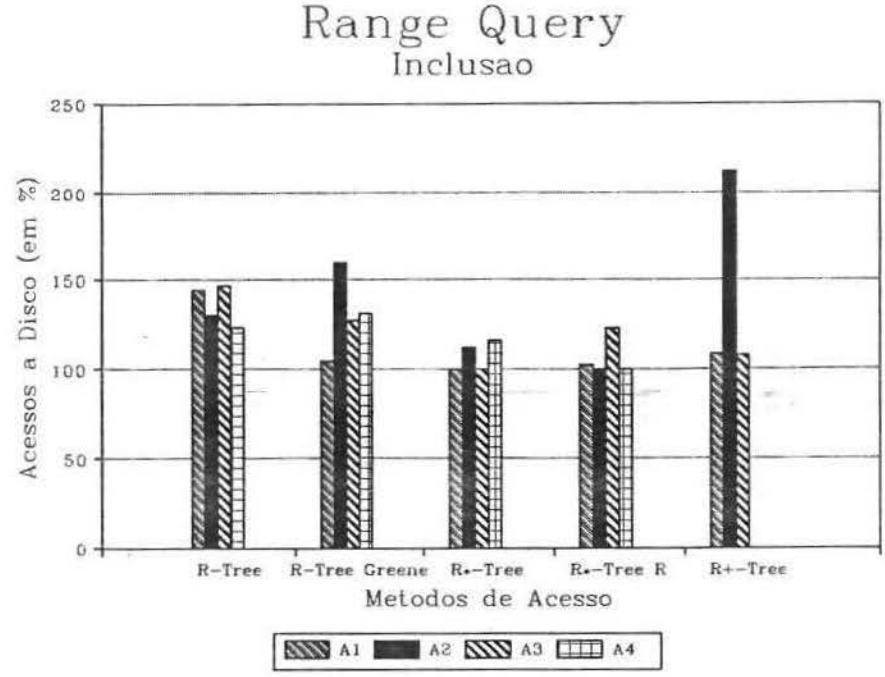
A seguir serão apresentados os gráficos obtidos.

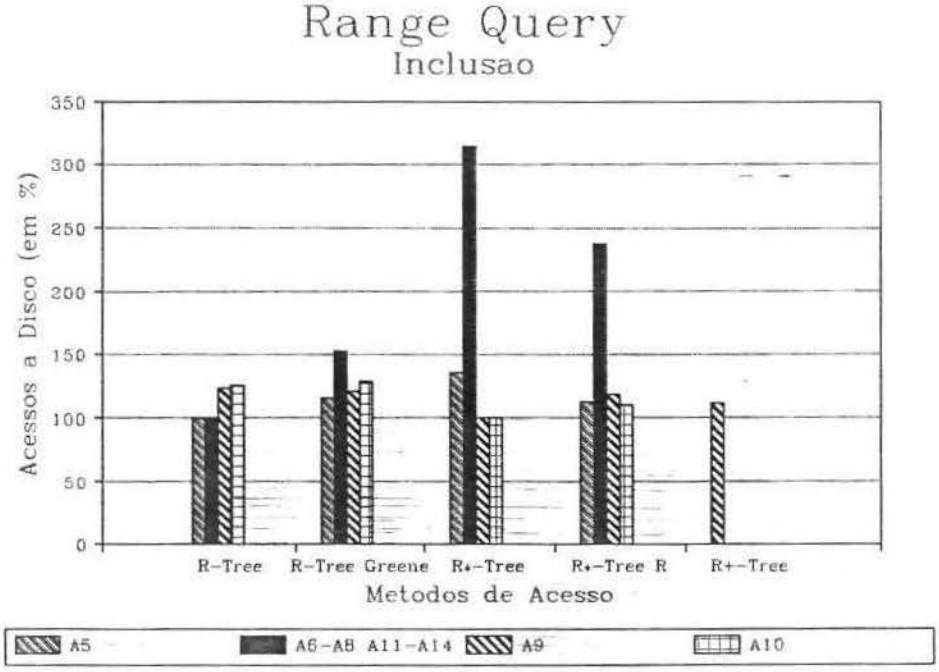
4.3.1 Fase 1



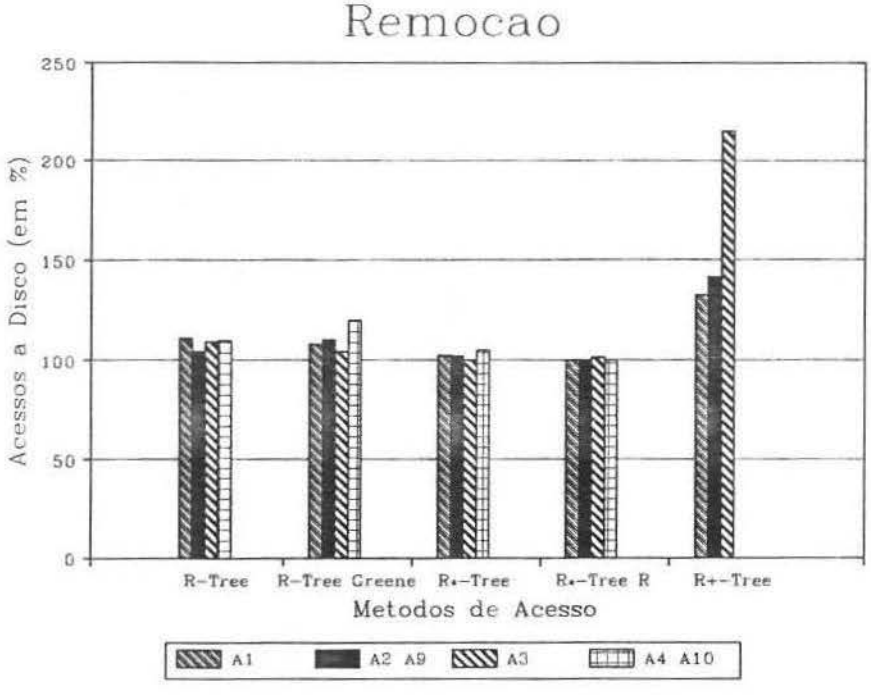

Point Query
Acessos a Disco (em %)
350
300
250
200
150
100
50
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A1
A2
A3
A4 A10
A9
A5-A8 A11-A14

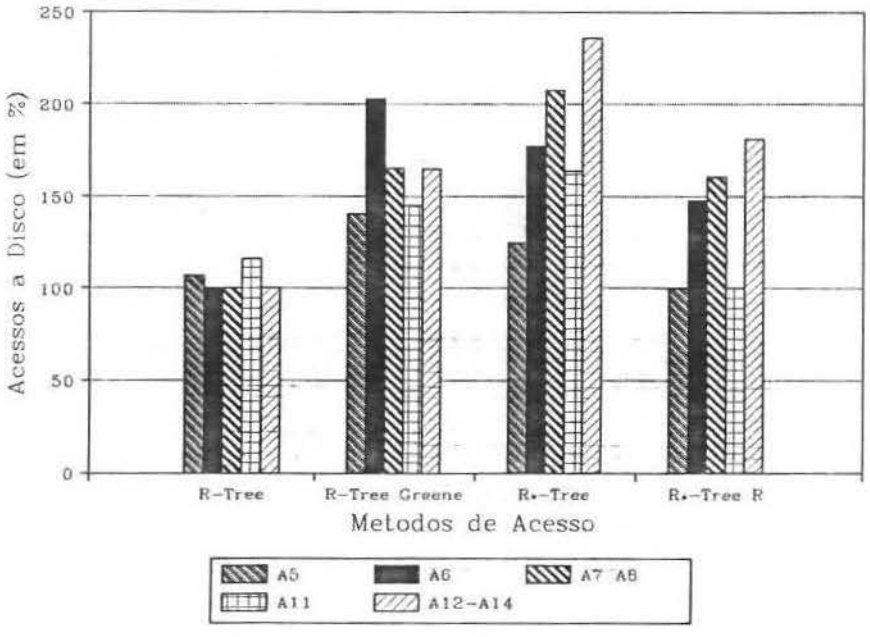

Range Query
Interseccao
Acessos a Disco (em %)
300
250
200
150
100
50
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A1 A2 A9
A4 A10
A3
A5 A6 A8 A12 A14
A7 A11 A13

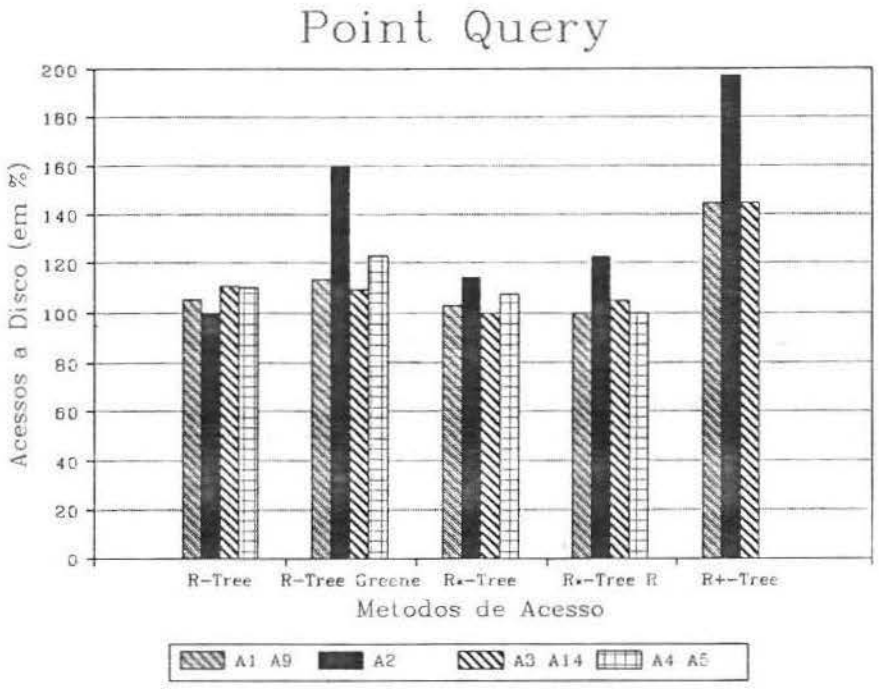

Range Query
Inclusao
Acessos a Disco (em %)
250
200
150
100
50
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A1
A2
A3
A4

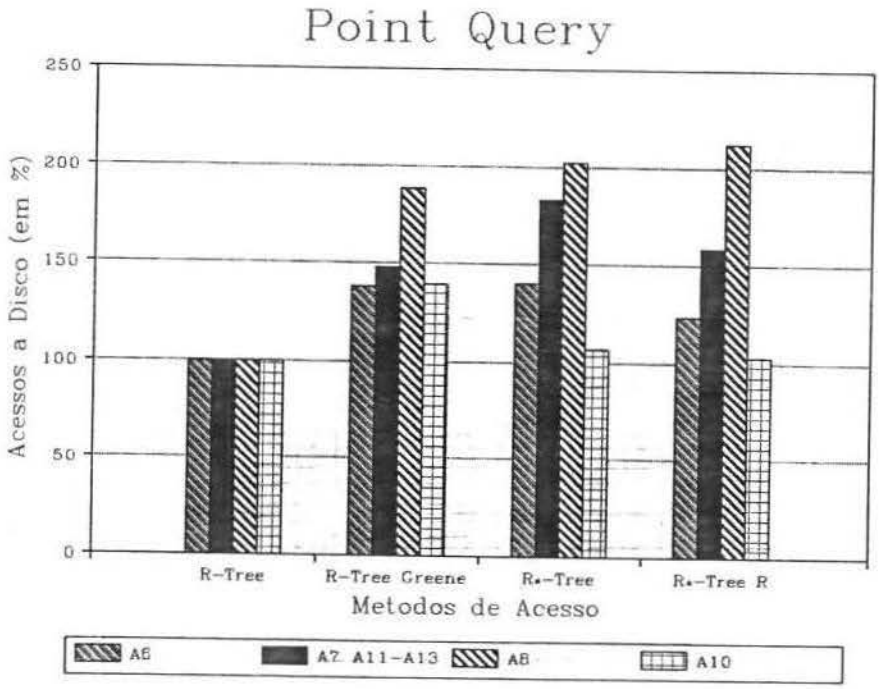

Range Query
Inclusao
Acessos a Disco (em %)
350
300
250
200
150
100
50
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A5
A6-A8 A11-A14
A9
A10

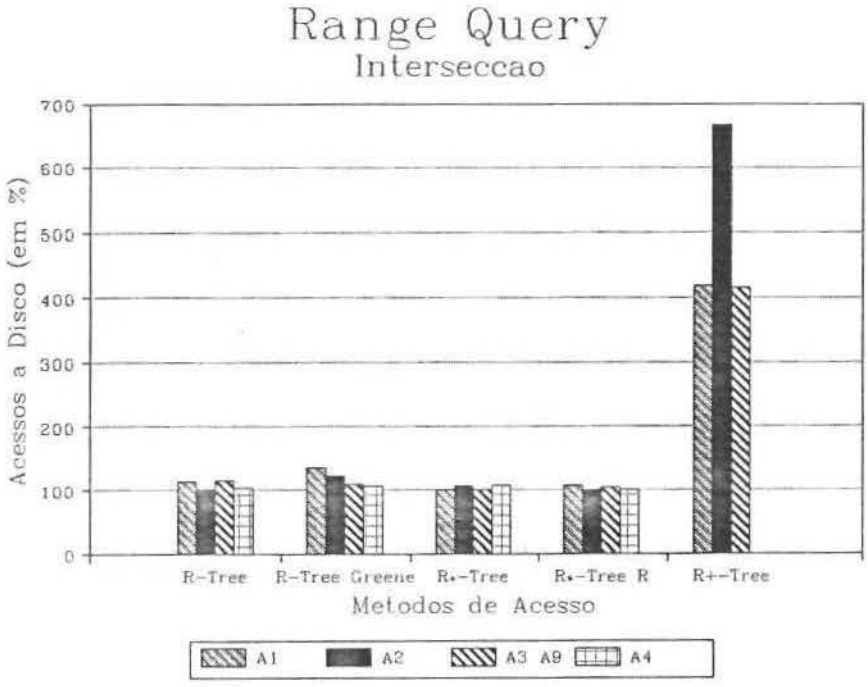
4.3.2 Fase 2

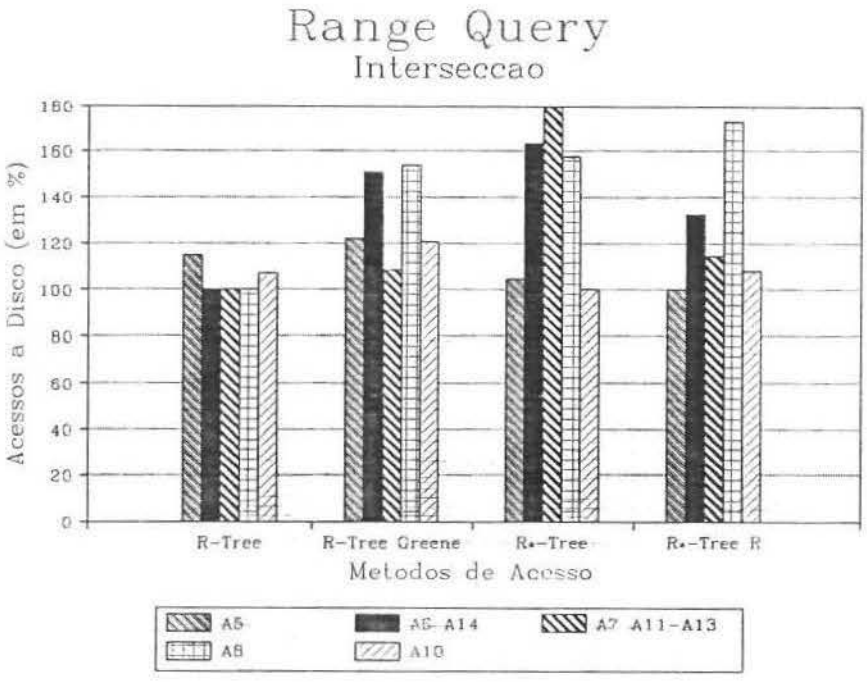

Remocao
250
200
150
100
50
0
Acessos a Disco (em %)
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A1
A2 A9
A3
A4 A10

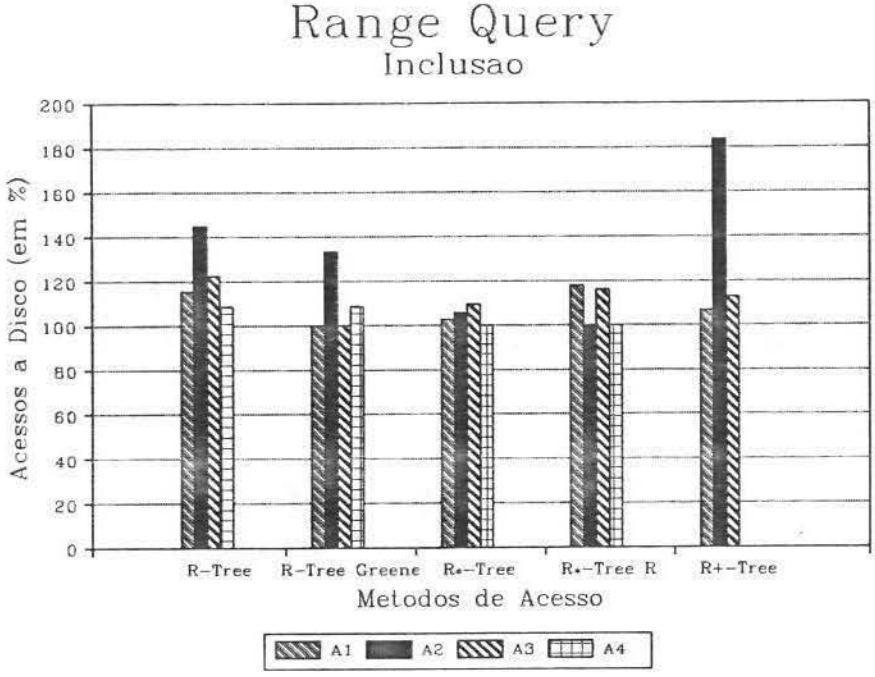

Remocao
250
200
150
100
50
0
Acessos a Disco (em %)
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
Metodos de Acesso
A5
A6
A7 A8
A11
A12-A14

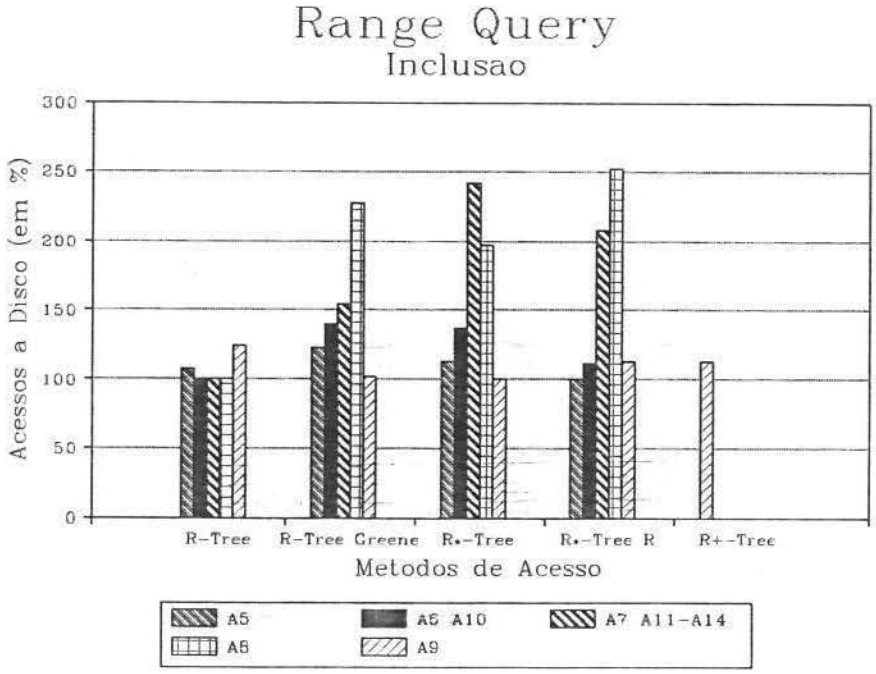

Point Query
Acessos a Disco (em %)
200
180
160
140
120
100
80
60
40
20
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A1 A9
A2
A3 A14
A4 A5


Point Query
Acessos a Disco (em %)
250
200
150
100
50
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
Metodos de Acesso
A6
A7 A11-A13
A8
A10

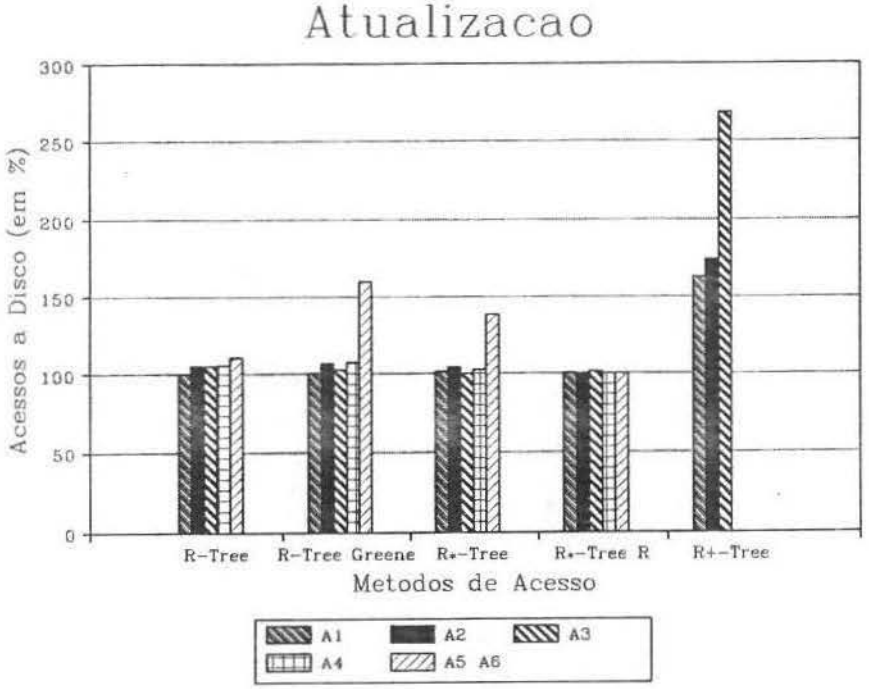

Range Query
Interseccao
Acessos a Disco (em %)
700
600
500
400
300
200
100
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A1
A2
A3 A9
A4

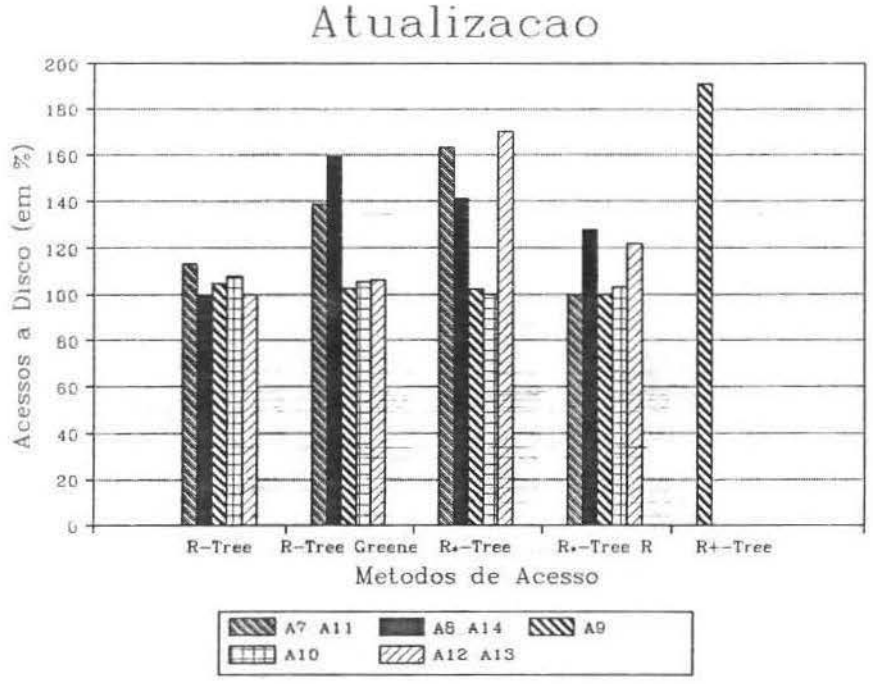

Range Query
Interseccao
Acessos a Disco (em %)
180
160
140
120
100
80
60
40
20
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
Metodos de Acesso
A5
A6 A14
A7 A11-A13
A8
A10

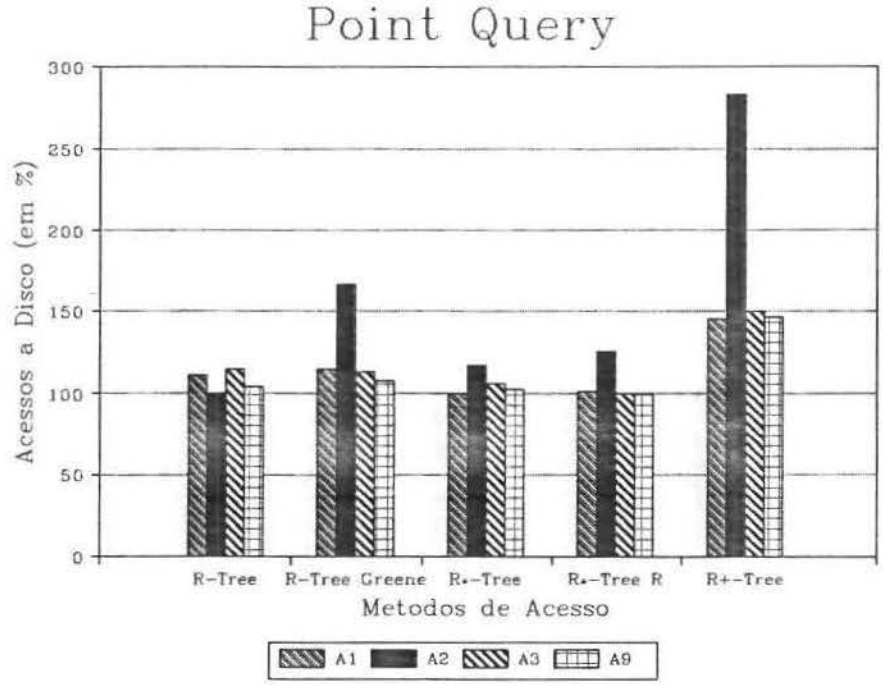

Range Query
Inclusao
Acessos a Disco (em %)
200
180
160
140
120
100
80
60
40
20
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A1
A2
A3
A4

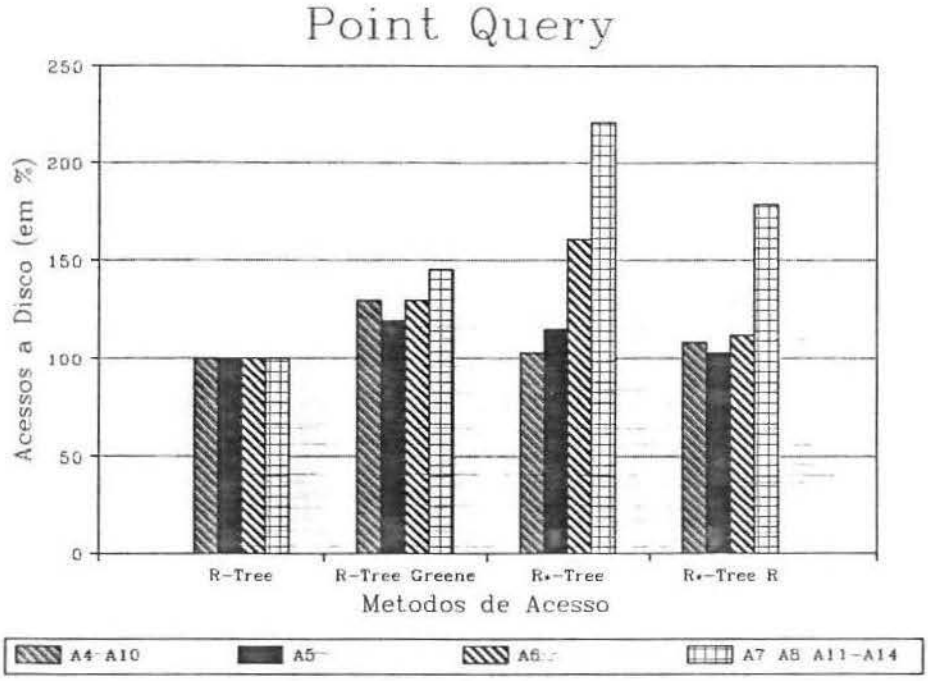

Range Query
Inclusao
Acessos a Disco (em %)
300
250
200
150
100
50
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A5
A6 A10
A7 A11-A14
A8
A9

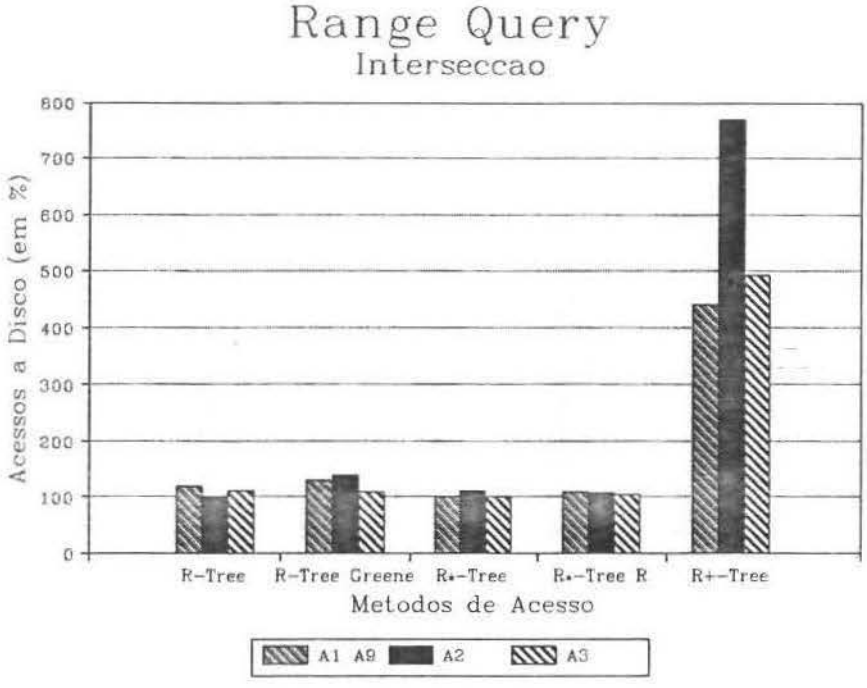
4.3.3 Fase 3


Atualizacao
Acessos a Disco (em %)
300
250
200
150
100
50
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A1
A2
A3
A4
A5 A6

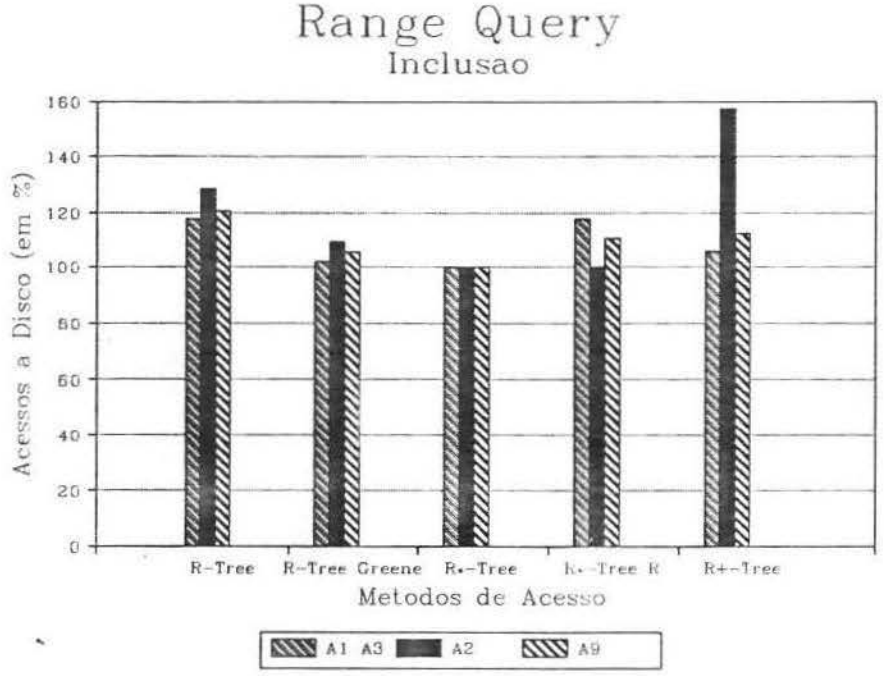

Atualizacao
Acessos a Disco (em %)
200
180
160
140
120
100
80
60
40
20
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A7 A11
A8 A14
A9
A10
A12 A13

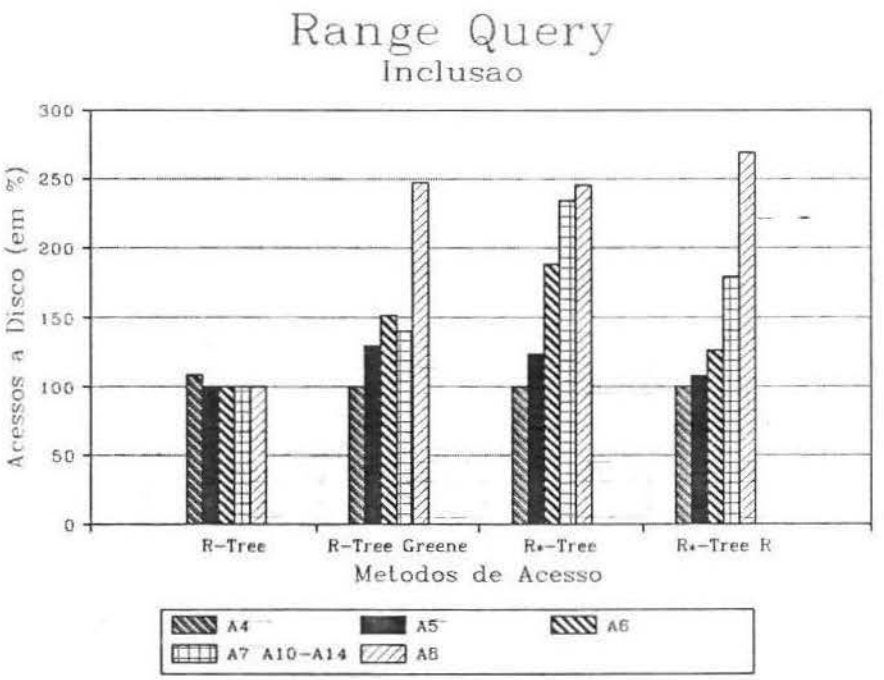

Point Query
Acessos a Disco (em %)
300
250
200
150
100
50
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A1
A2
A3
A9


Point Query
Acessos a Disco (em %)
250
200
150
100
50
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
Metodos de Acesso
A4-A10
A5
A6
A7 A8 A11-A14


Range Query
Interseccao
Acessos a Disco (em %)
800
700
600
500
400
300
200
100
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
R+-Tree
Metodos de Acesso
A1 A9
A2
A3


Range Query
Interseccao
Acessos a Disco (em %)
180
160
140
120
100
80
60
40
20
0
R-Tree
R-Tree Greene
R*-Tree
R*-Tree R
Metodos de Acesso
A4
A5
A6
A7 A11
A8 A12-A14
A10


Range Query
Inclusao
Acessos a Disco (em %)
160
140
120
100
80
60
40
20
0
R-Tree
R-Tree Greene
R*-Tree
R+-Tree R
R+-Tree
Metodos de Acesso
A1 A3
A2
A9


Range Query
Inclusao
Acessos a Disco (em %)
300
250
200
150
100
50
0
R-Tree
R-Tree Greene
R*-Tree
R+-Tree R
Metodos de Acesso
A4
A5
A6
A7 A10-A14
A8

4.4 Interpretação dos Resultados

Para avaliar quão robusto cada método se comportou durante a realização dos experimentos, os resultados obtidos nos gráficos foram interpretados através da análise conjunta das três fases de experimentos, restrita às operações semelhantes que foram submetidas em cada uma dessas fases (atualização, *point queries*, *range queries* para determinação de intersecção e *range queries* para determinação de inclusão).

4.4.1 Atualização (Inserções e Remoções)

Durante a inserção de elementos (fase 1), os métodos R*-Tree e a R*-Tree R foram os que requisitaram menor quantidade de acessos a disco para todos os arquivos que envolveram objetos grandes. Em particular, o método R*-Tree apresentou resultados ainda mais satisfatórios que o R*-Tree R.

Nos demais arquivos (A1, A2, A3, A4, A9 e A10), observou-se uma tendência de melhor desempenho para os métodos R-Tree e R-Tree Greene. O método R-Tree mostrou-se mais apropriado para arquivos compostos apenas por pontos, independentemente da distribuição utilizada, e por pequenos retângulos sob distribuição uniforme; o R-Tree Greene mostrou-se mais apropriado para suportar o conjunto formado de pontos e retângulos pequenos, independentemente da distribuição, e de retângulos pequenos sob distribuição não-uniforme.

Uma característica importante é que para os arquivos onde os métodos R-Tree e R-Tree Greene mostraram-se mais apropriados, os métodos R*-Tree e R*-Tree R obtiveram valores sempre muito próximos. Entretanto, para os demais arquivos onde determinou-se a superioridade do R*-Tree e R*-Tree R, a discrepância aos valores dos métodos R-Tree e R-Tree Greene foi bastante significativa.

No processo de remoção (fase 2) foram identificadas relações de desempenho inversas à obtidas durante a inserção. A maioria dos arquivos formados por retângulos grandes favoreceu ao método R-Tree. De maneira análoga, aos demais arquivos, o método de acesso R*-Tree R obteve melhores resultados.

Os resultados obtidos durante a atualização dos dados (fase 3) não possibilitaram a determinação de uma tendência de desempenho como às obtidas

durante a fase 1 e a 2 (inserção e remoção). Porém, observou-se uma indicação de melhor desempenho para os métodos R-Tree e R*-Tree R. Em particular, o método R-Tree apresentou-se mais robusto, pois mesmo para os arquivos de dados em que ele não obteve melhor desempenho, seus resultados sempre se aproximaram destes.

4.4.2 Point Queries

Durante a fase 1, o método R-Tree mostrou-se mais apropriado para todos os arquivos que envolveram objetos grandes. O método R*-Tree, que suportou a inserção desses dados com melhor eficiência, apresentou os piores resultados. O R-Tree ainda apresentou melhor desempenho ao conjunto composto de pontos sob distribuição não-uniforme. Para os demais arquivos (A1, A3, A4, A9 e A10) formados por pontos sob distribuição uniforme, retângulos pequenos e pela combinação desses dois últimos arquivos, o R*-Tree e R*-Tree R mostraram-se mais adequados, com uma melhor relação para esse último método.

Os resultados obtidos na fase 2 foram semelhantes aos obtidos na fase 1. O método de acesso R-Tree apresentou três alterações de desempenho. Este não mais apresentou os melhores resultados para o arquivo formado por retângulos grandes sob distribuição uniforme (A5), nem para o arquivo formado de pontos, retângulos pequenos e grandes sob distribuição não-uniforme (A14). Em contrapartida, no arquivo formado por pontos e retângulos pequenos (A10), o método R-Tree apresentou melhores resultados. Em relação aos métodos R*-Tree e R*-Tree R, estes obtiveram também os melhores resultados para os arquivos A14 e A5 respectivamente.

Após a intercalação de diversas operações de inserção e remoção (fase 3), o método R-Tree apresentou os melhores resultados em quase todos os arquivos de dados. Apenas para os arquivos formados exclusivamente por pontos, retângulos pequenos e pela combinação destes dois tipos, sempre sobre distribuição uniforme, é que o método R-Tree não obteve os melhores resultados (A1, A3 e A9). Nesses casos, os melhores resultados foram obtidos pelos métodos R*-Tree e R*-Tree R.

4.4.3 *Range Queries* para determinação de Intersecção

Às consultas realizadas após a inserção de elementos (fase 1), identificou-se melhor desempenho para os métodos R-Tree e R-Tree Greene quando os conjuntos de dados foram formados por retângulos grandes. Nesses casos,

verificou-se uma tendência para o melhor desempenho do método R-Tree Greene quando os arquivos foram formados a partir de distribuições uniformes.

Os métodos de acesso R*-Tree e R*-Tree R demonstraram-se mais apropriados para os demais arquivos de dados (A1, A2, A3, A4, A9 e A10), obtendo o método R*-Tree os melhores resultados.

Esses resultados de desempenho permaneceram com poucas alterações, mesmo após a realização das remoções e atualizações sobre os arquivos de dados (fases 2 e 3).

O arquivo formado por retângulos grandes sob distribuição uniforme (A5), após a remoção de elementos, proporcionou desempenho mais satisfatório ao método R*-Tree R. Contudo, após a atualização do arquivo através da seqüência de operações de inserção e remoção, o método R-Tree tornou a obter o melhor resultado.

4.4.4 *Range Queries* para determinação de Inclusão

Na primeira realização dessas consultas, após a inserção de elementos, o método R-Tree apresentou resultados mais satisfatórios para os arquivos de dados formados de objetos grandes.

Nas demais distribuições (A1, A2, A3, A4, A9 e A10), identificou-se a superioridade do R*-Tree, com exceção dos arquivos formados exclusivamente por pontos e retângulos pequenos, ambos sob distribuição não-uniforme (A2 e A4), onde o R*-Tree R mostrou-se ligeiramente superior.

De forma análoga aos demais tipos de consultas, estes resultados preservaram-se com poucas alterações durante as outras realizações dessas consultas.

Após a remoção de elementos, os resultados mais favoráveis obtidos pelo método R*-Tree, através dos arquivos A1, A3 e A10, foram atribuídos aos métodos R-Tree Greene e R-Tree. Em contrapartida, os melhores resultados para o arquivo A5 foram obtidos para o método R*-Tree R.

Esses últimos resultados para os arquivos A1, A3 e A5, após a atualização dos elementos (fase 3), foram revertidos aos mesmos resultados obtidos durante a fase 1: aos métodos R*-Tree R, para os arquivos A1 e A3, e ao método R-Tree, para o arquivo A5.

4.5 Conclusão

Identificou-se aqui uma tendência no desempenho dos métodos de acesso em relação aos arquivos que foram testados. Os arquivos compostos por objetos grandes, mesmo que não exclusivamente, obtiveram, na maioria das vezes, resultados semelhantes quando comparados ao desempenho dos diversos métodos de acesso. Em quase todos os tipos de consultas realizadas durante as três fases, o método R-Tree apresentou os melhores resultados referentes a esse conjunto de arquivos, mostrando-se, então, como o método de acesso mais apropriado para prover suporte a conjunto de dados com estas características (objetos grandes).

Aos arquivos de dados compostos por pontos, retângulos pequenos e pela combinação destes tipos, os métodos R*-Tree e R*-Tree R se mostraram mais apropriados. Os resultados de desempenho foram freqüentemente alternados entre esses métodos, porém se mantiveram sempre muito próximos.

O método de acesso R$^+$-Tree não suportou todos os arquivos de dados. Os arquivos A4, A5, A6, A7, A8, A10, A11, A12, A13 e A14 apresentaram números excessivos de sobreposições entre os objetos de dados que impossibilitaram a determinação do eixo de partição durante a rotina de *split* (na figura 4.5 é ilustrado um conjunto de dados com essas condições). Ademais, o pouco número de entradas permitido por nó, em relação aos demais métodos, resultou em quantidades de acessos a disco muito superiores. Apenas durante a realização de consultas para determinação de inclusão os resultados da R$^+$-Tree se mostraram competitivos.

Quanto ao método de acesso R-Tree, proposto por Greene, em poucas condições obteve melhor desempenho que o método R-Tree, como proposto originalmente por Guttman.

Neste trabalho foi observado que, entre os tipos de consultas efetuadas, as alterações de desempenho obtidas pelos métodos de acesso a um mesmo arquivo de dados, não justificam a implementação de vários métodos, por parte dos SGBDs, com a finalidade de prover suporte eficiente aos diversos tipos de consulta. Essas alterações, quando encontradas, refletiram a mudança em relação a métodos de acesso que obtiveram valores de desempenho sempre muito próximos.

O fato do desempenho obtido nas consultas ser preservado nas três fases em que os experimentos foram realizados, levou a comprovar as estruturas de dados dinâmicas utilizadas por esses métodos, que suportaram eficientemente a atualização dos elementos.

Quanto à quantidade de espaço requerida por cada método de acesso,

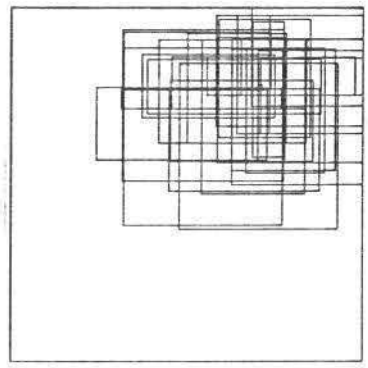


Figura 4.5: Ilustração de uma condição de dados que impossibilitou o funcionamento do método R^+-Tree.

conforme a figura 4.6, percebe-se que, com exceção ao método R^+-Tree, em média, os métodos de acesso obtiveram comportamento semelhantes. O método R^*-Tree R requisitou uma quantidade maior de espaço, comparado aos métodos R-Tree, R-Tree Greene e R^*-Tree, provavelmente, em decorrência da política de reinserção de entradas ocasionar uma maior quantidade de *splits*. O método R^+-Tree, diferente dos demais, requisitou uma quantidade de espaço bem superior. Entretanto, esse valor é justificado pela menor capacidade de entradas por nó, cerca de 56% comparada a capacidade dos demais métodos, e pela subdivisão dos objetos de dados. A figura 4.7 apresenta, a partir da figura 4.6, outra relação sobre a quantidade de espaço requisitada pelos métodos de acesso, por meio da porcentagem média de utilização dos nós.

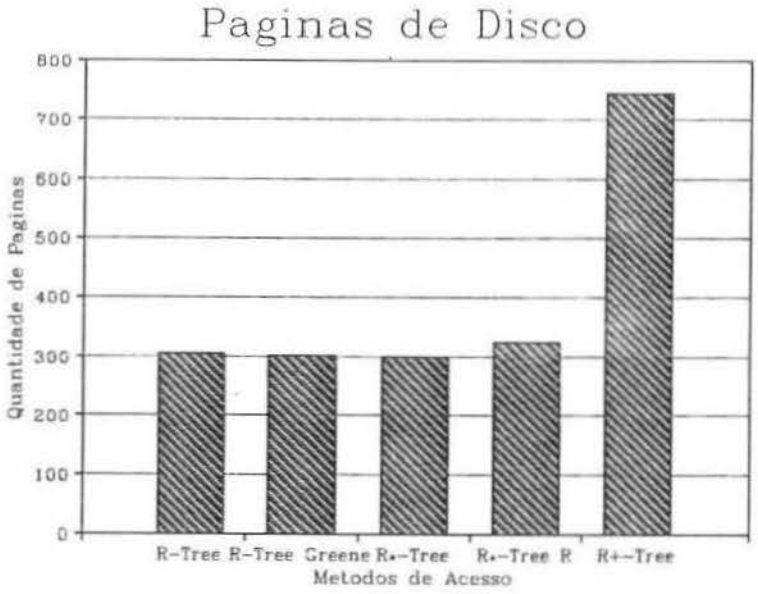


Figura 4.6: Relação da quantidade de páginas requerida, em média, para cada método de acesso.

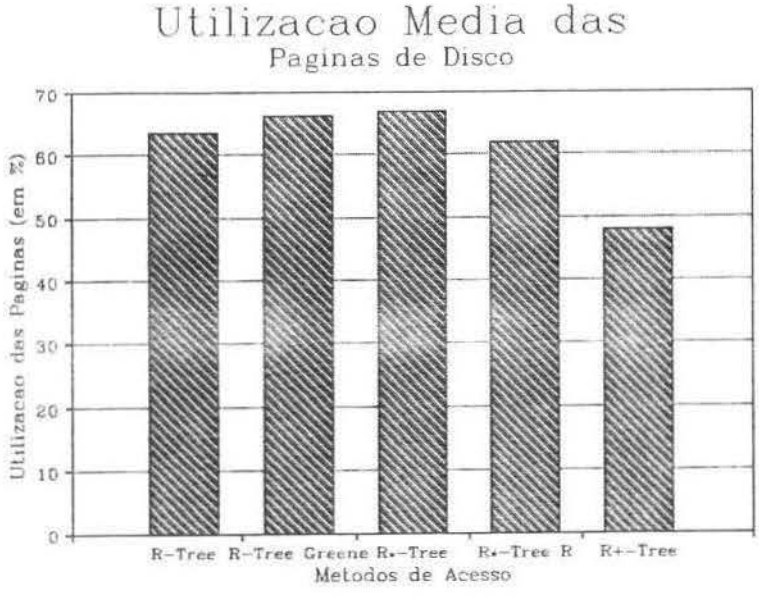


Figura 4.7: Relação da porcentagem de espaço utilizado por nó, em média, referente a cada método de acesso.

Capítulo 5

Conclusão

As pesquisas relacionadas ao estudo de métodos de acesso destinados a aplicações espaciais ainda se encontram em estágio inicial. Neste estudo foram identificados linhas de pesquisas diversas e contraditórias, como o estudo de Sellis [SRF87] e Kriegel [KSSS89], que descartaram totalmente a análise dos métodos de acesso derivados de árvores binárias, enquanto que Hsiao [HJ] tem desenvolvido pesquisas sobre essa classe, propondo, inclusive a sua utilização em aplicações espaciais; por outro lado, nos relatos de desempenho, referentes à comparação desses métodos de acesso, Greene [Gre89] e Sellis [SRF87] concluem a partir da análise da R-Tree e R^+-Tree, resultados opostos.

Conforme Guenther e Buchmann [GB90], aqui atribuiu-se essas divergências aos poucos estudos comparativos e à formas com que têm sido realizados. Como conseqüência, na prática, alguns SGBDs Espaciais existentes têm adotado estatégias pouco eficientes para determinação dos relacionamentos espaciais, pois as condições ideais de desempenho de cada método de acesso não são conhecidas.

Alguns autores, como Greene, defendem que essas condições ideais de desempenho não podem ser determinadas apenas por abordagens analíticas. Além de alguns métodos de acesso possuírem características que tornam difícil as suas representações em um modelo matemático, certas peculiaridades relacionadas com os SGBDs podem interferir no desempenho dos diversos métodos de acesso, de formas variadas, como utilização de *buffer pool*.

No *survey* apresentado no capítulo 2, chegou-se à conclusão que os métodos derivados de árvores *multiway trees* são os mais apropriados a serem utilizados nos SGBDs. Esse resultado foi obtido a partir das melhores relações de

desempenho, flexibilidade e simplicidade desses métodos quando comparado aos demais.

Entretanto, para determinação das condições ideais de desempenho dos métodos escolhidos, os experimentos, como realizados por Guttman [Gut84], Greene [Gre89] e Kriegel [KSSS89], não se mostraram satisfatórios. No levantamento do autor desta dissertação, sobre os métodos de acesso, foram encontrados alguns fatores relacionados aos dados e consultas, denominados determinantes de desempenho, que, quando atribuídos com valores diferentes, resultaram em alterações distintas de desempenho a cada método de acesso. Em virtude das estratégias adotadas pelos experimentos não cobrirem todos estes fatores, chegando-se, assim, a resultados apenas parciais, aqui foi proposta uma estratégia para a realização dos experimentos. Embora nem todos os possíveis valores dos fatores determinantes de desempenho possam ser identificados e, conseqüentemente, empregados nos experimentos, acredita-se que os resultados obtidos neste trabalho proporcionam uma análise mais completa que os demais. Foi identificado, por exemplo, que a R^+-Tree não é apropriada para distribuições não-uniformes. Tal conclusão, até então, não havia sido relatada em nenhum dos experimentos conhecidos.

Como resultado principal, além do extenso estudo comparativo dos métodos existentes, este trabalho apresenta a implementação de uma classe destes métodos. Discute algumas questões referentes a esta implementação, que por não terem sido apresentadas nos artigos originais, obrigaram o autor a tomar decisões particulares. Por fim, avalia os métodos implementados pela execução de uma série de experimentos, produzindo resultados bastante promissores.

Como continuação desta pesquisa, sugere-se:

- a avaliação do desempenho destes métodos sob Bases de Dados estáticas (caracterizadas pela ausência de atualizações). Embora os experimentos da fase 1 (inserção de elementos) atinjam esse objetivo, não foi analisado as estratégias propostas por alguns autores em proceder a inserção dos elementos conhecendo-se antecipadamente suas características espaciais (extensão e localização no espaço) [RL85].

- a análise da cadeia de referência, um resultado parcial dos experimentos, para estudos de técnicas de gerenciamento de *buffers* que melhor se adaptem as estruturas de acesso.

Bibliografia

[Ben75] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of ACM SIGMOD Conference on Management of Data*, 1990.

[BM72] R. Bayer and E. McCreight. Organization and Maintenance of Large Ordered Indexed. *Acta Informatica*, 1(3):173–189, 1972.

[BSW77] J. L. Bentley, D. F. Stanat, and E. H. Williams Jr. The Complexity od Finding Fixed-Radius Near Neighbors. *Information Processing Letters*, 6(6):209–212, December 1977.

[Bur83] W. A. Burkhard. Interpolation-Based Index Maintenace. In *Proceedings ACM SIGMOD-SIGACT Symposium*, 1983.

[C+86] M. J. Carey et al. Object and File Management in the EXODUS Extensible Database System. In *Proceedings of the 12th VLDB Conference*, pages 91–100, August 1986.

[CM90] F. S. Cox Jr. and G. C. Magalhães. Algumas Estruturas de Indexação para Banco de Dados Espaciais. In *Anais da III Semana de Informática da UFBa*. CPD - DCC (UFBa), Março 1990.

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of ACM*, 13(6):377–387, 1970.

[Com79] D. Comer. The Ubiquitous B-Tree. *ACM Computing Surveys*, 11(2):121–137, June 1979.

[EH88] W. Epstein and S. Hartman. An Implementation to the R+-Tree. Course Report of the Department of Computer Science at the University of Maryland, May 1988.

[FB74] R. A. Finkel and J. L. Bentley. Quad Trees a Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 4(1):1–9, 1974.

[Fre87] M. Freeston. The BANG File: A New Kind of Grid File. In *Proceedings of the 13th VLDB Conference*, pages 260–269, September 1987.

[FSR87] C. Faloutsos, T. Sellis, and N. Roussopoulos. Analysis of Object Oriented Spatial Access Methods. *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 426–439, May 1987.

[FSRM] C. Faloutsos, T. Sellis, N. Roussopoulos, and D. Metaxas. Access Methods for Spatial Objects: Algorithms and Analysis. Submitted to a Journal.

[GB90] O. Guenther and A. Buchmann. Research Issues in Spatial Databases. *ACM SIGMOD Record*, 19(4):61–68, December 1990.

[Gre89] D. Greene. An Implementation and Performance Analysis of Spatial Data Access Methods. In *Proceedings of Fifth IEEE International Conference on Data Engineering*, pages 606–615, February 1989.

[Gun88] O. Gunther. *Efficient Structures for Geometric Data Management*, volume 337 of *Lecture Notes in Computer Science*. Springer-Verlag, 1988.

[Gun89] O. Gunther. The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases. In *Proceedings of Fifth IEEE International Conference on Data Engineering*, pages 598–605, February 1989.

[Gut84] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 599–609, June 1984.

[GW87] O. Gunther and E. Wong. A Dual Space Representation for Geometric Data. In *Proceedings of the 13th VLDB Conference*, pages 501–506, September 1987.

[Hin85] K. Hinrichs. Implementation of the Grid File: Design Concepts and Experience. *BIT*, 25(4):569–592, 1985.

[HJ] P. Y. Hsiao and L. D. Jang. Spatial Data Access and Retrieval for Database Implementation. Submitted to the International Conference Extending Data Base Technology - 1992.

[HN83] K. Hinrichs and J. Nievergelt. The Grid File: A Data Structure Designed to Support Proximity Queries on Spatial Objects. In *Workshop on Graphtheorec Concepts in Computer Science*, June 1983.

[HSW88] A. Hutflesz, H.-W. Six, and P. Widmayer. Twin Grid Files: Space Otimizing Access Schemes. In *Proceedings of ACM SIGMOD Conference on Management of Data*, 1988.

[Ked82] G. Kedem. The Quad-CIF Tree: A Data Structure for Hierarchical On-Line Algorithms. In *Proceedings of the Nineteenth Design Automation Conference*, 1982.

[Knu73a] D. E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley Publishing Company, second edition, 1973.

[Knu73b] D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley Publishing Company, 1973.

[KS89] H. F. Korth and A. Silberschatz. *Sistemas de Banco de Dados*. McGraw-Hill, 1989.

[KSSS89] H. P. Kriegel, M. Schiwietz, R. Schneider, and B. Seeger. Performance Comparison os Point and Spatial Access Method. In *Lecture Notes in Computer Science - 409*. Springer-Verlag, 1989.

[KW87] A. Kemper and M. Wallrath. An Analysis of Geometric Modeling in Database Systems. *ACM Computing Surveys*. 19(1):47–91, March 1987.

[MO86] F. M. and J. A. Orenstein. Toward a General Spatial Data Model for an Object-Oriented DBMS. In *Proceedings of the 12th VLDB Conference*, pages 328–335, August 1986.

[Moo85] S. Moorehouse. ARC/INFO: A geo-Relational Model for Spatial Information. In *Proceedings of the ACM Seventh International Symposium on Computer Assisted Cartography*, 1985.

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database System*, 9(1):38–71, March 1984.

[OMSD87] B. C. Ooi, K. J. McDonell, and R. Sacks-Davis. Spatial kd-tree: An Indexing Mechanism for Spatial Databases. In *Proceedings of IEEE Comp. Software & Applications Conference*, pages 433–438, 1987.

[OO87] E. A. Ozkarahan and M. Ouksel. Dynamic and Order Preserving Data Partitioning for Database Machines. In *Proceedings of the 11th VLDB Conference*, August 1987.

[Ooi90] B. C. Ooi. *Efficient Query Processing in Geographic Information Systems*, volume 471 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[OSDM89] B. C. Ooi, R. Sacks-Davis, and K. J. McDonell. Extending a DBMS for Geographic Applications. In *Proceedings of Fifth IEEE International Conference on Data Engineering*, pages 590–597, February 1989.

[Ov82] M. H. Overmars and J. van Leeuwen. Dynamic Multi-Dimensional Data Structures Based on Quad and K-d Tree. *Acta Informatica*, 17(3):267–285, 1982.

[RL85] N. Roussopoulos and D. Leifker. Direct Spatial Search on Pictorial Databases Using Packed R-Trees. In *Proceedings of the ACM SIGMOD Conference*, pages 17–31, May 1985.

[Rob81] J. T. Robinson. The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. In *Proceedings of the ACM SIGMOD*, pages 10–18, 1981.

[RS87] L. A. Rowe and M. R. Stonebraker. The POSTGRES Data Model. In *Proceedings of the 13th VLDB Conference*, pages 83–96, September 1987.

[Sam80] H. Samet. Deletion in Two-Dimensional quad trees. *Comunications of the ACM*, 23(12):703–710, 1980.

[Sam84] H. Samet. The Quadtree and Related Hierarquical Data Structures. *ACM Computing Surveys*, 16(2):187–259, June 1984.

[Sam88] H. Samet. Hierarquical Representations of Collections of Small Rectangles. *ACM Computing Surveys*, 20(4):271–309. December 1988.

[Sam90a] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Publishing Company, 1990.

[Sam90b] H. Samet. Tutorial: Spatial data structures. V Simpósio Brasileiro de Banco de Dados, April 1990.

[SDDO87] R. Sackis-Davis, K. J. Donell, and B. C. Ooi. GEOQL - A Query Language for Geographic Information System. In *Australian and New Zealand Association for the Advancement of Scince Congress*, 1987.

[Sel] T. Sellis. The R^+-Tree. Communication through eletronic mail (1991).

[SK90] B. Seeger and H. P. Kriegel. The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base System. In *Proceedings of the 16th VLDB Conference*, pages 590–601, 1990.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the 13th VLDB Conference*, pages 507–518, September 1987.

[Wd89] L. Weyten and W. de Pauw. Quad List Quad Trees: A Geometrical Data Structure with Improved Performance for Large Region Queries. *IEEE Trans. on CAD/ICS*, 8(3):229–233, 1989.

[Wil82] D. E. Willard. Polygon Retrieval. *SIAM Journal on Computing*, 11(1):149–165, February 1982.