Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Algoritmos para Indexação em Bases de Dados Através de Estruturas Multidimensionais

Roberto Figueira Santos Filho
Agma Juci Machado Traina
Caetano Traina Júnior

Nº 85

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Abril/1999

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação e Estatística

Algoritmos para Indexação em Bases de Dados Através de Estruturas Multidimensionais[1]

Roberto Figueira Santos Filho
Agma Juci Machado Traina
Caetano Traina Júnior

Av. Dr. Carlos Botelho, 1465
CEP 13.560-970 – São Carlos – São Paulo – Brasil

ICMC-USP, São Carlos

Abril de 1999

[1] Este trabalho teve apoio financeiro da FAPESP.

RESUMO

A motivação para o desenvolvimento deste trabalho vem da necessidade atual de que os Sistemas de Gerenciamento de Bases de Dados possam armazenar e recuperar informações que não são apenas numéricas e textuais. Entre os tipos de dados "não tradicionais" que têm sido armazenados encontram-se, por exemplo, áudio digitalizado, partituras musicais, hipertextos, e imagens em geral. Este trabalho focaliza o estudo de estruturas de indexação para auxiliar o armazenamento e recuperação de imagens, imagens médicas em particular, dentro do contexto de um projeto maior, envolvendo a construção do núcleo de um Sistema de Armazenamento e Comunicação de Imagens (*Picture Archiving and Communication System*- PACS) para um Tomógrafo de Ressonância Magnética (TRM).

Para tanto, das imagens são extraídas informações que as caracterizam sob algum aspecto. Estas informações são representadas por vetores, conhecidos como vetores de características. Os vetores, assim formados, são tratados como pontos no espaço, cujas coordenadas correspondem às características extraídas das imagens. A esses pontos é dado o nome de **Dados Espaciais**. Este trabalho tem como objetivo introduzir os conceitos em torno dos métodos de indexação utilizados para indexar dados espaciais, conhecidos como **Métodos de Indexação Espaciais**. São apresentados, ainda, alguns exemplos desses métodos de indexação

Sumário

Lista de Figuras

Abreviaturas

BD	Base de Dados
SIRIUS/GO	Gerenciador de Objetos para o modelo de dados SIRIUS
MBR	*Minimum Bounding Rectangle* (mínimo retângulo externo a um objeto espacial)
MIE	Método de Indexação Espacial
MRO	Modelo de Representação de Objetos
MRO/GEO	Gerenciador de Objetos para o Modelo de Representação de Objetos
PACS	*Picture Archiving and Communication System* (Sistema de Armazenamento e Comunicação de Imagens)
PAM	*Point Access Methods* (Métodos de Acesso Pontual)
RM	Ressonância Magnética
SAM	*Spatial Access Methods* (Métodos de Acesso Espacial)
SGBD	Sistema de Gerenciamento de Base de Dados
TAD	Tipo Abstrato de Dados
TRM	Tomógrafo de Ressonância Magnética

1. INTRODUÇÃO

INTRODUÇÃO

1.1 Considerações Iniciais

Sistemas de Gerenciamento de Bases de Dados (SGBD) têm sido cada vez mais utilizados para o armazenamento e recuperação de dados que não são apenas numéricos ou textuais. Entre os tipos de dados "não tradicionais" que têm sido armazenados encontram-se, por exemplo: áudio digitalizado, partituras musicais [BekerM_97], hipertextos e imagens em geral [SantosR_97] [SchneiderM_97]. Este trabalho focaliza o armazenamento de imagens, imagens médicas em particular, dentro do contexto de um projeto maior, envolvendo a construção do núcleo de um Sistema de Armazenamento e Comunicação de Imagens (*Picture Archiving and Communication System* - PACS [DuttonG_90] [AlcocerP_96] [SclabassiR_96]) para um Tomógrafo de Ressonância Magnética (TRM). Porém, pretende-se que os resultados deste trabalho não se restrinjam ao armazenamento de imagens.

Os exames médicos, atualmente, podem gerar uma grande quantidade de dados em formato digital provenientes dos diversos instrumentos médicos computadorizados existentes [DuttonG_90] (TRM, radiografias, ultra-sonografia, eletrocardiograma, etc.), sendo que parte destes instrumentos é voltada à geração de imagens gráficas. O armazenamento e recuperação destas imagens em SGBDs tradicionais apresenta algumas dificuldades que se resumem em não ser imediata a definição de um critério que permita estruturar uma coleção das mesmas [GudivadaV_95]. Assim, a recuperação tem sido através da sua associação a dados numéricos ou textuais, sobre os quais as buscas são efetuadas [CardenasA_93] [FlicknerM_95]. Tais dados são usualmente obtidos pela descrição de seu conteúdo (por exemplo, um diagnóstico feito por um médico, no caso de uma imagem já analisada e processada pelo radiologista), ou por outros dados externos a elas relacionados (por exemplo, os parâmetros utilizados no equipamento de coleta das imagens) [BrownL_92].

A recuperação de imagens calcada em informações extraídas do seu conteúdo tem sido muito estudada [TamuraH_84] [WhiteD_95] [PetrakisE_97] e está fundamentada na extração de características das imagens (*features*), as quais podem ser indexadas. Estes índices são usados

para restringir drasticamente a coleção de imagens que fazem parte do conjunto de dados resultantes de uma consulta, de maneira que apenas algumas (relativamente) poucas imagens precisam ser, de fato, comparadas para obter o resultado de uma consulta firmada no conteúdo pictórico das imagens. Cada índice corresponde a uma determinada característica que tende a ser independente das demais. Porém, cada consulta vincula um determinado conjunto de características, estabelecendo um inter-relacionamento entre elas.

O grupo de Bases de Dados do Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo (ICMC-USP), vem desenvolvendo um “Sistema de Gerenciamento de Base de Dados Orientada a Objetos” (SIRIUS/GO), sendo este uma evolução do “Modelo de Representação de Objetos” (GEO/MRO) proposto originalmente em 1986 por [TrainaC_86], a qual gerou o modelo de dados “SIRIUS” [BiajizM_96]. Dentre as novas características deste modelo se encontra a representação de imagens como um tipo abstrato de dados (TAD) [SantosR_97], fruto de um projeto maior entre o já referido Grupo, o Grupo de Computação Gráfica e Processamento de Imagens do ICMC-USP e o Grupo de Ressonância Magnética do Instituto de Física de São Carlos (IFSC-USP) [SantosR_96] [SenzakoE_96] [TrainaC_97]. O TAD imagem é definido de maneira a permitir a indexação e recuperação de imagens baseadas em seu conteúdo.

As imagens no SIRIUS/GO são definidas no seu meta-modelo como uma especialização do conceito de **Atributo** associado a Objetos. Este meta-modelo considera que cada atributo tem uma **Característica**, a qual define o conjunto de operações em que está envolvido. Por exemplo, número é uma característica de atributo que define as seguintes operações: soma, multiplicação, etc.. Cadeia de caracteres, que é outra característica de atributo, engloba as seguintes operações: busca, concatenação, extração, etc.. Deve ser observado que o conjunto de operações definidas por uma certa característica é independente do tipo de dados usado para representar, em meio computacional, o atributo a que se refere. Sendo assim, não há diferença se um número está sendo representado como inteiro, real ou dupla precisão.

Característica é uma especialização do conceito de atributo de objetos, o qual, por sua vez, é uma especialização do conceito de TAD. Portanto, cada tipo de dados pode ter seu conjunto

específico de propriedades. Por exemplo, números complexos podem ter um método para calcular sua parte real.

Como foi mencionado anteriormente, imagem passa a ser tratada como uma característica de atributo, permitindo que diversos formatos de representação e diversos algoritmos de processamento de imagens sejam associados aos objetos que a possuem.

Para completar a definição de imagens neste meta-modelo, são necessários três conceitos [SantosR_96]:

1. **Constante-Imagem**: são imagens utilizadas como base para a busca de outras imagens que apresentem algum grau de similaridade, segundo alguma propriedade de interesse. Embora uma imagem seja um objeto representado por um grande volume de dados, não deixa de ser um valor para um atributo com característica de imagem, da mesma forma que um valor numérico é um valor para um atributo com característica de número. Em outras palavras, constantes imagem são imagens pré-definidas diretamente relacionadas com o domínio da aplicação. Por exemplo: uma imagem referente a uma determinada patologia de um certo órgão do corpo humano, obtida através de um TRM, caracteriza esta patologia de forma que pode ser colocada como uma imagem exemplo da mesma. Esta imagem poderia, então, ser utilizada para identificar outras imagens com a mesma informação. Quando uma imagem é tratada desta forma, é classificada como Constante-Imagem, passando a ser utilizada pelo sistema como padrão para comparação.
2. **Sumarizadores de Imagens**: um sumarizador de imagem é um procedimento computacional que opera sobre as imagens, extraindo dados úteis que são chamados “Parâmetros-da-Imagem“.
3. **Parâmetros-da-Imagem:** podem ser valores numéricos; uma coordenada da imagem ou de algum objeto contido na mesma; um conjunto de valores; etc.. Cada sumarizador pode retornar um ou mais Parâmetros-da-Imagem, sendo que, cada um representa alguma informação útil para o processo de seleção e recuperação das imagens.

Fundamentado nestes conceitos, o armazenamento das imagens pode ser feito em associação com objetos na base de dados (BD) onde são tratadas como características de atributos de objetos de um determinado tipo de dados. Para tanto, são agrupadas de acordo com os domínios de sua aplicação, os quais são identificados pela definição de conjuntos de constantes-imagem e sumarizadores significativos a esses domínios. Ou seja, quando uma imagem vai ser armazenada é submetida a um conjunto de pares <sumarizador, constante-imagem> significativos da aplicação da qual faz parte, e os parâmetros-da-imagem resultantes são usados para criar estruturas de índices, representando uma forma de "ordenação" entre o conjunto das imagens armazenadas e suas constantes-imagem, dentro de um mesmo domínio.

Quando uma aplicação é construída, o conjunto de Constantes-Imagem relacionadas a cada sumarizador é definido, e o tipo sumarizador de atributo de objeto precisa ser calculado. Com isso, na medida em que as imagens são armazenadas na BD como um valor de um atributo, calculam-se apenas os pares <sumarizador, constante-imagem> relativos ao domínio em que a imagem se aplica. Cada um dos parâmetros resultantes é, então, usado para a criação de estruturas de índices, sendo uma para cada sumarizador de imagem relativo a este atributo de objeto.

Os parâmetros resultantes são apresentados na forma de vetores (também chamados vetores de características - *feature vectors*), sob os quais são montadas as estruturas de índices, representando a imagem em um espaço de dimensão *d*, onde *d* corresponde ao número de elementos do respectivo vetor (parâmetro). Cada imagem armazenada pode pertencer a várias estruturas de índices simultaneamente, uma para cada parâmetro de cada sumarizador do tipo de atributo de objeto que tem esta imagem como valor.

1.2 Objetivos

Estruturas de indexação são muito utilizadas para tornar mais rápidas as respostas a consultas feitas em uma BD [ElmasriR_94]. Tendo em vista a diversidade entre os tipos de dados, existem inúmeras estruturas já elaboradas [SametH_90] [ElmasriR_94], cada uma com características próprias.

Algumas destas características são: tipo de dados a ser indexado, formas de acesso a esses dados, tamanho do conjunto de dados, possibilidade ou não de inclusão/exclusão ao longo do tempo (BD estática **X** BD dinâmica), necessidade de manter parte da estrutura de índices em disco (caso o conjunto de dados seja muito grande para ser totalmente mantido em memória primária), etc..

Como visto na seção 1.1, o parâmetro retornado por um sumarizador é representado na forma de um vetor proveniente de um preprocessamento efetuado sobre a imagem. Os elementos deste vetor podem, então, ser vistos como coordenadas de um ponto situado no espaço, cuja dimensão é determinada pela sua quantidade. Este número varia de acordo com o sumarizador, modificando-se, conseqüentemente, a dimensão do espaço representado. Observa-se que existe um alto grau de variabilidade na quantidade de elementos que um parâmetro pode ter, ou seja, o tamanho de um vetor resultante de um sumarizador pode atingir valores maiores que 300 [SantosR_96].

Os pontos assim gerados são armazenados em estruturas de indexação apropriadas para o armazenamento e recuperação de dados multidimensionais. Estas estruturas tornam possível a busca de imagens com certo grau de semelhança, através da busca de pontos (os quais representam vetores de características) que estejam próximos no espaço. Com isso, uma consulta típica seria: a busca de imagens com certo grau (80% a 100%) de semelhança (definida por: posição relativa dos órgão apresentados, ou possibilidade de existência de determinada patologia, ou histograma de cores, etc.) com uma imagem exemplo [SantosR_96].

Este trabalho introduz os conceitos relativos a estes tipos de dados (dados que representam objetos no espaço) e os métodos de indexação utilizados para auxiliar no seu armazenamento e recuperação. Alguns dos métodos de indexação encontrados na literatura serão, aqui, introduzidos.

1.3 Organização do Trabalho

Este trabalho apresenta a seguinte organização:

1. **capítulo 1**: são apresentadas as idéias que motivaram o desenvolvimento do trabalho e sua organização;
2. **capítulo 2**: são apresentados conceitos gerais sobre estruturas de indexação voltadas para dados multidimensionais e como esses podem ser representados em meio computacional. Em seguida, são resumidas as consultas e operações comuns a este tipo de dados, finalizando com uma coletânea de aplicações e trabalhos relacionados;
3. **capítulo 3**: são apresentadas e discutidas, a título de exemplificação, algumas das estruturas estudadas;
4. **capítulo 4**: são apresentadas as conclusões finais;
5. **Referências Bibliográficas**.

2. DADOS ESPACIAIS

2. DADOS ESPACIAIS

2.1 Introdução

Os SGBDs convencionais têm sido desenvolvidos e aplicados, predominantemente, para a organização de grande volume de dados relacionados a aplicações financeiras e administrativas, também chamadas "tradicionais" (*standard*). Porém, as recentes pesquisas na área de gerenciamento de dados têm se voltado cada vez mais para aplicações de âmbito técnico-científicas, também citadas como "não tradicionais" (*non-standard*).

Esta nova abordagem trouxe a necessidade de se adaptar os modelos de dados e os SGBDs convencionais às novas exigências impostas por aplicações não tradicionais, tais como: projeto auxiliado por computador (*CAD - Computer Aided Design*), processamento de imagens, sistemas multimídia, sistemas de informações geográficas (*GIS - Geographic Information Systems*), projeto de circuitos integrados auxiliado por computador e outras nas áreas médica e biológica [SchneiderM_97].

Dentre estas diversas exigências feitas aos SGBDs, uma que tem se tornado cada vez mais necessária devido à grande variedade de aplicações, é a capacidade de gerenciar **dados geométricos** ou **espaciais**[2] (também citados como dados multidimensionais), ou seja, relacionados com o espaço geométrico (ou simplesmente espaço) onde são representados [SchneiderM_97]. Estes dados consistem de objetos espaciais compostos por pontos, linhas, regiões no espaço, retângulos, superfícies, volumes, hipercubos e outros de dimensões maiores que envolvem, inclusive, a noção de tempo [ArefW_97].

[2] O termo "espaço" tem sido muito usado para referenciar o espaço em terceira dimensão. Daqui para frente, os termos relacionados ao espaço de representação dos dados ("espaço", "espacial", outros) não limitam a ordem de sua dimensão (a menos que isso seja colocado de forma explícita), fazendo referência a espaços *n*-dimensionais, sendo "*n*" um número natural diferente de zero. Será considerado, também, como definido o significado da expressão "*n*-dimensional", para indicar uma dimensão qualquer (incógnita) diferente de zero.

Exemplos de dados espaciais (ou objetos espaciais) incluem a representação para: rios, estradas, municípios, estados, acidentes geográficos, circuitos eletrônicos, tumores, transações financeiras, seqüências encadeadas como DNA e caracteres alfanuméricos, etc.. Exemplos de propriedades relativas a estes dados incluem: extensão e localização geográfica de um rio, limites de um município, variações financeiras ao longo de um intervalo de tempo, histograma de cores de uma tomografia, etc. [ArefW_97] [SametH_95] [LinK_94] [Berchtold_96]. São exemplos de consultas feitas em BDs espaciais: "obter as ultra-sonografias nas quais os valores dos índices do histograma de cores variam em torno de 15% com relação aos respectivos índices da ultra-sonografia de fulano...", "indicar as ruas de uma cidade que são cortadas por determinada linha do metrô...", etc..

Gaede e Günther em [GaedeV_97] relacionam algumas propriedades de dados espaciais que permitem um melhor entendimento dos requisitos de uma BD espacial:

1. **Complexidade estrutural**: dados espaciais podem ser compostos por um simples ponto ou por diversos milhares de polígonos distribuídos[3] arbitrariamente no espaço. Geralmente, não é possível armazenar coleções de objetos deste tipo em uma tupla de tamanho fixo, referente a uma simples tabela de BDs relacionais;
2. **Dinamismo**: dados espaciais são, geralmente, dinâmicos. Inserções, remoções e alterações são comuns ao longo do tempo e, conseqüentemente, as estruturas de dados utilizadas para suportar dados deste tipo têm que ser projetadas prevendo este dinamismo, para evitar uma deterioração;
3. **Quantidade**: BDs espaciais tendem a ser muito grandes (possuir grande volume de informações), o que torna necessária uma perfeita integração entre o armazenamento de dados em memória primária (ou, simplesmente, armazenamento primário) e o feito em memória secundária (ou armazenamento secundário);
4. **Inexistência de uma álgebra padronizada**: apesar da existência de diversas propostas, ainda não existe uma álgebra padronizada para dados espaciais. Ou seja, não existe um conjunto padrão de operadores básicos próprios de dados espaciais. O

[3] Considere-se "distribuição dos dados" como sendo a disposição dos dados ao longo do espaço. Esta distribuição pode ser regular (os dados se encontram espalhados de forma a terem mesma distância entre si), ou irregular (os dados são aglomerados em diversas porções do espaço).

conjunto de operadores (que serão tratados por operadores/operações espaciais) é estritamente dependente do domínio da aplicação em uso, embora alguns operadores (como interseção) sejam mais comuns que outros;

5. **Muitos operadores espaciais são *não fechados***: a intercessão de dois polígonos, por exemplo, pode resultar em um número qualquer de pontos, arestas, ou polígonos disjuntos. Esta propriedade é particularmente importante quando esses operadores são aplicados consecutivamente;
6. **Custo computacional**: embora o custo computacional varie entre os operadores espaciais, geralmente estes operadores são mais "caros" que os relacionais.

Após esta breve apresentação sobre dados espaciais, abrangendo seu significado, sua importância e sua complexidade, serão abordados alguns conceitos em torno de como são representados em meio computacional e algumas formas comuns de consulta. Em seguida, será apresentada uma rápida coletânea de trabalhos e exemplos de aplicação relacionados a este tipo de dados.

2.2 Representação de Dados Espaciais

A forma elementar de representar objetos espaciais em um meio computacional é tratá-los de forma explícita, ou seja, simplesmente definir seus atributos e relacionamentos com outros objetos e armazená-los em algum SGBD convencional.

Por exemplo, em uma base contendo dados sobre estradas de rodagem de um estado, uma opção seria atribuir, explicitamente, os campos: nome, extensão, tipo de capeamento (asfalto, barro, brita, etc.) e outros. E, quanto à sua posição no espaço, simplesmente acrescentar registros (possivelmente vários) indicando, por exemplo, as coordenadas (longitude e latitude) de início e fim de cada trecho (que podem ser limitados pelas cidades atravessadas pela estrada).

Esta visão pode ser facilmente implementada em BDs convencionais e é muito prática quando as consultas são simples e podem ser previstas com antecedência. Porém, há motivos que a tornam indesejável, como, por exemplo, os dois a seguir:

1. as consultas teriam que ser conhecidas a priori para que fossem feitos ajustes na representação dos dados de forma a suportá-las. Supondo a base definida com os atributos acima, uma consulta interessante seria: dada uma estrada, encontrar as estradas que a cruzam. Ora, a não ser que o cruzamento ocorra em uma cidade, a base como foi definida acima não pode responder a esta pergunta de forma rápida. Uma solução seria armazenar, adicionalmente, os cruzamentos existentes. Porém,
2. esta solução levaria o usuário a cadastrar uma quantidade de dados imprevisível, levando a uma quantidade de informações absurdamente grande e desnecessária (supondo a existência de outra forma de representação), acarretando um alto custo de processamento para a recuperação dos dados armazenados.

Em outras palavras, o problema é que as consultas feitas em BDs espaciais (ou simplesmente consultas espaciais) tendem a focalizar a localização dos objetos no espaço ou, dependendo do tipo de objeto (linhas, retângulos, cubos, etc.), sua extensão (comprimento, área, volume).

A habilidade em responder a consultas espaciais de maneira flexível está diretamente relacionada à forma como estes dados estão representados nas BDs. Além disto, para se obter um desempenho razoável, os dados devem estar ordenados de alguma forma [SametH_95].

A dificuldade em representar e ordenar os dados espaciais está em que "não existe uma ordenação total entre objetos espaciais que preserve a proximidade[4] existente no espaço" [SchneiderM_97]. Ou seja, não há como fazer um mapeamento de um espaço *n*-dimensional (com *n* maior que um) em um espaço unidimensional de forma que dois objetos quaisquer tidos como próximos no espaço original estejam próximos entre si na seqüência gerada pela ordenação.

A ordenação de dados espaciais deve, então, ser fundamentada em todas as informações que os mesmos apresentam inerentes ao espaço. Ou seja, a ordenação é feita sob a porção do espaço ocupada pelo dado. Técnicas que utilizam esta forma de ordenação são conhecidas como **métodos de indexação espacial** ou **métodos de indexação multidimensional** (por questões de

[4] As considerações feitas em todo este trabalho supõem o espaço *Euclideano* como base de representação, e a noção de distância (e proximidade) é relativa à distância *Euclideana*.

padronização e simplificação, será usado neste trabalho a abreviação da primeira denominação, ou seja, método de indexação espacial - **MIE**).

Uma forma bastante usual de implementar busca em BDs espaciais usando sistemas convencionais corresponde à utilização de estruturas de indexação unidimensionais (B-Tree [ComerD_79] e suas variantes) repetidamente, uma para cada dimensão. Este método tem se mostrado insatisfatório, pois não é possível decidir com total segurança, a partir de uma dimensão, o que pode ou não ser excluído (do conjunto resposta) sem antes ter consultado os dados relativos às outras dimensões (não necessariamente todas). Em outras palavras, o conjunto de dados a ser verificado durante uma consulta não pode ser reduzido gradualmente de forma satisfatória.

Gaede e Günther em [GaedeV_97] relacionam alguns requisitos básicos que os MIEs devem possuir em função das propriedades apresentadas pelos dados espaciais (ver seção 2.1) e suas aplicações (considerando o trabalho em estudo, será dada uma ênfase maior aos requisitos 1, 2, 4, 7 e 8 abaixo):

1. **Dinamismo**: os dados são inseridos e removidos da BD de forma aleatória. Os MIEs devem tratar estas mudanças continuamente;
2. **Permitir e gerenciar o armazenamento secundário**: tendo em vista o grande volume de dados, os MIEs devem prover formas de armazenamento secundário;
3. **Suporte a grande número de operações espaciais**;
4. **Independência quanto à seqüência de inserção dos dados e quanto à sua distribuição no espaço**: o desempenho dos MIEs não pode ser debilitado se a ordem de inserção for modificada, ou se os dados não estiverem uniformemente distribuídos no espaço;
5. **Simplicidade**: MIEs intricados mostram-se, em geral, passíveis de erro de implementação e, desta forma, não são suficientemente confiáveis para aplicações em larga escala;
6. **Escalabilidade**: os MIEs devem se adaptar facilmente ao crescimento da BD;

7. **Eficiência quanto ao tempo** (ou, eficiência temporal): consultas espaciais devem ser rápidas. O principal objetivo é alcançar as características de desempenho proporcionadas por uma B-Tree [ComerD_79]: os MIEs devem garantir um desempenho com complexidade de ordem logarítmica em relação a operações de busca, independentemente da distribuição dos dados e da ordem de inserção. Este desempenho deve ser mantido para qualquer combinação dos parâmetros relativos à sua dimensionalidade (ordem da dimensão espacial);
8. **Eficiência quanto ao número de *bytes* necessários para armazenamento** (ou, eficiência de armazenamento): um índice deve ser armazenado de forma a gastar uma quantidade de *bytes* menor do que a gasta com os atributos dos dados a serem indexados. Em muitos MIEs o tamanho do índice é diretamente proporcional (em alguns casos até em ordem exponencial) à dimensão do espaço [LinK_94];
9. **Concorrência e confiabilidade**: os SGBDs atuais permitem que os dados sejam acessados, inseridos e alterados por vários usuários ao mesmo tempo (de forma concorrente). Os MIEs devem estar preparados para lidar com o acesso concorrente aos dados sem implicar em grande perda de desempenho. Além disso, devem fornecer recursos para recuperar índices danificados por eventuais falhas no sistema (confiabilidade);
10. **Impacto mínimo**: por fim, a integração de um MIE a um sistema de gerenciamento de banco de dados deve gerar o menor impacto possível nas partes já existentes do mesmo.

Em acréscimo, Samet em [SametH_90] comenta que é comum a ocorrência de colisão em algumas aplicações envolvendo dados espaciais, ou seja, dois objetos espaciais podem ter a mesma coordenada no espaço. Por exemplo: algumas cidades possuem elevados em algumas de suas ruas (ruas elevadas ao longo de outras ruas como se fossem pontes, porém percorrendo o mesmo caminho). Uma BD espacial que considere apenas os aspectos relativos à localização bidimensional das ruas (coordenadas - latitude e longitude - de início e fim de seus segmentos) teria que permitir que dois objetos ocupassem o mesmo ponto no espaço.

Conseqüentemente, uma análise detalhada da distribuição espacial dos dados deve ser feita para verificar a possibilidade da existência de colisões e tomar as devidas providências com relação ao MIE a ser implementado. Samet propõe, ainda, que os registros de índices tenham um ponteiro a mais, direcionado para uma **lista de colisões**. Esta proposta, apesar de implicar um campo extra para uma situação que pode ser considerada rara, é justificada pela possibilidade de tornar necessários procedimentos de inclusão mais complicados, caso não seja implementada.

Porém, White e Jain em [WhiteD_95] colocam que em algumas aplicações (principalmente as voltadas para informações visuais que indexam vetores de características - *feature vectors*), em que o domínio dos dados em cada dimensão corresponde ao dos números reais, as chances de ocorrerem colisões são muito raras e diminuem à medida que a ordem da dimensão aumenta. Logo, cabe ao projetista do MIE decidir como abordar este problema.

Outra sugestão interessante apresentada por Samet, agora em [SametH_95], diz respeito à separação estrutural dos dados espaciais em dois subconjuntos: um subconjunto de **atributos espaciais,** relativos às características espaciais do objeto (coordenadas, por exemplo), e um subconjunto de **atributos não-espaciais**, relativos às demais características (nomes - o nome de uma rua por exemplo), mantendo uma ligação entre os dois subconjuntos (possivelmente por ponteiros). Com este recurso, procura-se obter um desempenho melhor nas consultas espaciais devido a ter-se, num certo momento, uma quantidade maior de chaves de indexação (atributos espaciais dos objetos) na memória principal, visto que a porção desta memória que era ocupada pelo segundo subconjunto de atributos está agora ocupada por outras chaves.

Para finalizar, existe ainda uma subdivisão dos MIEs, citada em [KriegelH_90] [GaedeV_97] [SellisT_97], que será utilizada nas demais seções:

1. **Métodos de acesso pontual** (ou "*point access methods*" - *PAM*): métodos desenvolvidos para acessar dados que correspondem a pontos no espaço, conhecidos por **dados pontuais**. Em outras palavras, dados situados em um espaço *n*-dimensional sem nenhuma extensão (dados espaciais gerados a partir de vetores de características são um exemplo de dados pontuais);

2. **Métodos de acesso espacial** (ou "*spatial access methods*" - *SAM*[5]): métodos desenvolvidos para acessar dados que correspondem a **objetos complexos** (que precisam de vários pontos distintos para serem representados), conhecidos por **dados espaciais**, tais como linhas, polígonos, regiões do espaço, cubos, hipercubos, etc..

2.3 Consultas Espaciais

Como foi dito anteriormente, não existe um conjunto básico padrão de operadores espaciais. Porém, diversos autores têm sugerido possíveis conjuntos padrão [FriedmanJ_77] [OrensteinJ_86] [KriegelH_92] [RoussopoulusN_95] [WhiteD_95]. Em especial, Gaede e Günther em [GaedeV_97] relacionam um conjunto bastante abrangente (com relação aos exemplos encontrados na literatura) de operadores espaciais, que será apresentado adiante com suas respectivas definições formais.

Para ressaltar a importância das operações espaciais, pode-se citar o seguinte comentário feito por Günther e Buchmann em [GuentherO_90]: "quando da escolha de uma estrutura de dados, é importante levar em consideração as operações que precisam ser suportadas, pois a eficiência das mesmas varia de acordo com a representação dos dados em que são aplicadas".

2.3.1 Definições preliminares

Os exemplos a seguir supõem o espaço *Euclideano* como base de representação dos objetos espaciais, o qual será tratado simplesmente por $\boldsymbol{E^d}$, em que *d* indica a ordem da dimensão do mesmo. Os objetos armazenados em uma BD espacial possuem apenas uma localização no espaço, expressa por suas *d* coordenadas. No entanto, vale lembrar que uma mesma localização espacial pode ser "ocupada" por vários objetos armazenados na BD.

A "porção" do espaço "ocupada" por um objeto, *o.G*, é descrita através dos seus atributos espaciais. Esta "porção" corresponde à geometria, formato ou **extensão espacial** do objeto. É

[5] Para manter uma correlação com o termo original em inglês, neste texto serão utilizadas as siglas originais, *PAM* e *SAM*, apenas mudando a fonte para itálico.

válido observar que objetos espaciais podem conter "buracos", a literatura apresenta uma analogia desta propriedade a um queijo suíço.

Diversos MIEs fazem uma aproximação do formato do objeto para um formato mais simples, como um retângulo ou uma esfera, por facilitar seu manuseio. Esta aproximação é feita considerando o retângulo mínimo ou a esfera mínima externa ao objeto, que enclausura toda a sua extensão espacial, de forma a obter a menor área ou volume possível. Este trabalho irá tratar apenas da aproximação para retângulos, por ser esta a mais comum encontrada nos MIEs.

Considerando o menor intervalo $I_i(o) = [a_i, b_i]$ ($a_i, b_i \in E^1$) que limita e descreve a extensão do objeto *o* ao longo da dimensão *i* ($0<i<n$) em um espaço *n*-dimensional, o retângulo mínimo limítrofe a este objeto é definido por $I^d(o) = I_1(o)$ X $I_2(o)$ X ... X $I_d(o)$. O termo utilizado na literatura inglesa para designar este retângulo é *minimum bounding rectangle* (***MBR***) ou *minimum bounding box* (*MBB*), sendo que a sigla *MBR*, do termo em inglês, será utilizada no decorrer deste trabalho por já ser de uso bastante difundido.

Com a utilização do *MBR* para representar os dados, uma consulta espacial pode ser feita em duas etapas (ver Figura 1), visando um melhor desempenho [PatelJ_96]:

1. **Filtragem**: nesta etapa, a consulta é feita sobre os *MBR*s dos objetos para gerar um conjunto de objetos candidatos, ou **conjunto candidato**, de tamanho reduzido. Porém, esta abordagem é passível de erro por não ter sido utilizado o formato real dos objetos.
2. **Refinamento**: nesta etapa, o formato real do objeto é utilizado para gerar o conjunto final de resposta, ou **conjunto resultado**. Esta análise pode ser computacionalmente muito custosa a depender do operador usado na consulta, da complexidade dos objetos e da dimensão do espaço, justificando a etapa de filtragem.

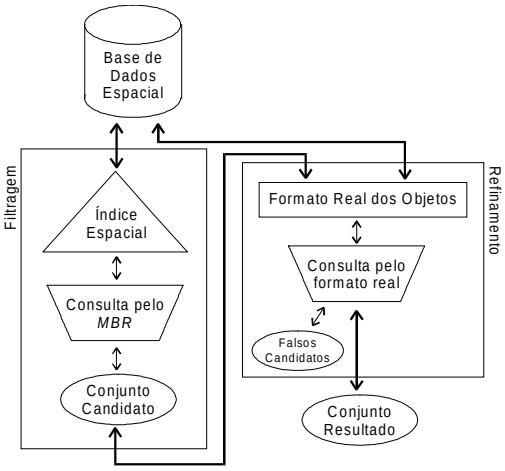

Figura 1: Etapas de uma consulta espacial.

2.3.2 Operadores espaciais

Os nomes das operações serão apresentados em inglês para manter o padrão encontrado na literatura. As figuras (da Figura 2 até a Figura 7) que ilustram estas operações foram obtidas em [GaedeV_97].

2.3.2.1 Exact Match Query/EMQ ou *Object Query*

Considere-se um objeto o' com extensão espacial $o'.G \subseteq E^d$. Esta operação resulta em um subconjunto com todos os objetos o que ocupam a mesma extensão espacial que o'.

$$EQM(o') = \{o \mid o'.G = o.G\}$$

2.3.2.2 Point Query/PQ

Considere-se um ponto $p \in E^d$. Esta operação resulta em um subconjunto com todos os objetos o que contém p (Figura 2).

$$PQ(p) = \{o \mid (p \cap o.G) = p\}$$

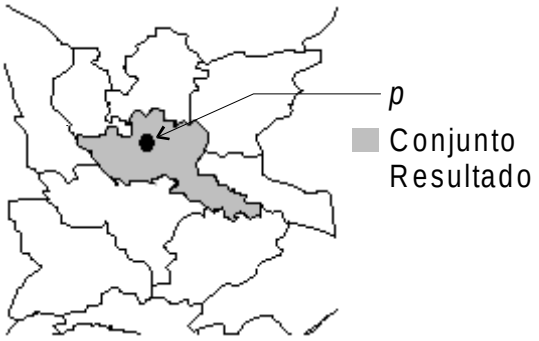

Figura 2: Exemplo de uma operação ***Point Query***.

Esta operação pode ser considerada como um caso especial das operações: *Window Query*, *Intersection Query*, ou *Enclosure Query*, descritas a seguir

2.3.2.3 Window Query/WQ ou *Range Query*

Considere-se o intervalo d-dimensional $I^d = [a_1, b_1] \text{ X } [a_2, b_2] \text{ X } ... \text{ X } [a_d, b_d]$. Esta operação resulta em um subconjunto com todos os objetos o que fazem interseção com pelo menos um ponto de I^d (Figura 3).

$$WQ(I^d) = \{o \mid (I^d \cap o.G) \neq \varnothing\}$$

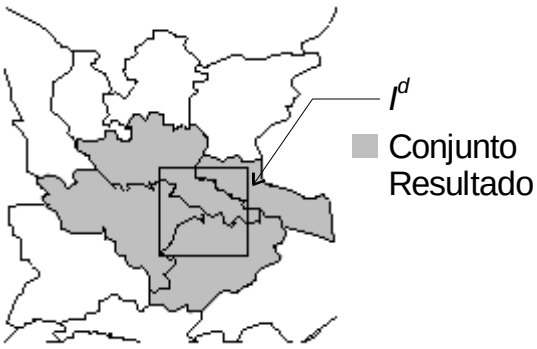

Figura 3: Exemplo de uma operação ***Window Query***.

Esta operação força que o intervalo seja limitado por retas paralelas aos eixos de coordenadas espaciais.

2.3.2.4 Intersection Query/IQ, ou *Region Query*, ou *Overlap Query*

Considere-se um objeto o' com extensão espacial $o'.G \subseteq E^d$. Esta operação resulta em um subconjunto com todos os objetos o que fazem interseção com pelo menos um ponto de o' (Figura 4).

$$IQ(o') = \{o \mid (o'.G \cap o.G) \neq \emptyset\}$$

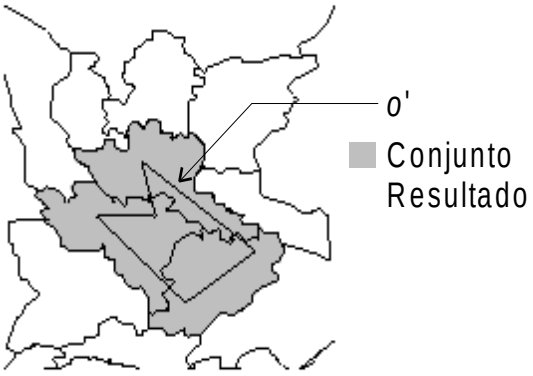

Figura 4: Exemplo de uma operação ***Intersection Query***.

Esta operação é uma generalização da operação *Window Query* de forma a permitir que o intervalo, ou melhor, que a região espacial, tenha formato e orientação quaisquer.

2.3.2.5 Enclosure Query/EQ

Considere-se um objeto o' com extensão espacial $o'.G \subseteq E^d$. Esta operação resulta em um subconjunto com todos os objetos o que englobam o' (Figura 5).

$$EQ(o') = \{o \mid (o'.G \cap o.G) = o'.G\}$$

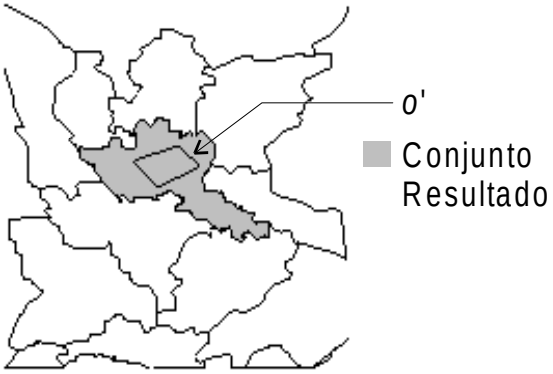

Figura 5: Exemplo de uma operação ***Enclosure Query***.

2.3.2.6 Containment Query/CQ

Considere-se um objeto o' com extensão espacial $o'.G \subseteq E^d$, esta operação resulta em um subconjunto com todos os objetos o que são englobados por o' (Figura 6).

$$CQ(o') = \{o \mid (o'.G \cap o.G) = o.G\}$$

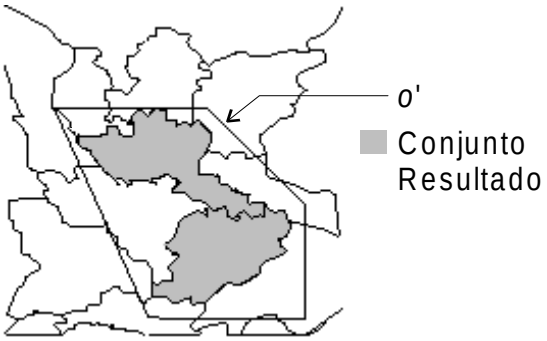

Figura 6: Exemplo de uma operação ***Containment Query***.

2.3.2.7 Adjacency Query/AQ

Considere-se um objeto o' com extensão espacial $o'.G \subseteq E^d$, esta operação resulta em um subconjunto com todos os objetos o adjacentes a (que fazem fronteira com) o' (Figura 7).

$$AQ(o') = \{o \mid ((o'.G \cap o.G) \neq \emptyset) \wedge (o'.G^o \cap o.G^o) = \emptyset\}$$

Sendo que $o'.G^o$ e $o.G^o$ correspondem à região interior às extensões espaciais $o'.G$ e $o.G$.

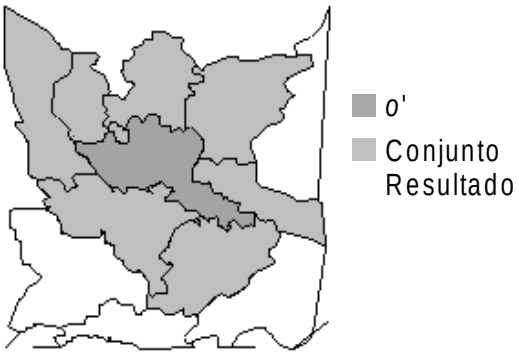

Figura 7: Exemplo de uma operação ***Adjacency Query***.

2.3.2.8 Nearest Neighbor Query/NNQ

Considere-se um objeto o' com extensão espacial $o'.G \subseteq E^d$. Esta operação resulta em um subconjunto com todos os objetos o que possuem a menor distância em relação a o'. É interessante lembrar que vários objetos podem estar ocupando uma posição ou porção do espaço (colisão), daí a possibilidade de existirem mais de um com distância mínima.

$$NNQ(o') = \{o \mid \forall o'': dist(o'.G, o.G) \leq dist(o'.G, o''.G)\}$$

A distância entre dois objetos complexos é geralmente definida como sendo a distância entre seus pontos mais próximos. A função distância entre dois objetos espaciais $dist(o'.G, o''.G)$

corresponde, normalmente, à distância *Euclideana*. Algumas vezes é utilizada a função distância *Manhattan*[6] (conhecida como "*city block*").

Uma variação desta operação corresponde a obter um subconjunto com os *k* objetos mais próximos a um certo objeto o'.

2.3.2.9 *Spatial Join*

Considere-se duas coleções, *R* e *S*, de objetos espaciais e um predicado θ. Esta operação resulta em um subconjunto com todos os pares de objetos $(o, o') \in R \text{ X } S$ em que $\theta(o.G, o'.G)$ tem valor verdadeiro.

$$R \bowtie_\theta S = \{(o, o') \mid (o \in R) \wedge (o' \in S) \wedge \theta\,(o.G, o'.G)\}$$

Gaede e Günther [GaedeV_97] relacionam, como exemplo, as seguintes possibilidades para o predicado θ:

- *faz_interseção_com*$(o.G, o'.G)$
- *contém*$(o.G, o'.G)$
- *é_englobado_por*$(o.G, o'.G)$
- *distância*$(o.G, o'.G)\Theta q$, com $\Theta \in \{=, \leq, <, \geq, >\}$ e $q \in E^1$
- *está_a_noroeste*$(o.G, o'.G)$
- *é_adjacente*$(o.G, o'.G)$

O predicado espacial *faz_interseção_com*, $R \bowtie_{faz\_interseção\_com} S$, merece um destaque especial por melhorar o desempenho computacional de diversos outros predicados. Por exemplo: para os predicados *contém*, *é_englobado_por* e *é_adjacente*, o predicado *faz_interseção_com* pode ser usado, na etapa de filtragem de uma consulta espacial, para gerar um conjunto candidato tipicamente muito menor que o produto Cartesiano *R* X *S* [GaedeV_97].

[6] A distância *Manhattan* entre dois pontos (x_1, y_1) e (x_2, y_2) é $|x_1 - x_2| + |y_1 - y_2|$ [SametH_90].

Algoritmos para a execução da operação *Spatial Join* podem ser encontrados em [PatelJ_96] [BrinkhoffT_94] [KriegelH_92] [ZhouX_97].

2.4 Exemplos de Aplicação e Trabalhos Relacionados

Atualmente diversas áreas do conhecimento têm usado BD espaciais. A seguir, estão relacionados alguns exemplos de aplicações que utilizam BD espaciais e alguns trabalhos recentes envolvendo MIEs:

1. **Medicina** [TamuraH_84]: esta talvez seja a área de maior motivação, tendo diversos trabalhos já desenvolvidos. São exemplos de aplicações: armazenamento e recuperação de imagens e auxílio no estudo de casos através da identificação de imagens com algum tipo de semelhança. Petrakis e Faloutsos, em [PetrakisE_97], propõem um método para facilitar consultas por similaridade a BDs de imagens utilizando Grafos Relacionais Rotulados (*Attributed Relational Graphs*) para representar o conteúdo das imagens.

2. **Base de dados Temporais e de Seqüências Amostrais** [WhiteD_96]: aqui são respondidas, com o auxílio dos MIEs, perguntas como “encontre ações da bolsa de valores que apresentam uma flutuação de mercado com o seguinte padrão...” e “enconte uma seqüência de DNA com padrões semelhantes à desta...”. Nascimento e Silva em [NascimentoM_98] apresentam uma extensão do MIE R-Tree [GuttmanA_84] chamada Historical R-Tree, usada para auxiliar consultas em BDs temporais em SGBDs espaciais que utilizam o MIE R-Tree [GuttmanA_84].

3. **Processamento de dados Geográficos** (ou “Sistema de Informações Geográficas”) [BentleyJ_79] [TamuraH_84] [GuentherO_90]: aqui as BDs espaciais auxiliam na elaboração de mapas e no armazenamento e recuperação de dados como: distribuição populacional, relevo, acidentes geográficos, planta de uma cidade, etc.. Auxiliam, também, no registro da evolução destes dados no decorrer do tempo. As operações mais comuns são: *Containment Query*, *Nearest Neighbors Query*, *Adjacency Query*, e *Spatial Joins*.

4. **Sensoriamento remoto** [TamuraH_84]: imagens e outros tipos de dados provenientes de sensores em satélites ao redor da Terra têm sido intensamente utilizados para fazer previsões meteorológicas, detecção de queimadas, detecção de plantações para produzir matéria prima usada na fabricação de drogas como a cocaína e a maconha, dentre outras. BD espaciais são utilizadas para auxiliar na análise e armazenamento destes dados.

5. **Sistemas de visão artificial** [GuentherO_90]: auxiliando áreas como robótica, linhas de produção industriais e navegação. O problema aqui envolve a identificação e localização de objetos tridimensionais a partir de imagens bidimensionais. Estas áreas geralmente exigem que interpretações e decisões rápidas sejam feitas, sendo de grande utilidade a aplicação de MIEs.

6. **BDs do tipo Multimídia** (*Multimedia Databases*) [LinK_94]: que armazenam áudio (sons, vozes, músicas, etc.), vídeo, etc.. Consultas típicas são: “obter uma música que apresenta uma seqüência de sons como...”, “encontrar em um filme uma imagem com determinadas características ou semelhantes a uma imagem dada...”, etc..

7. **Tratamento de cadeias de caracteres** [LinK_94]: envolvendo correção ortográfica, correção de erros em textos obtidos por reconhecimento óptico de caracteres (*Optical Character Recognition - OCR*), busca de seqüências de cadeias de caracteres em textos muito grandes, etc.. Com relação a este último exemplo, uma aplicação interessante é a busca de cadeias de caracteres parecidas (ou com erro): procurar em uma agenda eletrônica o endereço “Rua Major Alberto, no. 1234” quando o endereço correto é “Rua Major Carlos Alberto, no. 4321”.

2.5 Conclusão

Neste capítulo foram introduzidos os conceitos em torno de dados espaciais: o que são, como são representados em meio computacional, propriedades e as dificuldades que os mesmos apresentam em termos de armazenamento e recuperação. Foram apresentados os operadores espaciais mais comuns encontrados na literatura e como o desempenho de uma consulta espacial pode ser melhorada se feita em duas etapas. Por fim, foram apresentadas algumas aplicações que utilizam dados espaciais.

3. MÉTODOS DE INDEXAÇÃO ESPACIAL

3. MÉTODOS DE INDEXAÇÃO ESPACIAL

3.1 Introdução

Devido à crescente necessidade de indexar dados espaciais, diversos métodos têm sido desenvolvidos e aperfeiçoados. A Figura 8 apresenta a evolução destes ao longo de trinta anos de pesquisas, mostrando os métodos originais e suas derivações em ordem cronológica:

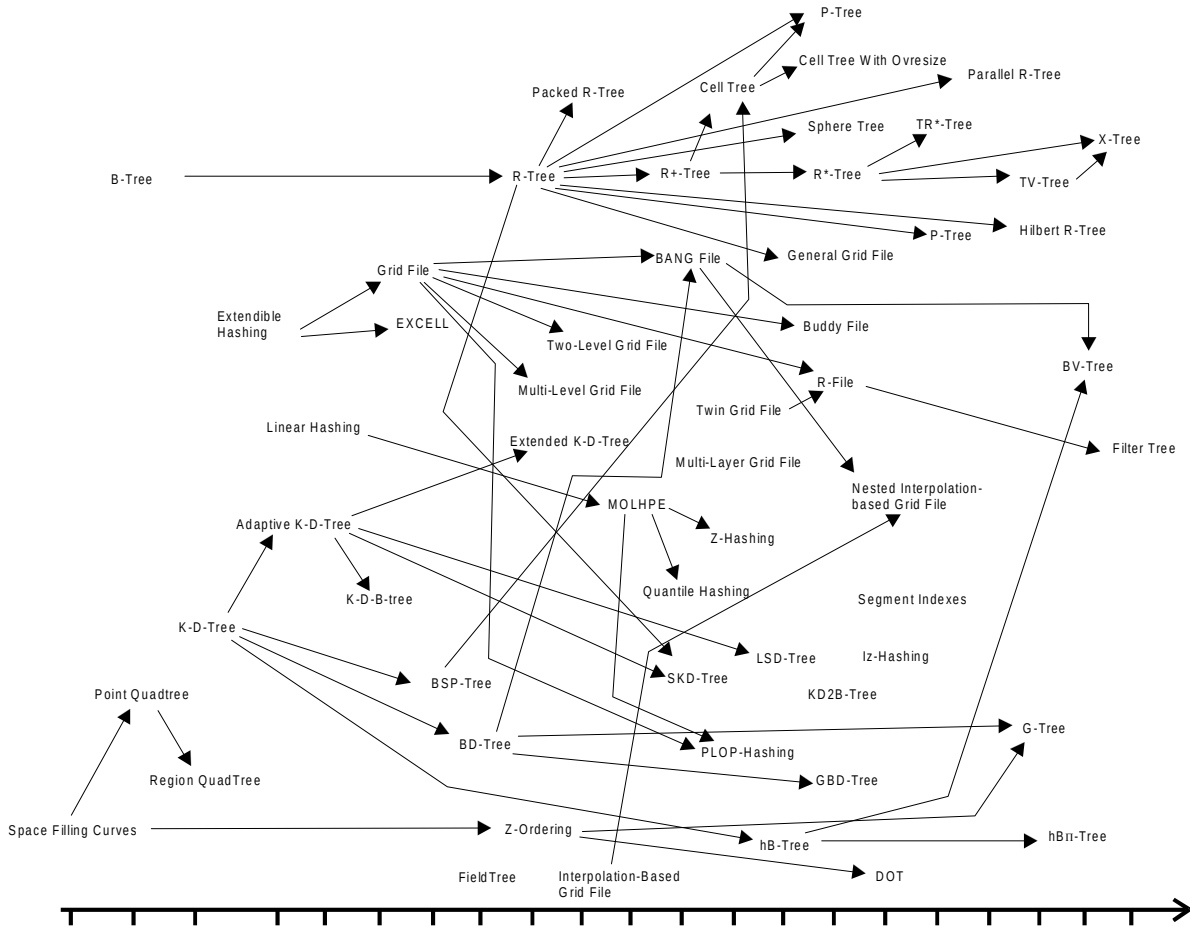

Figura 8: Evolução dos Métodos de Indexação Espacial [GaedeV_97]

De forma geral, estes MIEs podem ser diferenciados considerando as seguintes características:

- **O tipo de dados para o qual são utilizados**: pontuais ou espaciais. Para lidar com dados espaciais, alguns dos MIEs aplicam operações de transformação aos mesmos gerando pontos representados em um espaço cuja dimensão pode ser igual, maior, ou

até menor do que a do original [SametH_95]. Este procedimento é, geralmente, aplicado quando um *PAM* é utilizado.

- **A forma como o espaço é representado**:
    - O espaço multidimensional pode ser mapeado para um espaço unidimensional [JagadishH_90]. Este processo será tratado aqui por[7] **transformação do espaço** (ou transformação espacial);
    - Alternativamente, é mantida a dimensão original do espaço. Neste caso, o mesmo é dividido em regiões (ou porções) e a indexação é feita em função destas regiões. A este processo é dado o nome de **decomposição do espaço** [SametH_90] (ou decomposição espacial). Existe, ainda, uma classificação deste processo em função dos princípios que determinam como é feita esta decomposição:
        - a decomposição pode ser feita em regiões iguais, sendo conhecida como **decomposição regular do espaço** [SametH_90] (ou, decomposição espacial regular); ou pode ser feita em função da distribuição espacial dos dados, sendo tratada, aqui, por **decomposição irregular do espaço** (ou, decomposição espacial irregular). Este processo pode ser aplicado mais do que uma vez (dito em vários níveis), o que significa que as regiões geradas em um nível serão subdivididas no nível seguinte utilizando o mesmo critério (em regiões iguais ou não);
        - o número de vezes que este processo é aplicado, conhecido por **resolução da decomposição** [SametH_90], pode ser fixo (pré-determinado, indicado simplesmente por **resolução fixa**) ou não (por ser o caso mais comum, apenas a resolução fixa é indicada). Quando este princípio é utilizado mais do que uma vez, o processo passa a ser denominado **decomposição [fixa] recursiva do espaço** (ou decomposição espacial [fixa] recursiva). Pode, então, ser feita uma

[7] Algumas das classificações listadas não possuem um termo específico para identificá-las (ou o mesmo não foi encontrado na literatura). Portanto, quando for dito que uma classificação (ou processo, ou método, etc.) "será tratada aqui por ...", entenda-se que não foi encontrado um termo comum (ou consensual) na literatura estudada.

decomposição espacial regular recursiva, ou seja, a decomposição espacial é feita em regiões iguais e aplicada mais do que uma vez;

- Kriegel, Schiwietz, Schneider e Seeger, em [KriegelH_90], fazem, ainda, as seguintes distinções: as regiões geradas pela decomposição podem ser mutualmente **disjuntas** duas a duas, ou **não disjuntas**; essas regiões podem ser **intervalares**, em que o espaço é dividido por hiperplanos ortogonais aos seus eixos, ou **não intervalares** (o termo usado originalmente é retangular, mas aqui será usado intervalares por ser mais abrangente quanto à dimensionalidade) e o **particionamento** pode ser **completo** ou **incompleto**, ou seja, a união de todas as regiões geradas pelo particionamento regeneram o espaço original, ou apenas as porções onde se encontram os dados, respectivamente.

- **A estrutura de dados básica** [SametH_90] [KnuthD_73]: para organizar esses dados podem ser utilizadas estruturas baseadas em árvores, ditas **hierárquicas**, ou estruturas lineares, como vetores por exemplo, ditas **não hierárquicas**.

- **Quantidade de dados por unidade de armazenamento** (nó de uma árvore, posição de um vetor, etc...): quando uma unidade de armazenamento das estruturas de dados agrupa mais do que um dado (ou simplesmente **entrada**), elas são tratadas por ***bucket*** (em português significa balde, cesto ou caçamba) e as estruturas por **estrutura *bucket*** (ou seja, estruturas que indexam *buckets*) [NievergeltJ_84] [FreestonM_87] [SametH_90]. Aquelas estruturas cuja unidade de armazenamento contém apenas um dado são tratadas aqui por **estruturas não *bucket***. Os métodos de indexação, em função destas estruturas, são chamados, respectivamente, **métodos *bucket*** e **métodos não *bucket***. É comum, ainda, usar o termo **diretório** [FreestonM_87] [SametH_90] para indicar as estruturas *bucket*. Geralmente, quando se trata de uma estrutura deste tipo, presume-se que a mesma é organizada de forma a permitir armazenamento secundário, em que uma página de disco (ou bloco) corresponde a um *bucket*[8].

[8] Segundo Freeston, em [FreestonM_87], esse termo é usado ao invés de bloco para enfatizar que o seu conteúdo não está, necessariamente, ordenado.

Tendo em vista o grande número de MIEs (como pôde ser visto na Figura 8), apenas alguns serão aprofundados a título de ilustração. Para tanto, será considerado o critério baseado no tipo de dados para classificar os métodos como *PAM* ou *SAM*, por ressaltar melhor o interesse existente neste trabalho (uma classificação mais detalhada pode ser obtida em [SametH_90]).

3.2 Métodos de Acesso Pontual (*Point Access Methods - PAM*)

3.2.1 *Quadtree* (e algumas de suas variações)

O termo *quadtree* tem sido empregado com vários significados. De forma geral, é usado para descrever uma classe de estruturas hierárquicas com a propriedade comum de terem como princípio a decomposição espacial recursiva, podendo ser diferenciadas a partir das seguintes características ([SametH_90]):

- Tipo de dados que indexa: pontuais ou espaciais;
- Os princípios que determinam como é feita a decomposição espacial (regular ou irregular, fixa ou não);

Apesar de ser utilizado, inicialmente, para estruturas em 2 dimensões (em terceira dimensão, geralmente é usado *Octree*), o seu conceito pode ser adaptado para um número qualquer de dimensões. Como exemplo será apresentado o *PAM point quadtree* [SametH_90], usado para indexar pontos através da decomposição espacial irregular recursiva.

O *point quadtree* pode ser vista, também, como uma generalização multidimensional da árvore de busca binária em que os nós interiores têm 2^d filhos (onde d corresponde à dimensão do espaço), cada um representando um intervalo do espaço original, centrado no ponto correspondente ao nó.

Exemplo

Considere-se um conjunto de pontos situados no espaço bidimensional (Figura 9a). Cada um destes pontos pode ser visto como um nó de uma *quadtree* (Figura 9b), representado por um registro do tipo *Node* com sete campos (Figura 9c). Os quatro primeiros correspondem a

ponteiros para os quatro filhos relacionados às direções dos subespaços (quadrantes) gerados pela decomposição: noroeste (*NW*), nordeste (*NE*), sudoeste (*SW*) e sudeste (*SE*). Dois campos (*XCoord*, *YCoord*) são destinados a armazenar as coordenadas (*x*, *y*) referentes a cada dimensão. E o último campo (*ObjInfo*) contém informações descritivas sobre o ponto, por exemplo o nome da cidade que o mesmo representa (como mencionado na seção 2.2, este campo pode conter um ponteiro para os atributos não-espaciais do objeto armazenados separadamente).

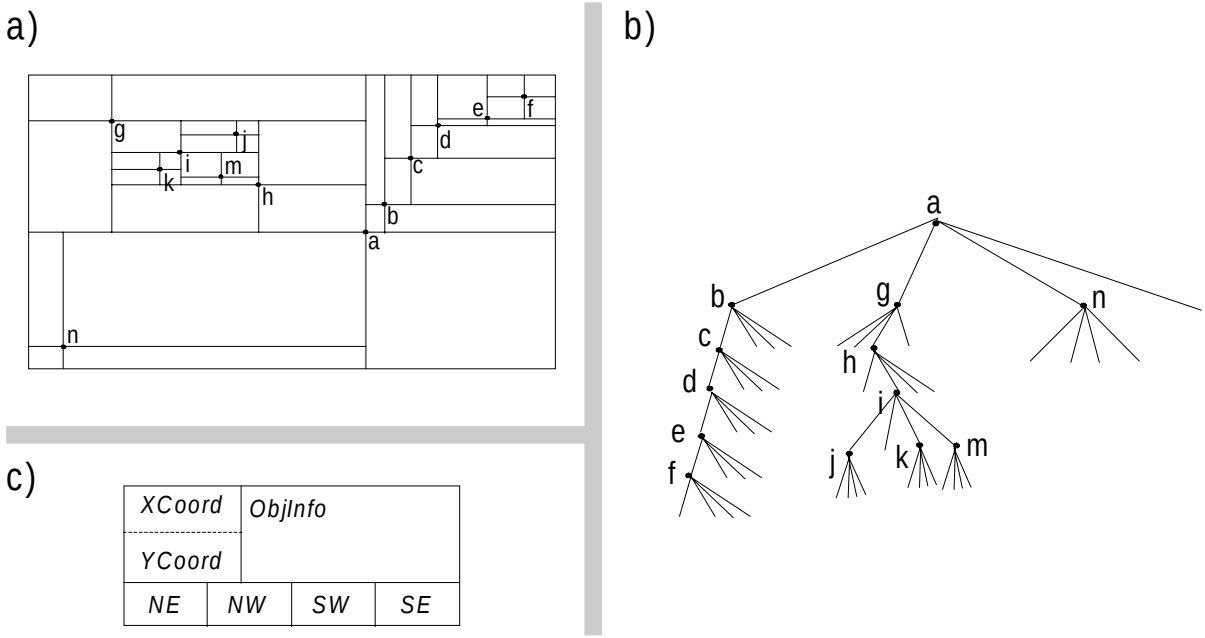

Figura 9: Exemplo de uma ***Quadtree****. a) Representação espacial; b) Representação esquemática; c) Estrutura (****Node****) de um nó.*

Inserção

O procedimento de inserção é similar ao feito em árvores binárias, com a diferença de que a árvore é percorrida em função do quadrante em que o ponto a ser incluído se encontra, o qual pode ser facilmente identificado comparando-se suas coordenadas às do ponto referente ao nó obtido a cada descida. Outra diferença é a necessidade de especificar critérios para tratar as possíveis colisões (ver seção 2.2).

Busca

A busca, assim como a inserção, se assemelha à feita em árvores binárias com as características já descritas anteriormente quanto a percorrer a árvore e seus quadrantes, tomando-se os devidos

cuidados com as colisões. Em operações do tipo *Window Query* é necessário percorrer apenas as subárvores que fazem interseção com o intervalo de busca.

Remoção

A remoção de um nó implica na restruturação da subárvore iniciada por este. Uma opção é reinserir todos os nós que estão abaixo daquele que está para ser removido. Porém esta abordagem pode ser muito custosa dependendo do número de nós a serem reinseridos. Em [SametH_90], são citadas algumas alternativas a esta opção.

Vantagens e desvantagens

Apesar de ser muito simples, dinâmico, de fácil implementação, e muito útil para operações de busca [SametH_90], este MIE apresenta alguns problemas:

- ao percorrer uma árvore deste tipo, o número de comparações cresce à medida em que aumenta a dimensão do espaço, pois cada uma das coordenadas, uma para cada dimensão, deve ser comparada;
- os nós folha (ou simplesmente folhas) apresentam um alto custo de armazenamento devido à grande quantidade de ponteiros com valor nulo (que também é proporcional à dimensão);
- o tamanho de um nó cresce exponencialmente em função da dimensão, pois cada nó tem 2^d filhos (em que *d* corresponde à dimensão do espaço) tornando necessários 2^d ponteiros por nó;
- foi desenvolvida, a princípio, para ser utilizada em memória principal;
- por fim, possui grande dependência quanto à seqüência de inserção dos dados, podendo ser degenerada, em seu pior caso, para uma lista encadeada (ver subárvore *b* na Figura 9b). Em outras palavras, seu algoritmo não gera uma árvore de busca balanceada [TenembaumA_96].

Em [SametH_90] são apresentadas outras variações da *quadtree*, como por exemplo a ***PR-Quadtree*** (*P* de *point* - ponto, e *R* de *region* - região) que é uma estrutura hierárquica com

decomposição espacial regular recursiva, porém limitada também à memória principal. Ou a ***Bucket PR-Quadtree***, com diferença, em relação à anterior, de ser uma estrutura *bucket*.

3.2.2 *K-D Tree* (e algumas de suas variações)

Este MIE é considerado como um dos mais importantes *PAM* [GaedeV_97]. O termo *k-d tree* significa árvore ***k-d***imensional (onde *k* corresponde à dimensão do espaço). Em princípio, esta estrutura é uma árvore de busca binária [SametH_90] [GaedeV_97] que faz a decomposição não regular de um espaço *d*-dimensional através de hiperplanos ($d - 1$)-dimensionais. Esses hiperplanos são iso-orientados, com suas direções alternando à medida em que *d* varia. Ou seja, para $d = 3$, os hiperplanos são perpendiculares aos eixos *x*, *y* e *z*, sucessivamente. O particionamento do espaço é feito à medida em que a árvore é percorrida em profundidade.

Exemplo

Considere-se um conjunto de pontos situados no espaço bidimensional (Figura 10a). Cada um destes pontos pode ser visto como um nó de uma *k-d-tree* (Figura 10b), representado por um registro do tipo *Node* com seis campos (Figura 10c). Os dois primeiros (*FEsquerdo* e *FDireito*) correspondem a ponteiros para os dois filhos relacionados às direções dos subespaços gerados pela decomposição. Os dois seguintes (*XCoord, YCoord*) são destinados a armazenar suas coordenadas (*x, y*) referentes a cada dimensão. Outro campo (*ObjInfo*) contém informações descritivas sobre o ponto, e o último (*Disc* - referente a discriminador) indica o eixo sobre o qual o hiperplano está decompondo o espaço.

Por convenção, quando um nó *P* é um discriminador do eixo *x* (ou simplesmente discriminador *x*) tem-se que, dentre todos os nós subsequentes, aqueles que possuem coordenada *x* menor do que a sua pertencem à subárvore esquerda do mesmo (indicada por *FEsquerdo*). Por conseqüência, todos os nós subseqüentes que possuem coordenada *x* maior do que a de *P* estão na sua subárvore direita (indicada por *FDireito*). Esta convenção é aplicada sucessivamente a cada nível da árvore, alternando os discriminadores (*x*, em seguida y e retornando ao *x*, supondo $d = 2$). O campo *Disc* não é exatamente necessário, visto que o discriminador pode ser facilmente calculado em função do nível da árvore e da dimensão do espaço.

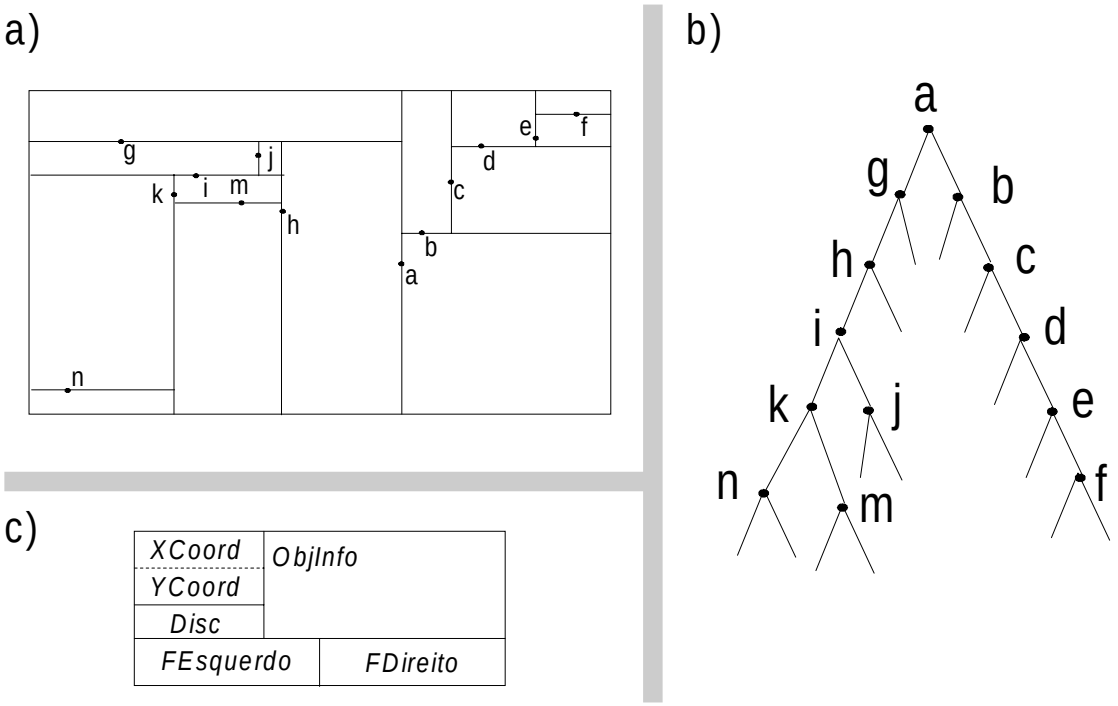

*Figura 10: Exemplo de uma **K-D-Tree**. a) Representação espacial; b) Representação esquemática; c) Estrutura (**Node**) de um nó.*

Inserção

Os pontos são inseridos de maneira similar à inserção em árvores binárias, com a diferença de que a árvore é percorrida em função do subespaço gerado pelo eixo discriminador: *x* para níveis ímpares (supondo o nível inicial - raiz - com valor 1) e *y* para níveis pares. Novamente, são necessários critérios para tratar as possíveis colisões.

Busca

A busca também é feita de forma semelhante à busca em árvores binárias, tendo sido praticamente descrita acima, tomando-se os devidos cuidados com as colisões. Em operações do tipo *Window Query* é necessário percorrer apenas as subárvores que fazem interseção com o intervalo de busca, diminuindo o número de comparações.

Remoção

A remoção de um nó em uma *k-d-tree* é muito mais complicada do que em uma árvore binária. Como pode ser observado na Figura 11, obtida em [SametH_90], ao contrário das árvores binárias nem toda subárvore da *k-d-tree* corresponde a uma outra árvore do mesmo tipo. Por exemplo, apesar de a árvore apresentada na Figura 11a corresponder a uma *k-d-tree*, sua subárvore direita (Figura 11b) não corresponde. O problema existente é que, no caso de um espaço bidimensional o nó raiz particiona o espaço em função do eixo *x*, ao passo que, os dois filhos do nó raiz da árvore na Figura 11b possuem a coordenada *x* maior que a do nó raiz. Um algoritmo para remoção é descrito de forma detalhada em [SametH_90].

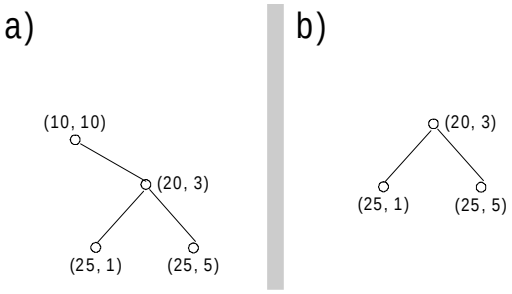

*Figura 11: Exemplo de uma **K-D-Tree** (a) em que seu filho direito (b) não corresponde a uma **K-D-Tree**.*

Vantagens e desvantagens

De forma geral, esta tem sido uma das estruturas de dados mais estudadas dentre as que se enquadram como *PAM*, servindo de base para a elaboração de diversas outras, como mostra a Figura 8 na seção 3.1. Isso se deve a motivos tais como:

- dinamismo;
- eficiência temporal em operações espaciais;
- é fácil de ser implementada, apesar da remoção apresentar problemas;
- os nós terminais não apresentam grande custo de armazenamento, pois só são armazenados dois ponteiros por nó (um para cada subárvore) independentemente da dimensão do espaço;
- por fim, e talvez o principal motivo, durante a busca o número de comparações feitas por nó independe da dimensão do espaço, pois apenas a coordenada especificada pelo discriminador deve ser comparada;

No entanto, sua implementação original apresenta, também, alguns problemas:

- o tamanho de um nó cresce em função da dimensão, pois cada nó precisa armazenar todas as coordenadas do ponto;
- foi desenvolvida inicialmente para trabalhar totalmente em memória principal;
- por fim, possui grande dependência quanto à seqüência de inserção dos dados, ou seja, não gera uma árvore de busca binária balanceada (ver subárvore *b* na Figura 10b).

Samet, em [SametH_90], coloca ainda uma vantagem da *quadtree* sobre a *k-d-tree*: a primeira pode ser percorrida em paralelo, pois a subárvore de uma *quadtree* é também uma *quadtree*, além da possibilidade de as comparações que envolvem as coordenadas serem feitas em paralelo. Já a *k-d-tree* não suporta este tipo de implementação (ver remoções em *k-d-tree*). Como conseqüência, Samet qualifica a primeira como uma estrutura de dados paralela superior, e a segunda, como uma estrutura de dados serial superior.

Como dito anteriormente, esta estrutura serviu de base para diversas outras, as quais procuram apresentar alguma melhoria sobre a original:

- ***Adaptive K-D-Tree*** [SametH_90] [GaedeV_97]: os dados são armazenados apenas nas folhas; os nós interiores contém a mediana (em relação a uma das coordenadas) do conjunto de dados e o discriminador. Conseqüentemente, as partições do espaço não são feitas sobre os pontos, mas sim em função desta mediana. Sua principal vantagem é que gera uma árvore com os nós mais bem distribuídos (tendendo a uma árvore balanceada). Porém, seu uso é limitado a BDs estáticas, nas quais todo o conjunto de pontos precisa ser conhecido antes do início do processo de inserção para que sejam calculadas as medianas. Além disso, seu projeto inicial não prevê armazenamento secundário.
- ***Bintree*** [GaedeV_97], também conhecida por ***PR K-D-Tree*** e ***PR trie*** [SametH_90]: suas principais diferenças em relação à *k-d-tree* são: faz decomposição espacial regular e os dados são armazenados apenas nas folhas. Sua vantagem é que os hiperplanos que particionam o espaço são conhecidos implicitamente, o que, dentre outras vantagens, diminui a quantidade de informação a ser armazenada em cada nó e simplifica o processo de remoção [SametH_90]. Suas principais desvantagens são: uma grande

quantidade de subdivisões pode vir a ser necessária para acomodar os pontos em subespaços distintos, pois a decomposição é regular e tem que ser efetuada até gerar subespaços contendo apenas um ponto; e há a necessidade de se saber, de antemão, o intervalo do espaço onde os pontos estão definidos, ou seja, o espaço precisa ser limitado para que a decomposição seja regular. Seu projeto inicial não prevê armazenamento secundário.

- ***BSP Tree*** [SametH_90] [GaedeV_97]: dentre as principais diferenças em relação à *k-d-tree* estão: os dados são armazenados apenas nas folhas, os hiperplanos que particionam o espaço não são necessariamente iso-orientados e não há uma seqüência pré-definida quanto ao eixo discriminador. Estas duas últimas diferenças tornam mais flexível a orientação dos hiperplanos de forma a permitir um particionamento mais adequado, que é feito em função da distribuição dos objetos no espaço. Outra vantagem é que o processo de remoção pode ser simplificado, por não existirem restrições quanto ao discriminador. Suas principais desvantagens são: necessita armazenar uma quantidade de informações maior do que métodos como o *adaptive k-d-tree*, pois requer que sejam indicados explicitamente o discriminador e a orientação do hiperplano; e é mais indicado, mas não necessário, que se conheça o conjunto de pontos antes de ser feita a inclusão para que seja realmente otimizado o particionamento, caso contrário não apresentaria grandes vantagens em relação à *k-d-tree* (isto **não** implica em uma estrutura estática); seu projeto inicial não prevê armazenamento secundário e, por fim, um cuidado especial deve ser tomado quanto ao critério utilizado para particionar o espaço, procurando evitar os que são degradados à medida em que a ordem de dimensão aumenta, ou seja, que não dependam diretamente desta ordem.

- ***K-D-B-Tree*** [GaedeV_97]: esta estrutura faz uma fusão entre as propriedades das estruturas ***Adaptive k-d-Tree*** [SametH_90] e ***B-Tree*** [ComerD_79]. Seu conceito consiste basicamente em particionar o espaço seguindo o mesmo processo que a primeira estrutura, associando os subespaços gerados com os nós de uma árvore com características semelhantes à segunda. Ou seja, assim como a *B-Tree*, a *k-d-B-Tree* é uma árvore perfeitamente balanceada que apresenta boa adaptação à distribuição dos dados. Porém, ao contrário do que ocorre com a *B-Tree*, não há garantias de eficiência

de armazenamento. Cada nó interior corresponde a um conjunto de regiões (intervalos) do espaço, utilizando o mesmo processo de decomposição apresentado pela *k-d-tree*. Os pontos ocupam apenas as folhas, nas quais os subespaços são limitados pelas regiões definidas nos respectivos nós pais (ou simplesmente pai). Buscas são executadas da mesma forma que nas *k-d-trees*. Para a inserção, primeiramente é feita uma busca para se encontrar o nó onde será feita a inclusão e, se o mesmo não estiver completo, a inclusão é feita. Caso contrário, o nó é particionado (dividido, rompido) e cerca de metade dos dados que o compõe é transferido para um novo nó, forçando mais uma entrada no seu pai. Este processo é feito recursivamente até que o nó imediatamente acima do que está sendo quebrado permita mais uma entrada, ou até ser atingida a raiz, quando um novo nó é criado. O critério utilizado para particionar um nó afeta diretamente os nós filhos, pois estes devem ser também particionados e ajustados à nova divisão feita no seu antigo pai em função dos subespaços que foram remanejados. Este ajuste, por sua vez, pode gerar um nó que não atenda os requisitos próprios a *B-Trees* quanto a sua ocupação mínima, não sendo possível, portanto, uma garantia quanto ao armazenamento. A remoção é feita de forma simples, implicando em uma busca e possíveis junções em nós internos, quando da ocorrência de um nó folha com poucos pontos. Quando uma junção ocorre, no mínimo um hiperplano em nós pais deve ser removido. Finalizando, esta estrutura apresenta como principal vantagem em relação à *k-d-tree* a utilização de memória secundária, tornando-a uma dos mais completos *PAM*, segundo os requisitos apresentados na seção 2.2.

Todos os *PAM* apresentados acima são disjuntos. Apenas o *PAM BSP-Tree* é não intervalar. Apenas os *PAM PR-Quadtree, Bucket PR-Quadtree, Adaptive K-D-Tree* e *K-D-Tree* são incompletos. Os únicos métodos *bucket* são o *K-D-B-Tree* e o *Bucket PR-Quadtree*. Samet em [SametH_90] coloca que os demais métodos, não *bucket*, podem ser estendidos de forma a suportar *buckets*, tanto em uso restrito à memória primária, quanto permitindo armazenamento secundário.

3.2.3 Outros *PAMs*

Existem, ainda, diversos outros *PAMs* que apresentam características diferentes das apresentadas acima:

PAMs que fazem transformação espacial (para dimensão 1) [JagadishH_90]

A forma mais conhecida de transformação espacial (ou linearização espacial) é a transformação de uma matriz bidimensional em um vetor unidimensional. Em se tratando de dados espaciais, o processo é praticamente o mesmo, variando apenas a função utilizada para mapear o espaço. Estas funções são aplicadas sobre as coordenadas do ponto a ser mapeado. Por exemplo (os exemplos a seguir supõem o espaço bidimensional, mas os conceitos apresentados podem ser estendidos para qualquer dimensão):

1. Uma das funções mais simples é a semelhante ao mapeamento de matrizes. Para tanto, é necessário saber de antemão o número máximo de linhas (*MAXLIN*) e colunas (*MAXCOL*) da matriz, pois após a transformação os dados serão armazenados em um vetor com (*MAXLIN * MAXCOL*) elementos. A função corresponde a: considerando um ponto de coordenadas (*x*, *y*), sua posição no vetor (nova coordenada) é obtida por ((*x * MAXLIN*) + *y*). Esta função é conhecida como *Column-wise Scan*. A Figura 12, obtida em [JagadishH_90], mostra a como a matriz bidimensional estaria representada.

2. Outra transformação consiste em intercalar os *bits* (conhecido como *bit interleaving*) das coordenadas (*X*, *Y*) dos pontos, onde os *bits* de y são, arbitrariamente, tidos como mais significativos. Ou seja, considerando um ponto de coordenadas (X, Y) = (x_m x_{m-1}... x_1 x_0, y_m y_{m-1}... y_1 y_0), sua nova posição é obtida por y_m x_m y_{m-1} x_{m-1}... $y_1 x_1$ y_0 x_0. A Figura 13, obtida em [SametH_90], mostra como a posição do ponto estaria representada. Devido ao formato da linha esta função é conhecida por *Z-Curve* (ou *N-Curve*) [SametH_90]. Esta função é conhecida também por: *Peano-Code*, *Z-ordering*, *Locational-Codes* [GaedeV_97]. Outras funções semelhantes são: *Gray Code* e *Hilbert Curve* [JagadishH_90].

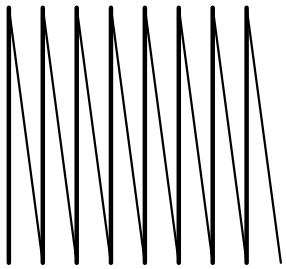

Figura 12: Transformação espacial com aplicação da função de mapeamento ***Column-wise Scan***.

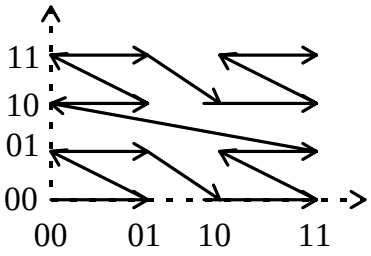

Figura 13:Transformação espacial com aplicação da função de mapeamento ***Z-Curve***.

Os MIEs que fazem transformação espacial utilizando *bit interleaving* (também conhecidos por *Space Filling Curves*), geralmente dividem os vetores em fragmentos e utilizam algum diretório para indexá-los. Esses métodos são classificados como *PAMs* que fazem transformação espacial, mapeando os dados para estruturas não hierárquicas armazenadas em diretórios. As regiões espaciais geradas são disjuntas e intervalares, com particionamento completo [KriegelH_90]. Segundo *Samet* [SametH_90], estas estruturas são muito eficientes quanto ao tempo para busca e *Window Query* (e suas variantes). Porém, trabalham com dados discretos e, geralmente, conjuntos finitos. Uma descrição mais aprofundada sobre estes métodos contendo: variações, aplicações, medidas de desempenho e análises de resultados experimentais, pode ser obtida em [JagadishH_90] [SametH_90].

3.3 Métodos de Acesso Espacial (*Spatial Access Methods - SAM*)

3.3.1 *R-Tree*

O *R-Tree* [GuttmanA_84] é, sem sombra de dúvidas, o *SAM* mais referenciado na literatura, sendo o mais importante entre os que aplicam o conceito de *MBR* (ver seção 2.3.1). Consiste basicamente de um *SAM* que faz decomposição espacial irregular recursiva, organizada em diretórios hierárquicos, com particionamento incompleto e regiões intervalares não disjuntas.

A *R-Tree* é uma árvore balanceada em função da altura (todos os nós folha se encontram no mesmo nível), sendo que as folhas contém ponteiros para os atributos não espaciais dos objetos.

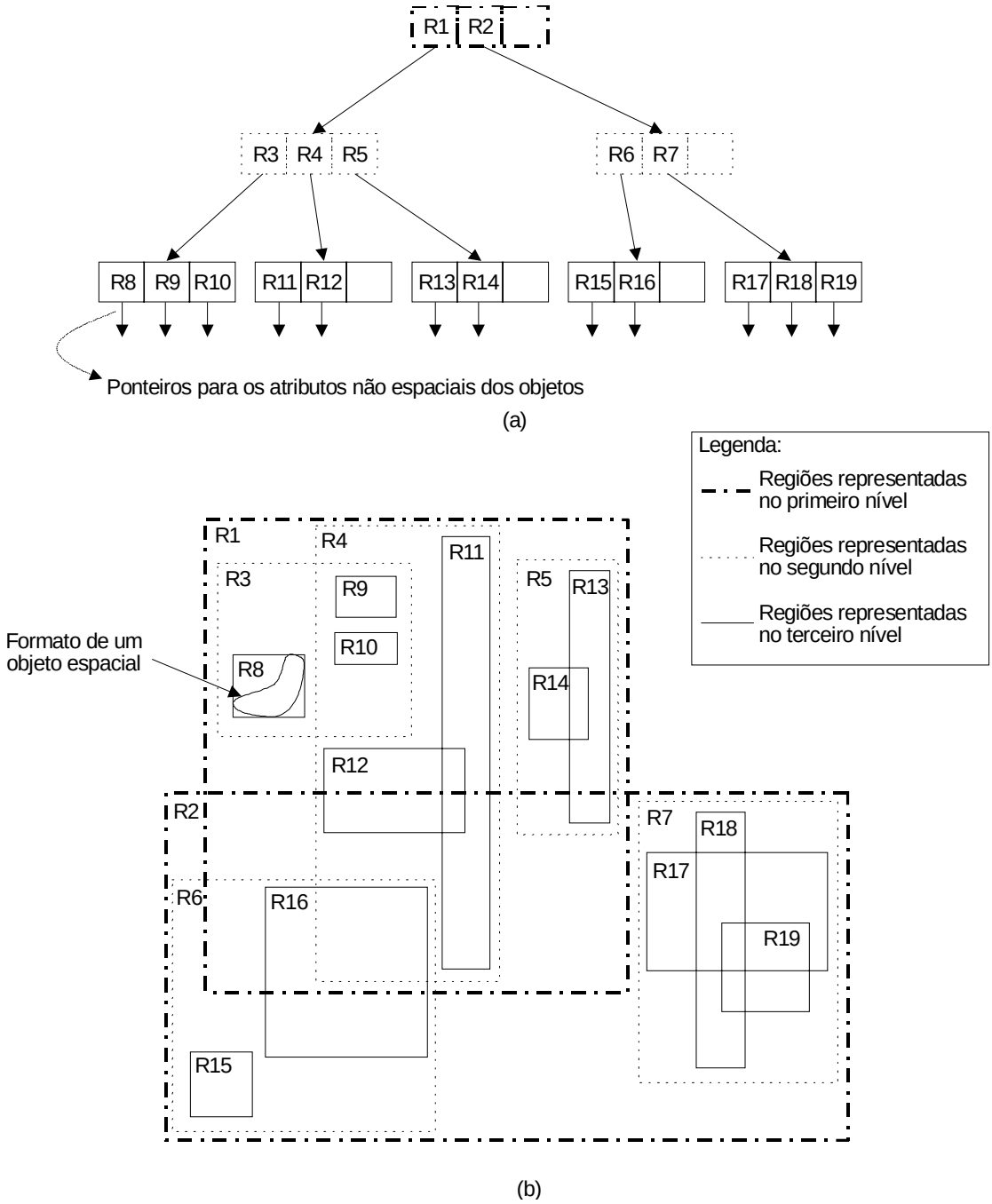

*Figura 14: Exemplo de uma **R-Tree**. a) Representação em memória; b) Representação espacial com os MBR.*

Os nós correspondem a páginas de disco, com as seguintes características (ver Figura 14, obtida em [GuttmanA_84]):

- cada folha contém entradas do tipo (*I*, *Tid*), em que *Tid* faz referência a uma tupla na BD, contendo os atributos não espaciais do objeto. E *I* corresponde ao *MBR* deste objeto;

- cada nó interno contém entradas do tipo (*I, Cid*), em que *Cid* é um ponteiro para um nó filho. Ao passo que *I* (atributos espaciais) corresponde ao *MBR* que cobre todas as regiões descritas pelos *MBR*s dos nós inferiores;
- Considere-se *M* o número máximo de entradas em um nó. Assuma-se, então, $m \leq M/2$ o parâmetro que especifica o número mínimo de entradas em um nó.7

A *R-Tree* apresenta as seguintes propriedades:

- Toda folha contém entre *m* e *M* entradas, a não ser que seja a raiz;
- Para cada entrada do tipo (*I, Tid*) em uma folha, *I* é o *MBR* que contém o objeto descrito por *Tid*;
- Todo nó interno tem entre *m* e *M* filhos, a não ser que seja a raiz;
- Para cada entrada do tipo (*I, Cid*) em um nó interno, *I* é o *MBR* que contém todos os *MBRs* ds seus filhos;
- A raiz tem no mínimo dois filhos, a não ser que seja uma folha;
- Todas as folhas aparecem no mesmo nível.

Busca

A busca na *R-Tree* é similar à feita na *B-Tree*. Considere-se *E.I* como sendo o *MBR* referente à entrada *E*, e *E.p* referente a *Tid* ou *Cid*. Uma busca poderia ser apresentada da seguinte forma: encontrar todas as entradas cujos *MRB* coincidem com um intervalo espacial de busca *S*, a partir da *R-Tree T*. Os seguintes passos são executados:

1. Se *T* é um nó interno, percorra todas as suas entradas *E* para verificar se *E.I* faz interseção com *S*. Para todas as entradas que fazem interseção, execute este passo novamente, considerando *T* como sendo a subárvore apontada por *E.p*;
2. Se *T* é uma folha, percorra todas as entradas *E* para verificar se *E.I* faz interseção com *S*. Se as entradas fazem interseção, então compõem o conjunto resposta.

Inserção

Inserção na *R-Tree* é similar à feita na *B-Tree*, sendo que novas entradas são acrescentadas nas folhas, e os nós que ultrapassam a capacidade máxima de entradas, *M*, são particionados. O particionamento é, então, propagado para os pais. Os seguintes passos são executados para se incluir um objeto com *MBR O.I*:

1. Percorrer a árvore iniciada por *T* e, a cada nível, escolher o nó *E* cujo *E.I* necessite ser aumentado da menor extensão possível para englobar o novo objeto *O.I*;
2. Se várias entradas satisfazem este critério, Guttman [GuttmanA_84] sugere que seja escolhido o descendente com menor extensão total.
3. Ao se atingir o nó folha, tenta-se inserir o objeto. Se a inserção provocar um aumento na extensão total deste nó, os ajustes apropriados são feitos e propagados árvore acima;
4. Se não há espaço suficiente par se inserir o novo objeto na folha, ela é então particionada e as entradas são distribuídas entre as folhas nova e antiga;
5. Por fim, os intervalos são ajustados adequadamente e o particionamento é propagado árvore acima.

Remoção

Os seguintes passos são executados para se excluir o objeto *O* da árvore *T*:

1. Executar uma operação de busca *Exact Match Query* (ver seção 2.3.2.1) tentando encontrar o objeto;
2. Se for encontrado, o mesmo será removido.
3. Se a remoção não fizer com que o nó fique com menos do que a quantidade mínima de nós (*m*), é verificada se a extensão total do nó pode ser reduzida e, se puder, reduzir e propagar árvore acima;
4. Porém, se a remoção fizer com que o nó fique com menos do que a quantidade mínima de nós, o nó é copiado para um nó temporário e removido, propagando-se, em seguida, a remoção árvore acima;

5. Em seguida, as entradas órfãs do nó temporário são inseridas.

Vantagens e desvantagens

As suas principais vantagens são:

- não é dependente da seqüência dos dados de entrada e gera uma árvore balanceada;
- permite armazenamento secundário;
- suporta dados pontuais e espaciais;
- é dinâmica;
- suporta várias operações e
- é simples de ser implementada.

Como desvantagens, apresenta [SametH_90] [GaedeV_97]:

- não garante eficiência temporal, pois quando uma região de busca recai sobre um nó que possui muitas regiões sobrepostas, as subárvores sob esta região de busca têm que ser testadas incondicionalmente, fazendo com que buscas desnecessárias sejam feitas;
- degrada à medida em que a dimensão aumenta. Dentre outros motivos, porque aumenta a incidência de regiões sobrepostas e a estrutura do nó aumenta em função da dimensão, podendo gerar uma ineficiência quanto ao espaço necessário para armazenamento.

3.3.2 Outros *SAMs*

Algumas estruturas têm sido propostas para solucionar estes problemas, tais como:

- *R*-Tree* [BeckmannN_90]: basicamente corresponde à *R-Tree* com uma diferença no algoritmo que determina os critérios usados para definir como será feito o particionamento de um nó. Com este algoritmo, a *R*-Tree* consegue reduzir o número de regiões sobrepostas e melhorar o desempenho temporal. Porém, apresenta as outras desvantagens apresentadas pela *R-Tree*, não eliminando totalmente as regiões sobrepostas;

- *R^+-Tree* [SametH_90]: sua diferença principal está em "cortar" os objetos de forma a não ser mais permitida a sobreposição entre as regiões espaciais. Torna as buscas um pouco mais rápidas quando a árvore apresenta muitos níveis, pois diminui a necessidade de percorrer nós desnecessariamente em decorrência da inexistência de regiões sobrepostas. Porém, as remoções requerem buscas e outras remoções extras, pois o objeto está dividido em vários nós e precisa ser totalmente eliminado;
- *X-Tree* [Berchtold_96]: apresenta, também, um outro algoritmo para minimizar as regiões sobrepostas. Propõe a criação de nós, chamados super nós, que englobam aqueles que contém estas regiões, de forma a mantê-los o mais hierárquicos possível. Procura, também, evitar particionamentos nestes super nós, o que iria gerar uma maior quantidade de sobreposições. Trabalha bem com dimensões próximas de 100 e melhora a eficiência temporal das *R-Trees*, mas ainda apresenta problemas quanto ao armazenamento.

3.4 Conclusão

Este capítulo apresentou um resumo de alguns dos mais significativos *MIEs* existentes, com a intenção de exemplificar e complementar os conceitos apresentados no capítulo 2. Foi apresentado, também, um conjunto de características para classificar os MIE. Para tanto, foram traduzidos alguns dos termos usados para identificá-las e propostos outros que não foram encontrados na literatura estudada. Além disto, este conjunto agrupa características de forma mais abrangente do que os encontrados em [SametH_90] [KriegelH_90] [GaedeV_97].

Finalizando, durante a elaboração deste trabalho observou-se que o conceito em torno de alta ou baixa dimensionalidade não é padronizado. Ou seja, alguns autores colocam que alta dimensão corresponde a algo em torno de 5 a 10 [BeckmannN_90], ou 20-30 [PetrakisE_97], já outros consideram que as dimensões de nível mediano estão abaixo de 20 [WhiteD_96]. Outros consideram alta dimensão algo em torno de 100 [LinK_94] [WhiteD_96], sendo que alguns sequer especificam o que corresponde a alta dimensão. O que se observou, é que este conceito está diretamente relacionado com a aplicação que é dada aos MIE, e não à sua estrutura em si. Esta falta de consenso gera um problema com relação à avaliação dos mesmos, pois, por não

existir um parâmetro formalizado para indicar a ordem de grandeza da dimensão, torna-se necessário o entendimento completo das estruturas para identificar se a mesma é adequada ou não ao problema ao qual será aplicada.

4. CONCLUSÃO

4. CONCLUSÃO

Este trabalho mostrou como uma imagem é tratada no Modelo de dados SIRIUS através da definição de um tipo de dados **Imagem**. Associada a este tipo de dados estão três conceitos: Constante-Imagem, que são imagens exemplo usadas como base para indexar as imagens a serem armazenadas; Sumarizadores de Imagens, que são procedimentos computacionais utilizados para extrair características das imagens; e Parâmetros-da-Imagem, que corresponde ao resultado gerado por um sumarizador aplicado a uma imagem (vetores de características). Em seguida, foi mostrado que estes vetores de características poderiam ser vistos como coordenadas de um ponto situado no espaço. Estes pontos passariam a ser indexados, facilitando a consulta das imagens em função das características extraídas pelos sumarizadores.

A indexação de pontos no espaço é feita através dos chamados Métodos de Indexação Espaciais. Para entender o funcionamento destes métodos, no restante do trabalho foram introduzidos diversos conceitos em torno de dados espaciais e métodos de indexação espaciais. Foram apresentados os operadores espaciais mais comuns, exemplos de aplicações de dados espaciais e exemplos de métodos de indexação.

Como principal contribuição, o presente trabalho pode ser muito útil como base de consulta e texto introdutório para o estudo dos conceitos e aplicações de dados espaciais e como podem ser representados em meio computacional, bem como dos métodos de indexação utilizados para armazenar e recuperar este tipo de dados.

Referências Bibliográficas

REFERÊNCIAS BIBLIOGRÁFICAS

[AlcocerP_96] Alcocer, P.R.C.; Melo, C.P.; Furuie, S.S.; Bertozzo Jr., N.; Parzianello, L.C.; Rebelo, M.: "*O Projeto PACS: Um Sistema para Visualisação Dinâmica e Armazenamento de Imagens de Angiografia Digital no INCOR*", in Anais do 3º Fórum Nacional de Ciência e Técnologia em Saúde, Campos do Jordão - SP, Nro.2, pp. 695-696, Outubro 1996.

[ArefW_97] Aref, W.G.; Samet, H.: "*Efficient Window Block Retrieval in Quadtree-Based Spatial Databases*", GeoInformatica, Vol. 1, Nro. 1, pp. 59-91, Abril 1997 (also University of Maryland Computer Science TR 2866).

[BeckmannN_90] Beckmann, N.; Kriegel, H.-P.; Schneider, R.; Seeger, B.: "*R*-tree: An Efficient and Robust Access Method for Points and Rectangles*", Proceedings of the ACM SIGMOD International Conference on the Management of Data, pg. 322-331, Maio 1990.

[BekerM_97] Beker, M. F. - "*Representação de Áudio em Banco de Dados"*, Dissertação de Mestrado apresentada ao ICMSC-USP, Setembro de 1997.

[BentleyJ_79] Bentley, J.L.; Friedman, J.H.: "*Data Structures for Range Searching*", Computing Surveys 11(4):397—409, Dezembro 1979.

[Berchtold_96] Berchtold, S.; Keim, D.A.; Kriegel, H.-P.: "*The X-tree: An Index Structure for High-Dimensional Data*", Proceedings of the 23rd VLDB Conference, Mumbai (Bombay), India, 1996.

[BiajizM_96] Biajiz, Mauro: "*Representação De Modelos de Dados Orientados a Objetos Através da Parametrização de Abstrações de Dados*", Tese apresentada ao IFSC-USP para obtenção do título de Doutor, Setembro de 1996.

[BrinkhoffT_94] Brinkhoff, T.; Kriegel, H. P.; Schneider, R.; Seeger, B.: "*Multi-Step Processing of Spatial Joins"*, Proc. ACM - SIGMOD, pp. 197-208, Maio 1994.

[BrownL_92] Brown, L.G.: "*A Survey of Image Registration Techniques*", ACM Computing Surveys, Vol. 24, Nro. 4, pp. 325-376, Dezembro de 1992.

[CardenasA_93] Cardenas, A. F. et all: "*The knowledge-based Object Oriented PICQUERY+ Language*", IEEE Trans. on Knowledge and Data Engineering, Vol. 5, Nro. 4, pp. 644-656, Agosto 1993.

[ComerD_79] Comer, D.: "*The ubiquitous B-tree*", ACM Computing Surveys, Vol. 11, Nro. 2, pp. 121-137, Junho 1979.

[DuttonG_90] Dutton, G.F.: "*The Current State of PACS*", Applied Radiology, pp 15-19, Agosto 1990.

[ElmasriR_94] Elmasri, R.; Navathe, S.B.: "*Fundamentals of Database Systems*", Addison-Wesley Publishing Company, 2ª Edição, 1994.

[FlicknerM_95] Flickner, M. et all: "*Query by Image and Video Content: The QBIC System*", IEEE Computer, Vol. 28, Nro. 9, pp. 23-32, Setembro 1995.

[FreestonM_87] Freeston, M.: "*The BANG file: a new kind of grid file*", in Proceedings of the SIGMOD Conference, pp. 260-269, San Francisco, Maio 1987.

[FriedmanJ_77] Friedman, J.H.; Bentley, J.H.; Finkel, R.A.: "*An algorithm for finding best matches in logarithmic expected time*", ACM Transactions on Mathematical Software, Vol. 3, pp. 209-226, Setembro 1977.

[GaedeV_97] Gaede, V.; Günther, O.: "*Multidimensional Acess Methods*", a ser publicado em ACM Computing Surveys - http://www.wiwi.hu-berlin.de/~gaede/survey.ver.ps.Z, 1997.

[GudivadaV_95] Gudivada, V.N.; Raghavan, V.V.: "*Content-Based Image Retrieval Systems*", IEEE Computer, Vol. 28, Nro. 9, pp. 18-22, Setembro 1995.

[GuentherO_90] Guenther, O.; Buchmann, A.: "*Research Issues in Spatial Databases*", SIGMOD RECORD, Vol. 19, No. 4, Dezembro 1990.

[GuttmanA_84] Guttman, A.: "*R-trees: A Dynamic Index Structure For Spacial Searching*", Proceedings of the ACM SIGMOD International Conference on the Management of Data, pp. 47-57, Junho 1984.

[JagadishH_90] Jagadish, H.V.: "*Linear Clustering of Objects with Multiple Attributes*", in Proceedings of ACM SIGMOD, pp. 332-342, Atlantic City, Maio 1990.

[KnuthD_73] Knuth, D.: "*The Art of Computer Programming: sorting and searching*", Vol. 3, Addison-Wesley Publ. Co., Reading, Mass., 1973.

[KriegelH_90] Kriegel, H.-P.; Schiwietz, M.; Schneider, R.; Seeger, B.: "A *Performance Comparison of Multidimensional Point and Spatial Access Methods*", Poc. 1st Symp. On the Design of Large Spatial Databases, Santa Barbara CA, 1989. Lecture Notes in Computer Science, No. 409, Pub. Springer-Verlag, 1990.

[KriegelH_92] Kriegel, H.-P.; Horn, H.; Schiwiets, M.: "*The Performance of Object Decomposition Techiniques for Spatial Query Processing*", Final Report on the DFG Special Joint Initiantive: Data Structures and Efficient Algorithms. Lecture Notes in Computer Science, No. 594, Pub. Springer-Verlag, 1992.

[LinK_94] Lin, K.-I.; Jagadish, H.V.; Faloutsos, C.: "*The TV-tree - an index structure for high-dimensional data*", VLDB Journal, Vol. 3, pp. 517-542, Outubro 1994.

[NascimentoM_98] Nascimento, M.A.; Silva, J.R.O.: "*Towards Historical R-Trees*", Proceedings of the 1998 ACM Symposium on Applied Computing, pp. 235-240, 1998.

[NievergeltJ_84] Nievergelt, J.; Hinterberger, H.; Sevcik, K.C.: "*The grid file: an adaptable, symmetric multikey file structure*", Proceedings of the ACM Transactions on Database Systems, Vol. 9, pp. 38-71, Março 1984.

[OrensteinJ_86] Orenstein, J.A.: "*Spatial Query Processing in na Object Oriented Database System*", Proc. ACM SIGMOD'86, pp. 326-336, Washington, D.C., Maio 1986.

[PatelJ_96] Patel, J.M.; DeWitt, D. J.: "*Partition Based Spatial-Merge Join*", Proceedings of the ACM SIGMOD'96 Montreal, Canada, pp. 259-270, Junho 1996.

[PetrakisE_97] Petrakis, E.G.M.; Faloutsos, C.: "*Similarity Searching in Medical Image Databases*", IEEE Trans. On Knowledge and Data Engineering, Vol. 9, Nro. 3, pp. 435-447, Maio/Junho 1997.

[RoussopoulusN_95] Roussopoulus, N.; Kelley, S.; Vincent, F.: "*Nearest Neighbor Queries*", Proc. ACM - SIGMOD, pp. 71-79, Maio 1995.

[SametH_90] Samet, H.: "*The Design and Analysis of Spatial Data Structures*", Addison-Wesley, Reading, ISBN 0-201-50255-0, MA, 1990.

[SametH_95] Samet, H: "*Spatial Data Structures*", in Modern Database Systems: The Object Model Interoperability, and Beyond - W. Kim, Ed., Addison-Wesley/ACM Press, pp. 361-385, 1995.

[SantosR_96] Santos, R. R.; Traina, A.J.M.; Traina Jr., C.: "*Uma Linguagem de Definição e Recuperação de Imagens Baseada em Contúdo em uma Base de Dados Orientada a Objetos*", anais do XI Simpósio Brasileiro de Banco de Dados, pp. 127-142, São Carlos-SP, Outubro de 1996.

[SantosR_97] Santos, R. R. - "*Incorporação do Tipo de Dado Imagem em um Banco de Dados Orientado a Objetos*", Tese de Doutorado apresentada ao IFSC-USP, Novembro 1997.

[SchneiderM_97] Schneider, M.: "*Spatial Data Types for Database Systems: finite resolution geometry for geographic information systems*", Lecture Notes in Computer Science, Vol. 1288, Pub. Springer-Verlag, 1997.

[SclabassiR_96] Sclabassi, R.J.; Krieger, D.; Simon, R.; Lofink, R.; Gross, G.; DelLauder, D.M.: "*NeuroNet: Collaborative Itraoperative Guidance and Control*", IEEE Computer Graphics and Applications, (16) Nro. 1, pp. 39-45, 1996.

[SellisT_97] Sellis, T.; Roussopoulos, N.; Faloutsos, C.: "*Multidimensional Access Methods: Trees Have Grown Everywhere*", Proceedings of the 23rd VLDB Conference, Athens, Greece, pp. 13-14, Agosto 1997.

[SenzakoE_96] Senzako, E.Y.: "*SisMatch - Matching para Imagens de Tomografia por Ressonância Magnética*", Dissertação de mestrado, ICMSC-USP, Outubro de 1996.

[TamuraH_84] Tamura, H.; Yokoya, N.: "*Image Database Systems: A Survey*", Pattern Recognition, Vol. 17, Nro. 1, pp. 29-49, 1984.

[TenembaumA_96] Tenembaum, A.M., Langsam, Y.: "*Data Structures Using C And C++*", Second Edition - Prentice Hall, 1996.

[TrainaC_86] Traina Jr., C.: "*Máquina e Modelo de Dados Dedicados para Aplicações de Engenharia*", Tese apresentada ao IFQSC-USP para obtenção do título de Doutor, Dezembro 1986.

[TrainaC_97] Traina Jr., C.; Traina, A.J.M.; Santos, R. R.; Senzako, E. Y.: "*Content-Based Medical Image Retrieval in Object Oriented Databases*", in Journal of Informatica Medica Slovenica, Vol. 3, Nro. 12, pp. 67-72, Junho 1997.

[WhiteD_95] White, D.A.; Jain, R.: "*Similarity Indexing with the SS-tree*", IEEE 1995.

[WhiteD_96] White, D.A.; Jain, R.: "*Algorithms and Strategies for Similarity Retrieval*", Technical Report, VCL-96-101, Visual Computing Lab., Univ. California (http://vision.ucsd.edu/papers/simret/journ1.ps.gz), San Diego, Julho1996.

[ZhouX_97] Zhou, X.; Abel, D.J.; Truffet, D.: "*Data Partitioning for Parallel Spatial Join Processing*", Proceeedings of the 5th Intl. Symposium, SSD'97: Advances in Spatial Databases. Berlin, Germany, July 1997. Lecture Notes in Computer Science, No. 1262. Pub. Springer-Verlag, 1997.