Capítulo 2

Métodos de Acesso Multidimensionais e Sistemas de Informações Geográficas

Um método de acesso multidimensional é uma estrutura de indexação voltada ao suporte de objetos espaciais. Em geral, um método de acesso organiza os objetos através de uma estrutura hierárquica na forma de uma árvore de múltiplos caminhos (*multiway tree*), ou ainda na forma de uma estrutura *hash* ou uma combinação de ambas, sendo principalmente voltado à indexação direta de pontos e/ou retângulos. O principal objetivo de um método de acesso multidimensional (MAM) é proporcionar uma rápida obtenção dos objetos que satisfazem um certo relacionamento topológico, métrico ou direcional. Neste sentido, o espaço indexado é organizado de tal forma que, por exemplo, a recuperação dos objetos espaciais contidos em uma área particular requeira apenas o acesso aos objetos próximos a esta área, em oposição à análise do conjunto completo de objetos armazenados em memória secundária. Verifica-se, portanto, que MAMs minimizam o conjunto de objetos espaciais pesquisados, sendo que este conjunto é muito menor do que o total de objetos armazenados em um arquivo de dados, no caso de uma seleção espacial, e inferior ao produto cartesiano de dois conjuntos de dados, no caso de uma junção espacial. Pode-se dizer que um MAM é projetado como um caminho otimizado aos dados com base em um conjunto definido de predicados sobre os atributos, e seu uso melhora significativamente o desempenho de sistemas gerenciadores de banco de dados espaciais (SGBDEs) no processamento de consultas.

Um método de acesso multidimensional é definido através de: (1) uma estrutura de dados adaptada ao gerenciamento de objetos espaciais em memória secundária (disco); (2) algoritmos de pesquisa na estrutura que permitem a determinação de vários tipos de relacionamento, principalmente topológicos e métricos, tais como interseção, adjacência, “contém”, “está contido em” e “vizinho mais próximo”. Um MAM pode oferecer diversos algoritmos distintos, cada qual voltado para o suporte a um relacionamento espacial, ou oferecer um único algoritmo voltado para o suporte conjunto a vários relacionamentos, sendo que neste caso o algoritmo geralmente é baseado no relacionamento topológico de interseção e (3) algoritmos de alteração da estrutura de dados. Estes algoritmos são os responsáveis pela inserção de novos objetos espaciais no método de acesso, assim como pela remoção de objetos da estrutura de indexação. A operação de modificação, em particular, é comumente implementada pelos MAMs por intermédio de “uma remoção seguida de uma inserção”, com a geometria do objeto espacial indexado modificada entre as duas operações. Isto previne que a atualização da geometria dos objetos não degenere a organização hierárquica dos dados.

Métodos de acesso, alguma vezes, possuem uma extensão de sua estrutura em memória principal, ou seja, um *buffer-pool* dedicado. Um *buffer-pool* consiste de um bloco de RAM utilizado especificamente para armazenar de forma temporária páginas de disco relativas aos objetos indexados residentes em memória secundária. O armazenamento replicado de páginas objetiva reduzir o número de acessos a memória secundária e consequentemente melhorar o desempenho de métodos de acesso multidimensionais. Um método de acesso que usa um *buffer-pool* como extensão de sua estrutura é denominado de MAM virtual. Atualmente, a implementação física de MAMs é comumente baseada em MAM virtual.

2.1 Tipos de Dados Espaciais

Atualmente existe uma grande variedade de aplicações que necessitam armazenar e recuperar informações relativas aos objetos contidos em algum espaço multidimensional. Como exemplo, pode-se citar: aplicações médicas de modelagem de membros humanos, aplicações *VLSI CAD* e aplicações georeferenciadas. O gerenciamento destes objetos é mais complexo do que o efetuado por tradicionais sistemas gerenciadores de banco de dados (SGBDs), uma vez que torna-se necessário o tratamento de novos tipos de dado, responsáveis por descrever a geometria de objetos espaciais a partir de um conjunto de coordenadas, os quais são conhecidos como tipos de dados espaciais (TDEs). A seguir são descritos os principais TDEs oferecidos por sistemas gerenciadores de banco de dados espaciais (SGBDEs) e por sistemas de informações geográficas (SIGs).

Um **ponto** é a menor unidade possível para representar um objeto espacial. Um objeto espacial formado por apenas um único ponto não possui extensão. Uma **linha** é uma seqüência de pontos conectados retilineamente, ao passo que em uma **linha poligonal** os pontos não estão dispostos de forma retilínea. Para ambas, linhas e linhas poligonais, cada par de pontos conectados corresponde a um **segmento de linha**. Um **polígono** é formado por uma seqüência de linhas e/ou linhas poligonais, sendo que esta seqüência é fechada, isto é, o primeiro ponto coincide com o último. Polígonos e linhas poligonais fechadas são funcionalmente equivalentes, diferindo somente, em alguns SIGs e SGBDEs, quanto ao armazenamento explícito de atributos convencionais derivados de polígonos, como exemplo área e perímetro. **Polígonos complexos**, por sua vez, podem permitir buracos e/ou consistir de diversas partes disjuntas. Já um **poliedro** é limitado por quatro ou mais polígonos, denominados faces, sendo que as interseções das faces formam as arestas e as interseções das arestas formam os vértices. Os TDEs pontos, linhas (e/ou poligonais), polígonos (e/ ou complexos) e poliedros são estruturas 0-, 1-, 2- e 3-dimensionais, respectivamente.

Em geral, (1) pontos são usados para representar localizações discretas como uma cidade em um mapa; (2) linhas e linhas poligonais abertas são utilizadas para representar objetos espaciais lineares, tais como rios, estradas, ferrovias e redes de infra-estrutura; (3) polígonos (simples), polígonos complexos e linhas poligonais fechadas são utilizados para representar objetos bidimensionais, como exemplo a área ocupada por um bairro, cidade, região ou país e por fim (4) poliedros são utilizados para representar sólidos no espaço, tal como uma peça mecânica ou um membro humano. A figura 2.1 ilustra os vários TDEs.

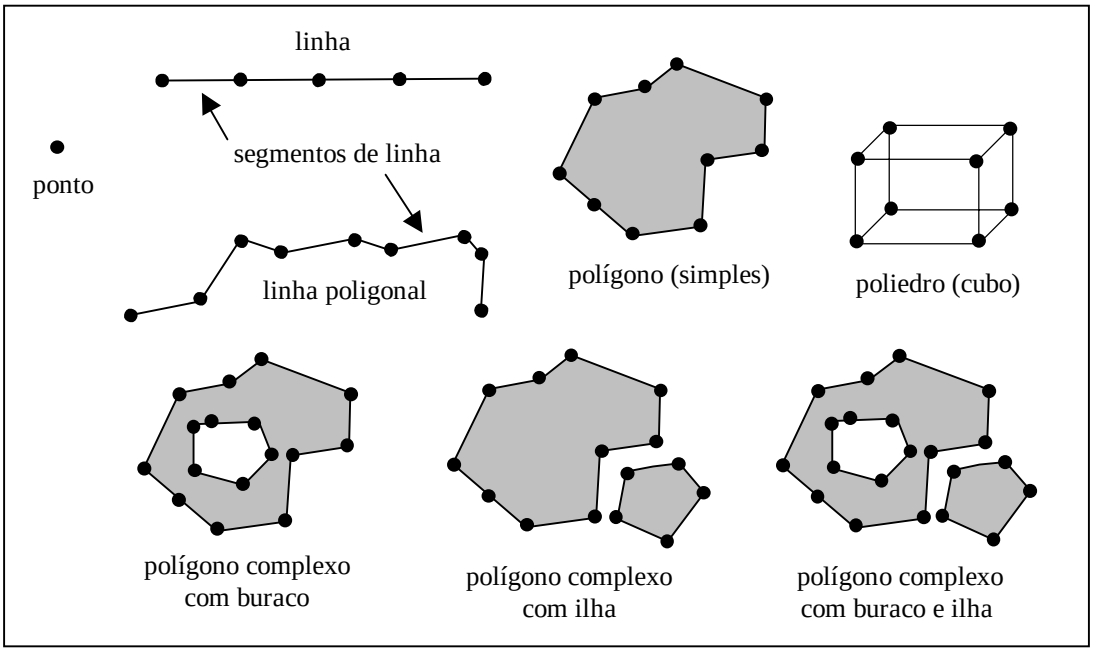

Figura 2.1 Tipos de dados espaciais (TDEs)

Um objeto espacial ***o*** tem associado a si pelo menos um atributo que define a sua extensão e localização no espaço Euclidiano *d-dimensional* E^d ou em algum subespaço deste. Este atributo, denominado **atributo espacial** e representado por ***o.G***, consiste de um conjunto de coordenadas (*x*, *y*) ou (*x*, *y*, *z*) referentes a algum TDE. Usaremos a notação $o.G^{\circ}$, $o.\partial G$ e $o.G^{-}$ para denotar, respectivamente, o interior, a fronteira e o exterior de ***o.G***.

Um objeto espacial pode naturalmente possuir mais de um atributo espacial e por conseguinte múltiplas representações [MCG99, CCH+96]. Em geral, uma representação refere-se a um nível de abstração específico. Como exemplo, um objeto espacial **cidade** pode ter dois atributos espaciais, sendo que um destes representa um nível de abstração mais detalhado (tal como, o limite da área urbana), o qual é suportado por um TDE polígono, enquanto o outro atributo representa um alto nível de abstração (tal como, a localização central ou o marco zero da cidade), que é suportado por um TDE ponto. Ademais, pode-se prover múltiplas representações para um objeto espacial que possua apenas um único atributo espacial. Neste caso, as demais representações são derivadas dinamicamente a partir de alguma computação feita sobre a representação armazenada, como exemplo, através do cálculo da centróide de um retângulo. O gerenciamento de múltiplas representações, no entanto, introduz vários problemas, tais como redundância e inconsistência dos dados.

A determinação de relacionamentos topológicos para objetos que possuem mais de um atributo espacial pode ser efetuada: (1) automaticamente com base no atributo que possui o nível de abstração mais detalhado; (2) através de um atributo espacial específico definido pelo usuário na formulação da consulta ou (3) a partir de um atributo espacial pré-selecionado pelo projetista do banco de dados no momento da criação do arquivo de dados (ou tabela) relativo ao objeto espacial. A estratégia escolhida varia de sistema para sistema, sendo altamente dependente do poder de expressão da linguagem de consulta e de características internas do módulo de processamento de consultas. Neste contexto, o

atributo espacial representado por ***o.G*** e usado na definição formal de diversas consultas na seção 2.5, poderá ser escolhido de acordo com qualquer uma das estratégias citadas, sendo requerido, entretanto, que este atributo tenha um MAM associado. Na indexação de um arquivo através de múltiplos atributos espaciais deve-se contrastar os custos de armazenamento e atualização com os ganhos proporcionados na eficiência das consultas.

2.2 Estratégias de Representação dos Dados

O armazenamento de um objeto espacial requer, no caso específico de objetos espaciais de dimensão não-zero (linhas, polígonos, poliedros, dentre outros), a alocação de uma grande porção de espaço em memória secundária para acomodar os diversos pares de coordenadas que descrevem a geometria exata do objeto. Aplicações georeferenciadas, por exemplo, armazenam comumente polígonos contendo uma complexidade de 50, 100 e até 1.000 pontos (vértices). O armazenamento da geometria exata de um objeto espacial não é apropriada para métodos de acesso multidimensionais, quando o objeto possui uma alta complexidade, desde que isto diminui o *fan-out* da estrutura. O *fan-out* especifica o número de filhos ou apontadores de um dado nó. O pequeno número de entradas por nó, causado pelo armazenamento de uma grande quantidade de *bytes* por objeto espacial, reduz o *fan-out*, tornando a árvore mais profunda (grande número de níveis) do que rasa. Idealmente, um árvore deve ser "rasa e larga", ou seja, deve possuir poucos níveis e um grande número de nós em cada um dos níveis. Métodos de acesso multidimensionais com baixo *fan-out* requerem mais acessos a disco no suporte às consultas espaciais e às operações de alteração. Vale destacar que, em geral, os nós folhas e internos da estrutura de dados estão diretamente relacionados com o tamanho da página de disco, ou seja, um nó corresponde a uma página de disco.

Deve-se diferenciar, portanto, entre a estratégia de representação utilizada para o armazenamento de objetos espaciais em memória secundária e a estratégia de representação usada para o armazenamento de objetos espaciais no método de acesso multidimensional. Segundo Cox Júnior [Cox91], a ênfase das duas estratégias de representação são opostas, pois enquanto a primeira estratégia preocupa-se em representar o objeto através da sua decomposição (especialização), de modo que possa ser manipulado eficientemente sem qualquer perda de precisão (geometria exata), na estratégia de representação utilizada por métodos de acesso há a preocupação em abstrair (generalizar) os objetos através da representação destes por formas geométricas mais simples, visando diminuir os requisitos de armazenamento e o custo para se determinar a satisfação de relacionamentos métricos topológicos e direcionais.

Assim, métodos de acesso multidimensionais utilizam abstrações (ou aproximações) para representar a geometria de objetos espaciais. Uma aproximação, entretanto, deve ser conservativa. Isto ocorre se, e somente se, cada ponto da geometria do objeto espacial também estiver contido na geometria da aproximação. Uma aproximação conservativa preserva as propriedades geométricas do objeto espacial, isto é, a localização e a extensão no espaço. A abstração mais utilizada pelos MAMs é o MBB (*minimum bounding box*), também conhecido por "retângulo envolvente mínimo" e por MBR (*minimum bounding*

rectangle). O MBB consiste do menor retângulo *d-dimensional* com lados paralelos aos eixos que contém completamente o objeto espacial. Além do MBB ser uma aproximação conservativa, o armazenamento deste requer somente alguns poucos *bytes* referentes a quatro coordenadas (intervalos $I_x = [x_L, x_U]$ e $I_y = [y_L, y_U]$). Dado um objeto ***o*** formado por uma seqüência de pontos $p_1, ..., p_n$, sendo $p_i = (x_i, y_i)$ para $1 \leq i \leq n$, o respectivo MBB corresponde a: (1) x_L = mínimo x_i com $1 \leq i \leq n$; (2) y_L = mínimo y_i com $1 \leq i \leq n$; (3) x_U = máximo x_i com $1 \leq i \leq n$ e (4) y_U = máximo y_i com $1 \leq i \leq n$. Ademais, o MBB facilita muito a avaliação de predicados espaciais, devido às propriedades de retângulos.

Outros tipos de aproximação conservativa, conforme destacado por Brinkhoff *et al.* [BKSS94, BK94] são: o retângulo envolvente mínimo rotacionado (*rotated minimum bounding rectangle – RMBR*), o círculo envolvente mínimo (*minimum bounding circle – MBC*), o casco convexo (*convex hull – CH*), o polígono envolvente mínimo com *m* vértices (*minimum bounding m-corner – m-C*) e a elipse envolvente mínima (*minimum bounding ellipse – MBE*). As aproximações *RMBR*, *MBC*, *CH*, *m-C* e *MBE*, requerem, respectivamente, o armazenamento de cinco, três, *n* (variável), 2 * *m* e 5 coordenadas. A figura 2.2 ilustra os diversos tipos de aproximação conservativa.

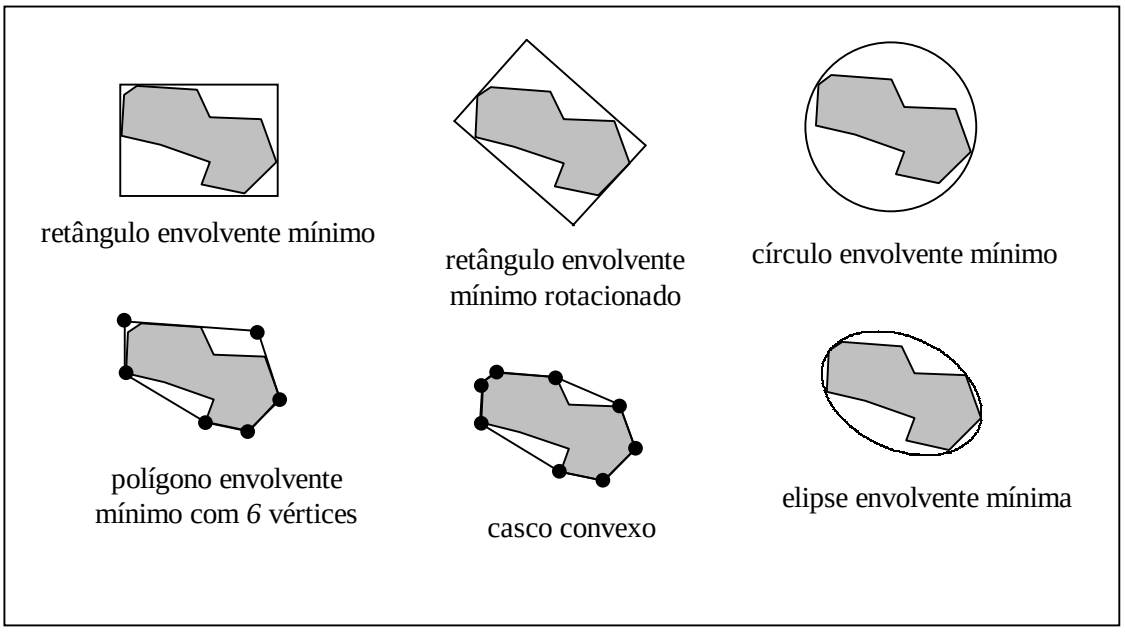

Figura 2.2 Aproximações conservativas

2.3 Fases de Filtragem e Refinamento

A utilização de formas geométricas mais simples para representar a geometria de objetos espaciais, em especial, o uso de aproximações, implica em uma certa perda de precisão na representação da geometria. Esta perda de precisão determina a formação de uma área vazia, ou seja, um espaço contido dentro da aproximação que não faz parte da geometria do objeto espacial. Tal área é conhecida como área de *dead space* (figura 2.3). Quando o relacionamento espacial é determinado com base na área de *dead space*, o objeto espacial recuperado constitui um falso candidato, uma vez que este não satisfaz o relacionamento espacial. Por outro lado, a utilização de aproximações garante que nenhum dos objetos espaciais que satisfaz um certo relacionamento espacial seja desconsiderado na resposta da consulta. Isto é garantido pela propriedade conservativa das aproximações (se um objeto intersecta uma janela de consulta, a aproximação do objeto também intersecta a janela de consulta, por exemplo).

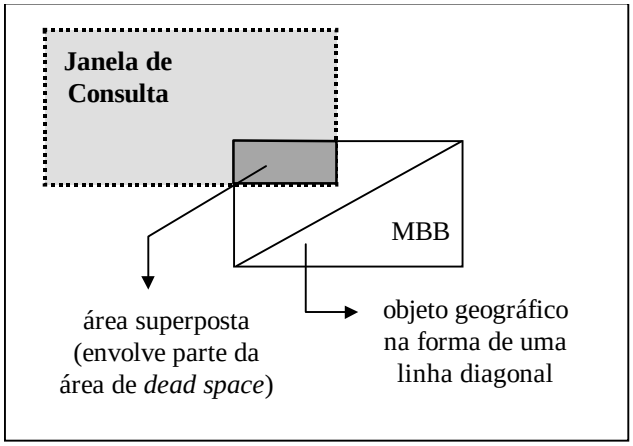

Figura 2.3 Recuperação de um falso candidato (*dead space*)

Deste modo, pode-se dizer que métodos de acessos são imprecisos, no sentido que estes não retornam a resposta final e exata das consultas, retornando ao invés disto um superconjunto de candidatos. Para obter a resposta final, o processamento de consultas espaciais usando MAMs é efetuado em duas fases:

- **fase de filtragem:** esta fase apenas filtra (ou descarta) os objetos espaciais que certamente não satisfazem um determinado relacionamento espacial. A determinação do relacionamento é feita com base em aproximações, gerando-se verdadeiros e falsos candidatos (superconjunto de candidatos). A fase de filtragem é pouco custosa, pois manipula formas geométricas mais simples, diminuindo assim o custo da transferência de dados para a memória principal e o custo para se determinar a satisfação de um relacionamento espacial;
- **fase de refinamento:** devido à presença de falsos candidatos no conjunto de objetos selecionados, é necessário a verificação, para cada objeto candidato, se o relacionamento espacial é satisfeito com relação à geometria exata do objeto espacial, sendo por ventura descartados quaisquer falsos candidatos. A fase de refinamento é altamente custosa, pois requer tanto o acesso à geometria exata dos objetos espaciais, e portanto a transferência para memória principal de uma

quantidade relativamente grande de dados, quanto a realização de cálculos geométricos complexos para se determinar a satisfação do relacionamento espacial.

O ganho de desempenho proporcionado pela utilização de aproximações é baseado nas seguintes premissas: (1) a fase de refinamento é extremamente custosa e deve ser aplicada a uma quantidade muito reduzida de objetos espaciais e (2) a fase de filtragem é várias vezes menos custosa do que a fase de refinamento e assim, deve ser usada para restringir a quantidade de objetos espaciais que serão analisados na fase de refinamento (superconjunto de candidatos). Desta forma, a fase de filtragem serve para diminuir o custo da fase posterior de refinamento, o qual basicamente determina o desempenho no suporte a uma consulta espacial. Há algum ganho de desempenho significativo, no entanto, apenas se a fase de filtragem realmente reduzir drasticamente a quantidade de objetos a serem analisados na fase de refinamento, ou seja, se o grau de seletividade for baixo. Em especial, para consultas do tipo *containment range query* não é necessário a realização da fase de refinamento. Neste caso, a resposta final e exata da consulta pode ser obtida diretamente a partir da fase de filtragem. A figura 2.4 ilustra os processos de filtragem e refinamento.

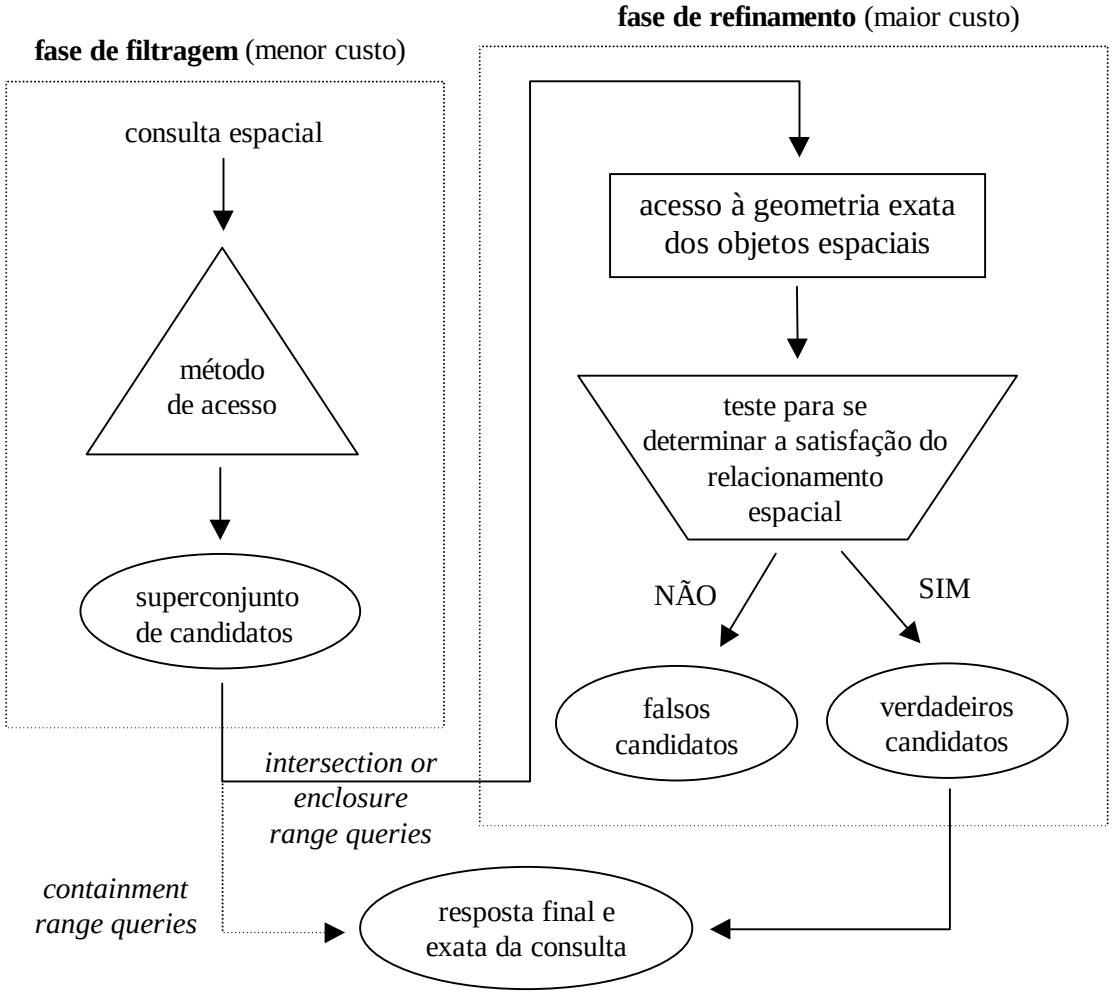

Figura 2.4 Fases de filtragem e refinamento

Como destacado anteriormente, deve-se restringir a quantidade de objetos espaciais que serão analisados na fase de refinamento. Para isto, Brinkhoff *et al.* [BKSS94, BK94] propõem a utilização de outras aproximações conservativas mais precisas, tais como *RMBR*, *CH*, *m-C* e *MBE*, as quais foram descritas na seção 2.2. Embora tais abstrações diminuam a área de *dead space*, o seu uso aumenta os requisitos de armazenamento e o custo para se determinar a satisfação de relacionamentos espaciais. Por outro lado, métodos de acesso multidimensionais baseados nestas aproximações podem conduzir a um conjunto mais restrito de objetos candidatos, que por sua vez servirá para diminuir o custo da fase de refinamento, o que consequentemente pode proporcionar um melhor desempenho no suporte às consultas espaciais. Outra opção, é o armazenamento de duas aproximações para cada objeto indexado, um MBB e uma outra aproximação mais precisa. Esta última aproximação é usada em uma nova fase no processamento de consultas entre as fases de filtragem e refinamento, chamada fase de aproximação. Nesta fase, cada objeto candidato recuperado na fase de filtragem através do uso do MBB é testado, para se verificar a satisfação do relacionamento espacial, mas usando desta vez a aproximação mais precisa. Somente os objetos candidatos que satisfizerem novamente o relacionamento espacial é que são analisados posteriormente na fase de refinamento.

Uma aproximação progressiva constitui em um tipo alternativo de aproximação, na qual a aproximação de um objeto é um subconjunto de seus pontos, ou seja, um objeto é progressivamente aproximado se, e somente se, o conjunto de pontos da aproximação for um subconjunto dos pontos do objeto [Car98, Gat00]. Este tipo de aproximação deve ser utilizado imediatamente antes da fase de refinamento. Uma vez que, por exemplo, a interseção entre uma aproximação progressiva e uma janela de consulta garante que o objeto espacial também intersecte a janela de consulta, uma aproximação progressiva pode ser usada para detectar verdadeiros candidatos. Assim, somente os objetos que não forem detectados através da utilização da aproximação progressiva é que devem ter a sua geometria exata recuperada e verificada quanto à satisfação do relacionamento espacial. O cálculo de uma aproximação progressiva, no entanto, é muito mais custoso do que o cálculo de uma aproximação conservativa, especialmente quando almeja-se a aproximação progressiva máxima. Alguns exemplos de aproximações deste tipo são: o retângulo inscrito máximo (*maximum enclosed rectangle* – MER), o círculo inscrito máximo (*maximum enclosed circle* – MEC), os segmentos de linha inscritos máximos (*maximum enclosed line segments* – EL) e uma combinação de MER e EL (figura 2.5).

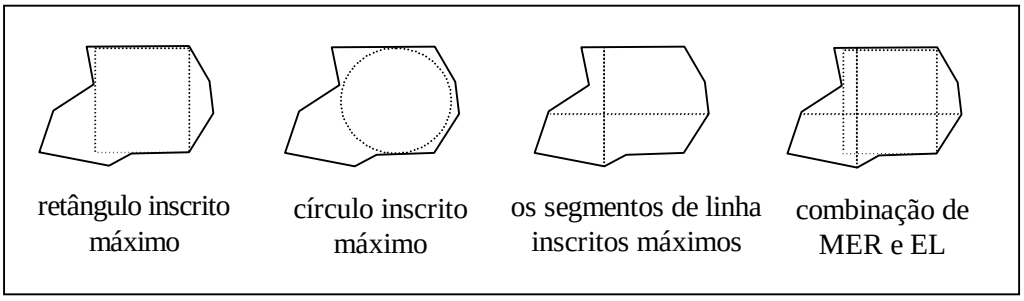

Figura 2.5 Aproximações progressivas

Outras variações de aproximações conservativas e progressivas existem, tal como a aproximação por retângulos envolventes mínimos decompostos (*decomposed minimum*

bounding rectangles – DMBR) [LLRC96]. Apesar disto, a aproximação MBB continua sendo a mais popular, devido a sua simplicidade e eficiência.

2.4 Propriedades Ideais de um Método de Acesso Multidimensional

Segundo Gaede e Günther [GG98], há uma grande variedade de requisitos relacionados às características dos dados e das aplicações espaciais que um MAM idealmente deveria atender. Assim, espera-se que um método de acesso possua as seguintes propriedades:

- **um MAM deve ser dinâmico:** uma vez que objetos são inseridos e removidos do banco de dados segundo uma ordem arbitrária, métodos de acesso devem continuamente manter informações sobre as várias alterações nos dados. Ademais, um método de acesso dinâmico deve permitir que operações de inserção e remoção sejam intercaladas com consultas espaciais sem que haja a necessidade de qualquer reorganização periódica de sua estrutura de dados;
- **um MAM deve oferecer suporte para o armazenamento secundário e terciário:** apesar do incrível crescimento da capacidade de armazenamento de dados em memória principal, em geral não é possível armazenar completamente o banco de dados em RAM, uma vez que os requisitos de armazenamento atuais são grandes e também crescem ao longo do tempo. Portanto, métodos de acesso multidimensionais devem também oferecer, de uma forma homogênea, suporte para o armazenamento secundário e terciário;
- **um MAM deve atender a um grande número de operações:** um método de acesso deve prover suporte para um grande número de consultas espaciais e para todas as possíveis operações de alteração (inserção, remoção e modificação de dados). Além disto, métodos de acesso não devem prover um suporte eficiente apenas para um tipo de operação particular (tal como, pesquisa), a um custo proibitivo para outras operações (tal como, inserção);
- **um MAM deve ter um bom desempenho independentemente das características dos dados:** o desempenho de métodos de acesso multidimensionais não deve ser degradado, por exemplo, em função de distribuições não-uniformes altamente assimétricas ou em função da seqüência de inserção dos dados. Esta propriedade é fundamental, desde que os dados encontram-se distribuídos de uma forma particular para cada uma das dimensões;
- **um MAM deve ser simples:** métodos de acesso com uma organização extremamente elaborada contendo inúmeras exceções freqüentemente conduzem a algum erro de implementação e portanto não são suficientemente robustos para serem usados em aplicações de grande porte;
- **um MAM deve ser adaptável à escalabilidade dos dados:** métodos de acesso devem adaptar-se a crescentes volumes de dados, tal como 100.000, 1.000.000 ou até mesmo 1.000.000.000 de objetos espaciais;

- **um MAM deve ser eficiente quanto à complexidade de tempo:** pesquisas na estrutura devem ser rápidas, tais como consultas do tipo *point query*, *intersection range query* e *nearest neighbor query*. A principal meta do projeto de MAMs é alcançar os níveis de desempenho de uma **B-tree**. Em primeiro lugar, métodos de acesso multidimensionais devem garantir, no pior caso, um desempenho logarítmico para consultas espaciais, considerando todas as possíveis distribuições de dados e seqüências de inserção dos dados. Em segundo lugar, deve-se garantir o mesmo desempenho mínimo no pior caso para qualquer número de dimensões do espaço;
- **um MAM deve ser eficiente quanto à complexidade de espaço:** Um mecanismo de indexação espacial deve ser pequeno em volume quando comparado com o volume de dados do arquivo indexado. Assim, um MAM deve garantir uma taxa mínima de ocupação dos nós (páginas de disco);
- **um MAM deve integrar mecanismos de controle de concorrência e de recuperação de falhas:** para sistemas de banco de dados modernos, nos quais inúmeros usuários concorrentemente inserem, modificam, removem e recuperam objetos do banco de dados, um método de acesso multidimensional deve prover técnicas robustas para gerenciamento de transações, sem que isto implique em uma grande queda no desempenho;
- **um MAM deve gerar um mínimo impacto no sistema:** a integração de um método de acesso em um sistema gerenciador de banco de dados espacial deve gerar o menor impacto possível nos outros componentes do sistema;

2.5 Tipos e Subtipos de Consulta Espacial

Esta seção tem por objetivo caracterizar, de modo formal e informal, os diversos tipos e subtipos de consulta espacial para os quais um método de acesso multidimensional deve prover algum tipo de suporte eficiente.

2.5.1 Exact Match Query

Esta consulta também é conhecida na literatura como *object query* e *member query* [Güt94, GG98]. Dado um objeto ***o'*** com atributo espacial $o'.G \subseteq E^d$, encontre todos os objetos ***o*** que possuam as mesmas coordenadas que ***o'***, ou seja, que possuam a mesma extensão e localização no espaço.

$$EMQ\ (o', dataset) = \{\ o \mid o \in dataset \wedge o.G = o'.G\ \}$$

A execução deste tipo de consulta espacial pode ser realizada em três situações distintas, dependendo do conteúdo do arquivo de dados *dataset*. Por exemplo, considere objetos representados por pontos, tais como postes, poços artesianos e edificações de destaque. A primeira situação consiste da busca de um ponto inexistente no arquivo de dados. Neste caso, o método de acesso multidimensional deve ser capaz de descobrir tal inexistência sem ter que percorrer parte significativa de sua estrutura de dados. Já a segunda

situação acontece quando o arquivo de dados não contém duplicatas (objetos espaciais com coordenadas idênticas) e a pesquisa é efetuada para um ponto existente. Como resultado, é recuperado um único ponto dentre o universo de pontos armazenados no arquivo de dados. A última situação é uma variante da anterior, quando o arquivo de dados contém duplicatas. Nesta situação, podem ser recuperados vários objetos, sendo testada a habilidade do método de acesso em agrupar duplicatas. O armazenamento em separado de duplicatas conduz a um maior número de acessos a disco e por conseguinte em uma queda no desempenho.

2.5.2 Partial Match Query

Dado um objeto ***o'*** com atributo espacial $o'.G \subseteq E^d$, encontre todos os objetos ***o*** que possuam, em uma dada dimensão, a mesma extensão e localização no espaço que ***o'***.

$$PMQ\,(o', dataset, dim) = \{\, o \mid o \in dataset \wedge [min(o.G.dim), max(o.G.dim)] = [min(o'.G.dim), max(o'.G.dim)] \,\}$$

sendo que $o.G = \{c_1, \ldots, c_n\}$ **onde** $c_i = (d_1, \ldots, d_m)$ **para** $1 \le i \le n$ **e** $m \ge 1$

$o.G.dim = \{c_1', \ldots, c_n'\}$ **onde** $c_i' = (d_{dim})$ **para** $1 \le i \le n$ **e** $1 \le dim \le m$

Para objetos espaciais bidimensionais, tal como um polígono, esta consulta espacial determina uma faixa perpendicular com relação à dimensão escolhida, segundo a qual os objetos devem ter suas respectivas extensão e localização limitadas. Para pontos, esta faixa torna-se uma reta (figura 2.6). Como exemplo, para um conjunto de cidades representadas por pontos, encontrar as cidades que possuam a mesma longitude de Londres.

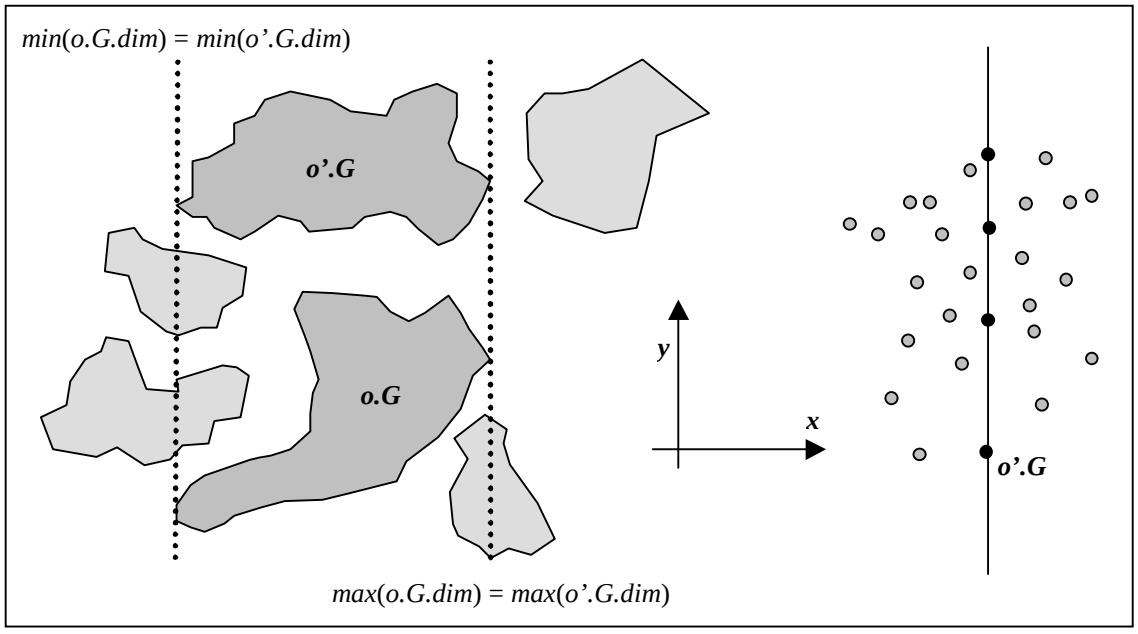

Figura 2.6 Exemplos de *partial match queries* para a dimensão *x*

2.5.3 Point Query

Esta consulta também é conhecida na literatura como *stabbing query* [KF92]. Dado um ponto $\boldsymbol{p} \in E^d$, encontre todos os objetos ***o*** que sobrepõem ***p***.

$$PQ\ (p, dataset) = \{\ o \mid o \in dataset \wedge o.G \cap p = p\}$$

Este tipo de consulta espacial pode ser considerado como um caso especial de outros subtipos de consulta, como exemplo *intersection range query*, *intersection region query*, *enclosure range query* e *enclosure region query* (estes subtipos são especificados no texto posteriormente), para as quais a janela de consulta ou a região de pesquisa se reduz a um ponto. Para arquivos de dados do tipo ponto, esta consulta é equivalente a *exact match query*. A figura 2.7 ilustra um exemplo para um arquivo de dados do tipo polígono. Neste caso, quer-se determinar em qual estado a cidade do Recife se localiza.

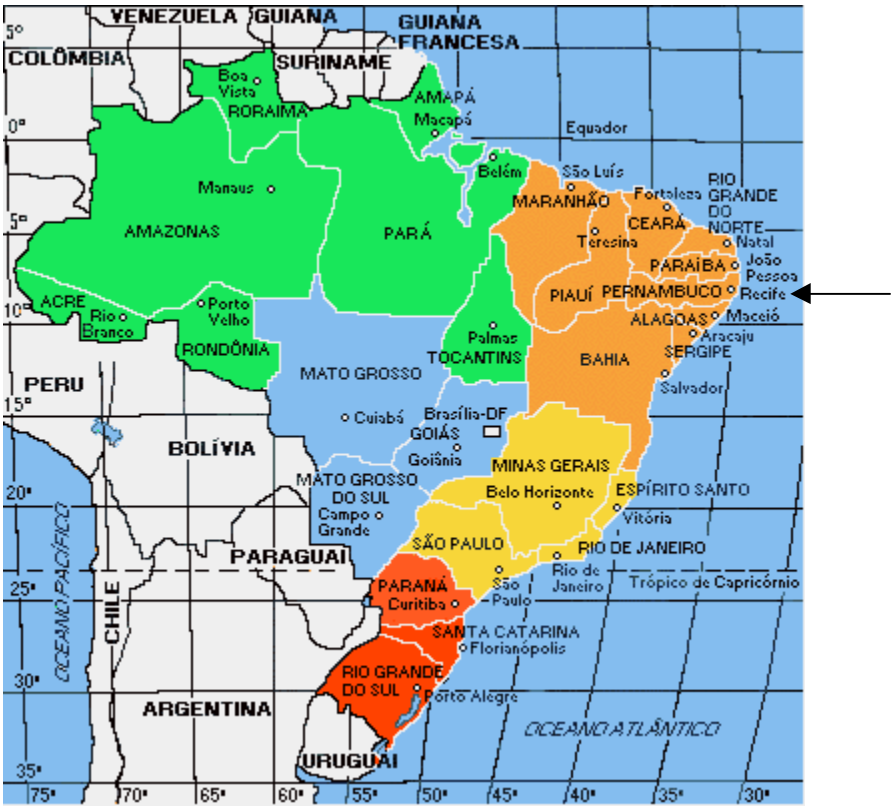

Figura 2.7 Exemplo de *point query*

2.5.4 Range Query

Esta consulta também é conhecida na literatura como *window query* [GG98] e *rectangle query* [KSSS89]. Dado um retângulo *d-dimensional* $\boldsymbol{R} \subseteq E^d$ cujos lados são paralelos aos eixos de suas respectivas dimensões (*iso-oriented*), encontre todos os objetos ***o*** que satisfaçam um certo relacionamento topológico com relação ao retângulo *d-dimensional* ***R*** (θ (***o.G***, ***R***) = {"intersecta", "contém", "está contido em"}). Cada tipo de relacionamento topológico caracteriza um dos subtipos específicos de *range query*, que são descritos

posteriormente. O retângulo *d-dimensional* ***R*** é conhecido principalmente pelo termo "janela de consulta", mas também como "retângulo de consulta" e "retângulo de busca", sendo este representado por um conjunto de intervalos *d-dimensionais* fechados $I = \{[l_1, u_1], ... [l_k, u_k]\}$, onde $[l_i, u_i]$ descreve o *extent* ao longo da dimensão *i*, para $1 \leq i \leq k$ (figura 2.8).

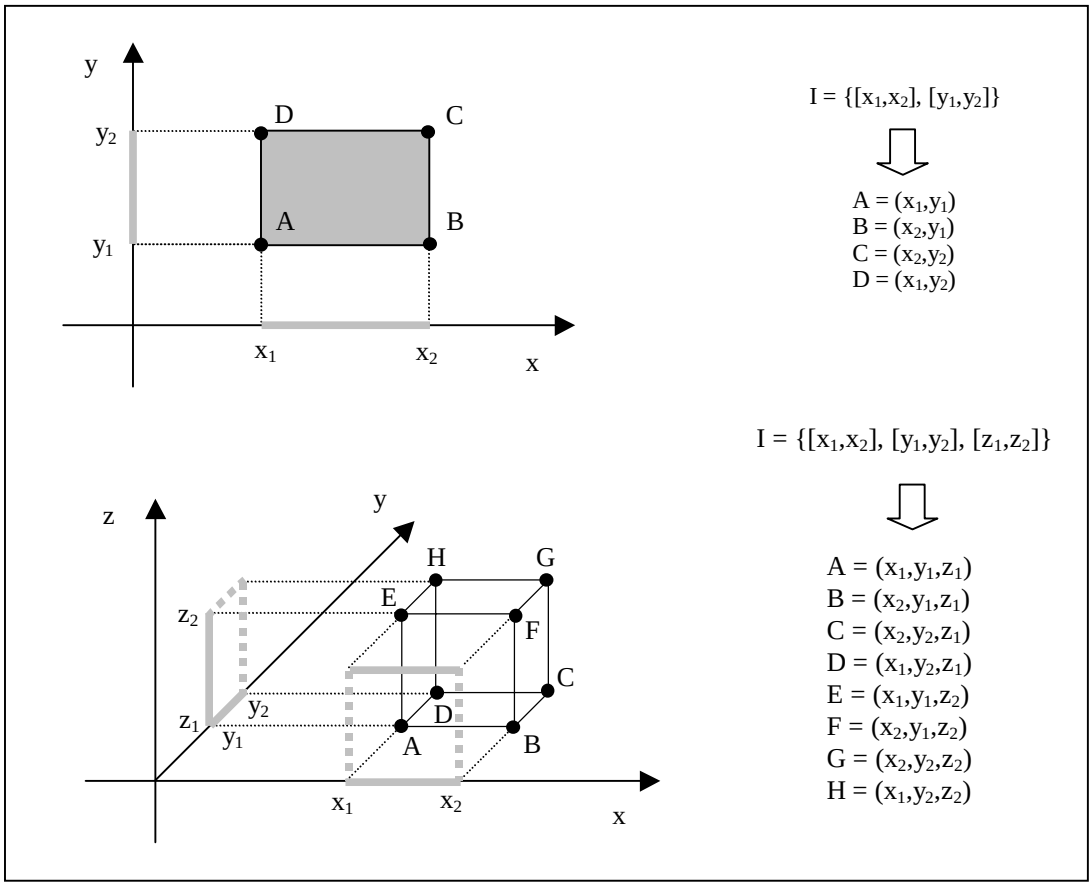

Figura 2.8 Retângulo *R* e o conjunto de intervalos *d-dimensionais* fechados *I*

2.5.4.1 Intersection Range Query

Esta consulta também é conhecida na literatura como *rectangle intersection query*, *range query* para determinação de interseção ou simplesmente *range query* [KSSS89, BKSS90, Cox91, See91, Car98, GG98], a qual consiste de um subtipo de *range query* com relacionamento topológico θ (***o.G***, ***R***) de interseção. Desta maneira, dado um retângulo *d-dimensional iso-oriented* $\boldsymbol{R} \subseteq E^d$, encontre todos os objetos ***o*** que tenham pelo menos um ponto em comum com ***R***.

$$\boldsymbol{IRaQ\ (R, dataset) = \{\ o \mid o \in dataset \wedge o.G \cap R \neq \emptyset\}}$$

Como exemplo de uma consulta espacial elaborada por um usuário final, pode-se citar: "Dada uma região retangular destinada para a construção de um novo aeroporto, identifique todas as propriedades que terão terras desapropriadas" (figura 2.9). Para arquivos de dados do tipo ponto, este subtipo de consulta é equivalente a *containment range query*. Segundo

Frederick [Fre99], este subtipo de consulta também é usado com a finalidade de fazer o *clipping* de objetos na visualização de objetos gráficos na tela.

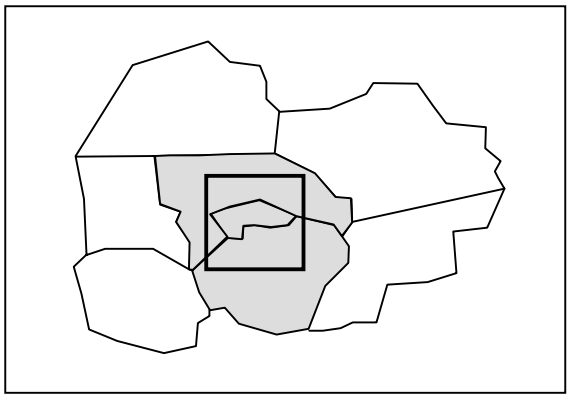

Figura 2.9 Exemplo de *intersection range query*

2.5.4.2 Containment Range Query

Esta consulta também é conhecida por *rectangle containment query*, *range query* para determinação dos objetos espaciais contidos na janela de consulta, *range query* para determinação de não-inclusão ou simplesmente *range query* [KSSS89, Cox91, See91, Güt94, Car98], a qual consiste de um subtipo de *range query* com relacionamento topológico θ (***o.G***, ***R***) do tipo "está contido em". Desta maneira, dado um retângulo *d-dimensional iso-oriented* $\boldsymbol{R} \subseteq E^d$, encontre todos os objetos ***o*** contidos em ***R***.

$$\boldsymbol{CRaQ\ (R, dataset) = \{\ o \mid o \in dataset \wedge o.G \cap R = o.G\ \} \cong \{\ o \mid o \in dataset \wedge o.G \subseteq R\ \}}$$

A figura 2.10 ilustra um exemplo para um arquivo de dados do tipo ponto. Neste caso, dado uma região retangular, quer-se identificar todos os focos de dengue nesta região. Já para o exemplo ilustrado na figura 2.9, este foi adaptado para identificar as propriedades que terão **todas** as suas terras desapropriadas (figura 2.11). Para arquivos de dados do tipo ponto, esta consulta é equivalente a *intersection range query.*

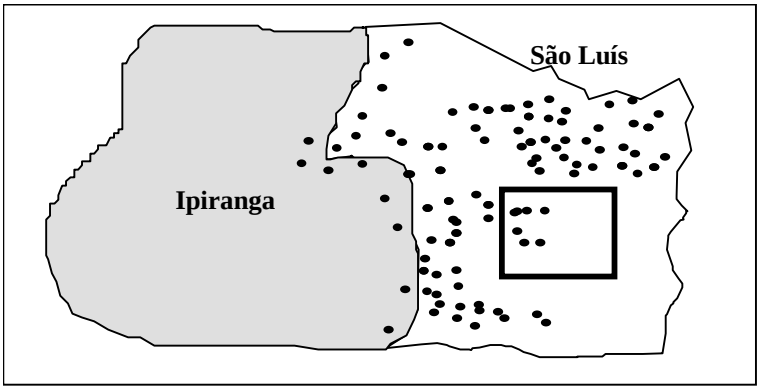

Figura 2.10 Exemplo de *containment range query* para um arquivo de dados do tipo ponto

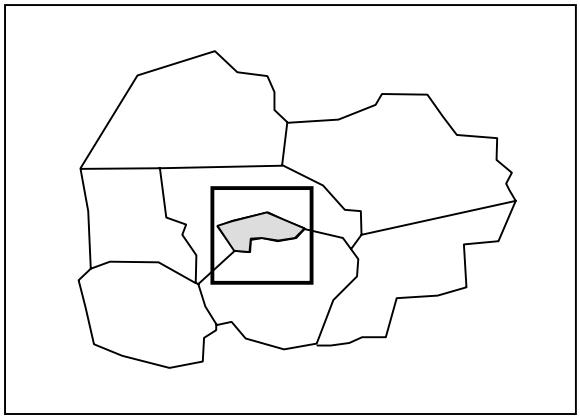

Figura 2.11 Exemplo de *containment range query* para um arquivo do tipo polígono

2.5.4.3 Enclosure Range Query

Esta consulta também é conhecida como *rectangle enclosure query*, *range query* para determinação dos objetos espaciais que contêm a janela de consulta e *range query* para determinação de inclusão [BKSS90, Cox91, See91, Car98], que consiste de um subtipo de *range query* com relacionamento topológico θ (***o.G***, ***R***) do tipo "contém". Deste modo, dado um retângulo *d-dimensional iso-oriented* $\boldsymbol{R} \subseteq E^d$, encontre todos os objetos ***o*** que englobam ***R***.

$$\boldsymbol{ERaQ\ (R,\ dataset) = \{\ o \mid o \in dataset \wedge o.G \cap R = R\ \} \cong \{\ o \mid o \in dataset \wedge o.G \supseteq R\ \}}$$

Como exemplo de uma consulta espacial deste subtipo, pode-se citar: "dado uma área retangular, relativa a uma parte de uma reserva florestal, identifique quais satélites cobrem totalmente a área retangular para fins de monitoramento de poluição ambiental e de desflorestamento"(figura 2.12). Para arquivos de dados do tipo ponto, este subtipo de *range query* não faz sentido, uma vez que pontos não possuem extensão e portanto não podem englobar a janela de consulta.

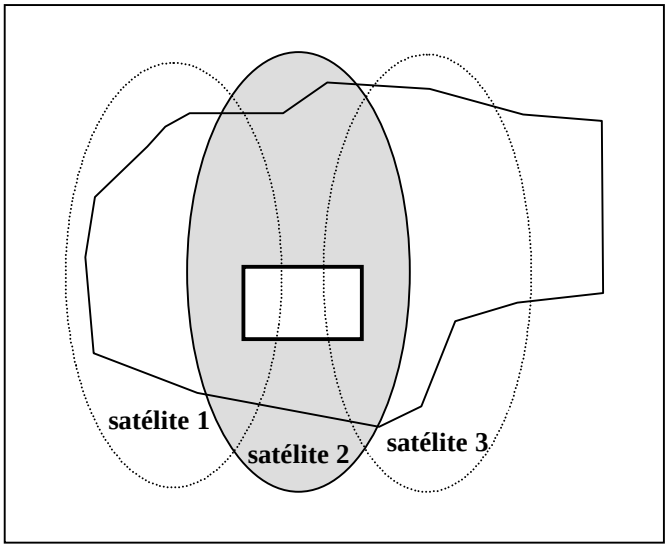

Figura 2.12 Exemplo de *enclosure range query*

2.5.5 Region Query

Esta consulta é uma versão mais genérica de consultas do tipo *range query*, a qual permite que a área de pesquisa tenha orientação e formato arbitrários, representada em geral por polígonos, círculos ou poliedros. Neste sentido, a definição deste tipo de consulta é similar à definição utilizada para *range queries*, substituindo-se somente o retângulo *d-dimensional iso-oriented* ***R*** por um objeto ***o'*** com atributo espacial $o'.G \subseteq E^d$ na definição de cada subtipo de consulta. No entanto, existe um aumento da carga de processamento para se determinar qualquer relacionamento topológico entre um objeto específico ***o'*** e os objetos ***o*** de um certo arquivo de dados. Assim, esta consulta pode tornar-se *CPU-bound* (isto é, gastar mais tempo com processamento do que com entrada e saída de dados em memória secundária).

2.5.5.1 Intersection Region Query

Esta consulta também é conhecida na literatura como *intersection query*, *overlap query* ou simplesmente *region query* [GG98]. Dado um objeto ***o'*** com atributo espacial $o'.G \subseteq E^d$, encontre todos os objetos ***o*** que tenham pelo menos um ponto em comum com ***o'.***

$$IReQ\ (o', dataset) = \{\ o \mid o \in dataset \wedge o.G \cap o'.G \neq \varnothing\}$$

2.5.5.2 Containment Region Query

Esta consulta também é conhecida como *containment query* [GG98]. Dado um objeto ***o'*** com atributo espacial $o'.G \subseteq E^d$, encontre todos os objetos ***o*** contidos em ***o'***.

$$CReQ\ (o', dataset) = \{\ o \mid o \in dataset \wedge o.G \cap o'.G = o.G\ \} \cong \{\ o \mid o \in dataset \wedge o.G \subseteq o'.G\ \}$$

Para arquivos de dados do tipo ponto, esta consulta é equivalente a *intersection region query*. A figura 2.13 adapta o primeiro exemplo definido para consultas do tipo *containment range query* (figura 2.10), considerando como **região de consulta** o bairro do Ipiranga. Note que os focos de dengue neste bairro são raros, ao contrário do que ocorre no bairro adjacente de São Luís. Caso o método de acesso multidimensional não determine eficientemente o relacionamento topológico de inclusão, este perderá muito tempo analisando focos de dengue que não pertencem ao bairro do Ipiranga.

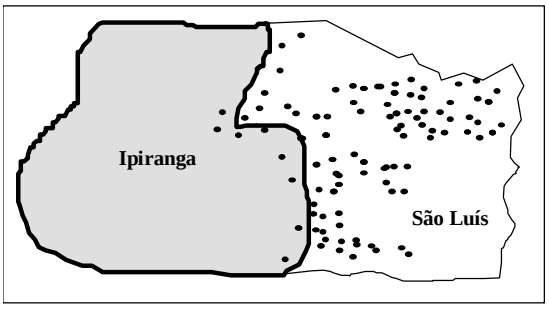

Figura 2.13 Exemplo de *containment range query*

2.5.5.3 Enclosure Region Query

Esta consulta também é conhecida como *enclosure query* [GG98]. Dado um objeto ***o'*** com atributo espacial $o'.G \subseteq E^d$, encontre todos os objetos ***o*** que englobam ***o'***.

$$EReQ\ (o', dataset) = \{\ o \mid o \in dataset \wedge o.G \cap o'.G = o'.G\ \} \cong \{\ o \mid o \in dataset \wedge o.G \supseteq o'.G\ \}$$

2.5.6 Adjacency Query

Dado um objeto ***o'*** com atributo espacial $o'.G \subseteq E^d$, encontre todos os objetos ***o*** que são adjacentes a ***o'***. Como exemplo deste tipo de consulta, pode-se querer identificar todos os estados que tenham fronteira com o estado de Pernambuco (figura 2.14).

$$AQ\ (o', dataset) = \{\ o \mid o \in dataset \wedge o.G \cap o'.G \neq \varnothing \wedge o.G^\circ \cap o'.G^\circ = \varnothing\}$$

Figura 2.14 Exemplo de *adjacency query*

2.5.7 Nearest Neighbor Query

Este tipo de consulta envolve o cálculo do relacionamento métrico de distância entre objetos espaciais, sendo esta distância definida com base em seus pontos mais próximos para objetos espaciais de dimensão não-zero. Há basicamente dois subtipos de consulta. O primeiro é baseado no cálculo da menor distância, enquanto o segundo é baseado no cálculo das k menores distâncias. Vale destacar que o cálculo da *n-ésima* menor distância ($0 < n \leq k$), quando feita de forma não incremental, pode degenerar significativamente o desempenho, isto é, o cálculo deve ser feito com base na *(n-1)-ésima* menor distância, com exceção da primeira. A simples eliminação do objeto que representa a *(n-1)-ésima* menor distância e uma posterior computação da menor distância neste novo conjunto de dados não constitui uma estratégia apropriada.

2.5.7.1 One-Nearest Neighbor Query

Dado um objeto ***o'*** com atributo espacial $o'.G \subseteq E^d$, encontre todos os objetos ***o*** que tenham a distância mínima de ***o'***. A figura 2.15 ilustra um exemplo deste subtipo de consulta, na qual é determinada a cidade mais próxima de Caruaru.

$$1\text{-}NNQ\ (o', dataset) = \{\ o \mid \forall o'' : o, o'' \in dataset \wedge \mathbf{dist}\ (o.G, o'.G) \leq \mathbf{dist}\ (o''.G, o'.G)\ \}$$

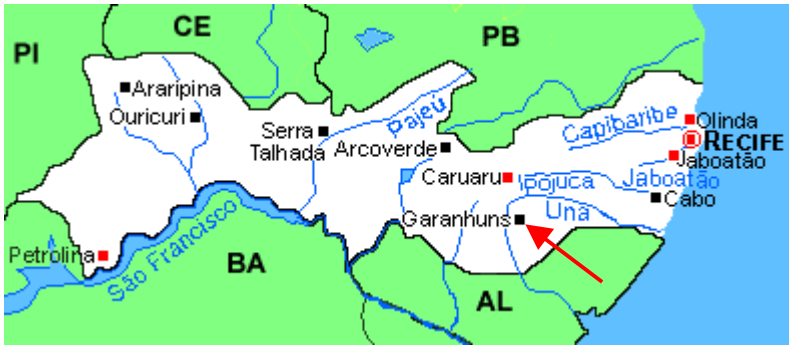

Figura 2.15 Exemplo de *1-nearest neighbor query*

2.5.7.2 *k*-Nearest Neighbor Query

Dado um objeto ***o'*** com atributo espacial $o'.G \subseteq E^d$, encontre os k objetos espaciais ***o*** mais próximos de ***o'***. A figura 2.16 adapta o exemplo anterior para mostrar as duas cidades mais próximas de Caruaru.

$$k\text{-}NNQ\ (o', dataset) = \{\ o_1 \ldots o_k \mid \forall o'' - \{o_1 \ldots o_k\} : o_1 \ldots o_k, o'' \in dataset \wedge \mathbf{dist}\ (o_1.G, o'.G) \leq \mathbf{dist}\ (o_2.G, o'.G) \leq \ldots \leq \mathbf{dist}\ (o_{k-1}.G, o'.G) \leq \mathbf{dist}\ (o_k.G, o'.G) \leq \mathbf{dist}\ (o''.G, o'.G)\ \}$$

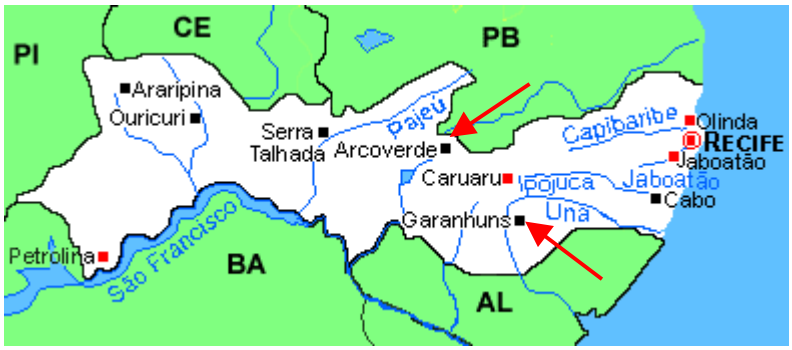

Figura 2.16 Exemplo de *k-nearest neighbor query*

2.5.8 Direction Query

Este tipo de consulta espacial envolve o cálculo de um relacionamento direcional (também conhecido como posicional) entre objetos espaciais. Dado um objeto ***o'*** com atributo

espacial $\boldsymbol{o'.G} \subseteq E^d$, encontre todos os objetos ***o*** posicionados na direção ***s*** com relação a ***o'***. Possíveis valores para a direção ***s*** são: "ao norte de", "ao sul de", "a leste de", "a oeste de", "a nordeste de", "a sudeste de", "a noroeste de" e "a sudoeste de". Desta forma, o predicado espacial direcional θ_s (***o.G***, ***o'.G***) deve ser verdadeiro para uma dada direção ***s***.

$$DQ\ (o', dataset, s) = \{ o \mid o \in dataset \wedge \theta_s(o.G, o'.G)\}$$

Um exemplo deste tipo de consulta é ilustrado na figura 2.17, na qual são determinadas todas as propriedades localizadas completamente ao sul da linha do Equador. Alguns tipos de relacionamentos direcionais, tal como "longe de", dependem diretamente de aspectos cognitivos, os quais variam conforme a cultura de um povo ou o país. Tais tipos foram omitidos para evitar qualquer ambigüidade na interpretação da consulta espacial e por conseguinte na interpretação do resultado obtido. Ademais, para conjuntos de objetos tridimensionais, também são válidos os relacionamentos direcionais de "atrás de" e "na frente de". Note também que para alguns relacionamentos posicionais há variação na nomenclatura, como exemplo "a esquerda de" é equivalente a "a oeste de".

Figura 2.17 Exemplo de *direction query*

2.5.9 Spatial Join (ou Junção Espacial)

Dado dois conjuntos de objetos espaciais *A* e *B* e um predicado espacial θ, determine todos os pares de objetos **(*o*, *o'*)** $\in$ *A* x *B* onde θ (***o.G***, ***o'.G***) é verdadeiro. O predicado espacial pode ser definido a partir de um relacionamento topológico (por exemplo, "intersecta", "contém", "é adjacente a"), a partir de um relacionamento métrico (como exemplo, *k-nearest neighbor* e [distância $\{=, \leq, <, >, \geq\}$ *q*]), ou de um relacionamento direcional (tais como, "ao norte de", "ao sul de", "a leste de" e "a oeste de"). Desta forma, percebe-se que o uso de diferentes tipos de relacionamento conduz a subtipos específicos de junção espacial. A nomenclatura utilizada para alguns destes subtipos é apresentada logo após a definição formal genérica de junção espacial.

$$SJ\ (A, B, \theta) = A \bowtie_{\theta} B = \{ (o, o') \mid o \in A \wedge o' \in B \wedge \theta\ (o.G, o'.G) \}$$

2.5.9.1 Subtipos de Junção Espacial

***intersection spatial join*: θ =** relacionamento topológico de interseção;

***containment spatial join*:** θ = relacionamento topológico do tipo "está contido em";

***enclosure spatial join*:** θ = relacionamento topológico do tipo "contém";

***adjacency spatial join*:** θ = relacionamento topológico de adjacência;

***exact match spatial join*:** θ = relacionamento topológico do tipo "é igual a";

nearest neighbor spatial join*:** θ = relacionamento métrico *k-nearest neighbor*, para $k \geq 1$. Vale destacar que para **(*o, o'*)**, ***o é um dos *k-nearest neighbors* de ***o'***;

distance minor spatial join*:** θ = relacionamento métrico de distância $< q$. A substituição do nome do operador conduz a outros subtipos, tal como ***distance equal spatial join para distância = q. A omissão do operador conduz a um subtipo genérico que representa todos os possíveis operadores (***distance spatial join***);

direction north spatial join*: θ =** relacionamento direcional do tipo "ao norte de". A substituição da direção conduz a outros subtipos, tal como ***direction northwest spatial join para "a noroeste de".

Segundo Gaede e Günther [GG98], o subtipo *intersection spatial join* é comumente utilizado para filtrar candidatos **(*o, o'*)** para outros subtipos de junção espacial baseados em relacionamentos topológicos (tais como, *containment spatial join*, *enclosure spatial join* e *adjacency spatial join*), uma vez que este conduz a um conjunto muito menor de tuplas candidatas do que o produto cartesiano A x B. Neste sentido, dentre os subtipos definidos a partir de um relacionamento topológico, *intersection spatial join* deve ser indiscutivelmente parte integrante da carga de trabalho de um *benchmark* voltado para a análise de desempenho de MAMs em função do tipo e subtipo de consulta. Ademais, seria interessante também considerar os subtipos de consulta *nearest neighbor spatial join* e *direction north spatial join*, desde que estes subtipos são versões bem particulares de junção espacial.

2.5.9.2 Exemplos de Junção Espacial

Diversos exemplos de junção espacial são encontrados na literatura. Huang *et al.* [HJR97] definem para *intersection spatial join* o seguinte exemplo: "Encontre todas as trilhas que passam por alguma floresta", que é ilustrado na figura 2.18. Roussopoulos *et al.* [RKV95] definem um exemplo para *nearest neighbor spatial join*: "Encontre os três restaurantes mais próximos de cada cinema" (figura 2.19). Já Câmara *et al.* [CCH+96] definem um exemplo para *distance minor spatial join*, o qual pode ser resumido em "Selecione todas as cidades atendidas por aeroportos, sendo que uma cidade é atendida por um aeroporto se a cidade dista menos de 50 km do aeroporto" (figura 2.20). Por fim, um exemplo de *direction north spatial join* é: "Encontre todas as cidades que estão ao norte de algum rio" (figura 2.21).

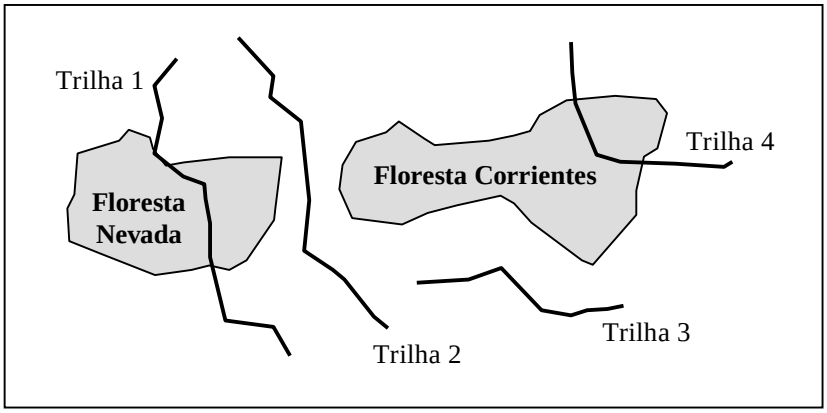

Figura 2.18 Exemplo de *intersection spatial join*

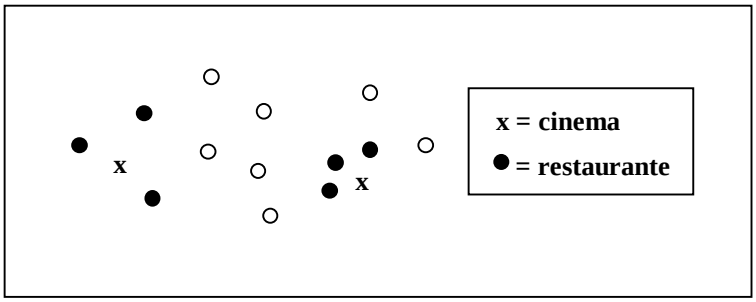

Figura 2.19 Exemplo de *nearest neighbor spatial join*

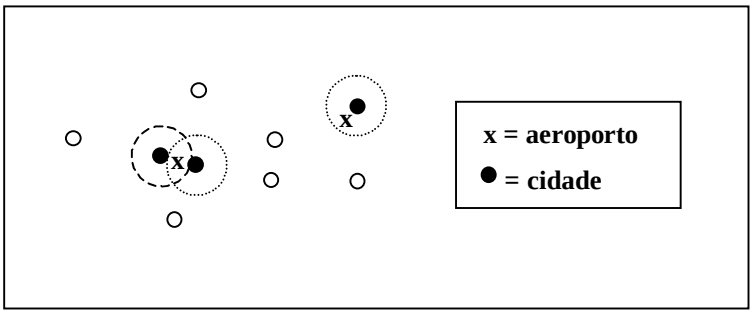

Figura 2.20 Exemplo de *distance minor spatial join*

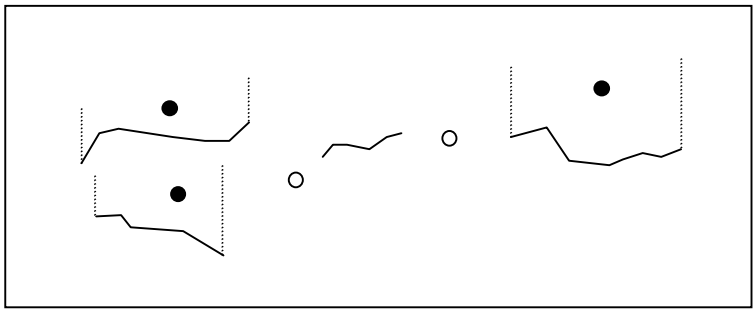

Figura 2.21 Exemplo de *direction north spatial join*

2.5.10 Distance Query

Dado um objeto ***o'*** com atributo espacial $o'.G \subseteq E^d$, encontre todos os objetos ***o*** cuja distância com relação a ***o'*** seja no máximo *q*. Este popular tipo de consulta é equivalente a *intersection region query* com um parâmetro ***o''*** específico derivado de ***o'***. No caso de ***o'.G*** ser representado por um TDE ponto, ***o''.G*** será um círculo de raio *q* e centro na localização espacial de ***o'.G***. Quando ***o'.G*** representa um objeto espacial de dimensão não-zero, ***o''.G*** será a união entre ***o'.G*** e a região formada por uma zona de *buffer* com distância *q* gerada ao redor de ***o'.G***. Uma zona de *buffer* consiste de um "corredor", cujos limites externos possuem uma distância fixa *q* com relação a um objeto espacial, e cujos limites internos são formados pelos limites do próprio objeto espacial a partir de onde a distância *q* começou a ser medida. Ciferri [Cif95] caracteriza os diversos tipos de zona de *buffer*. Analogamente para junção espacial, *distance minor spatial join* pode ser representada por uma consulta *intersection spatial join* com $o'' \in A'$ resultante de uma transformação de cada objeto $o \in A$. Deve-se, no entanto, desconsiderar todos os pares (***o''***, ***o'***) com distância exata *q* (uma vez que o operador é $<$ e não $\leq$).

$$DTQ\ (o', dataset, q) = \{\ o \mid o \in dataset \wedge dist\ (o.G, o'.G) \leq q\ \} \cong IReQ\ (o'', dataset)$$

onde o'' = região formada por o'.G e distância q

2.6 Exemplos de Métodos de Acesso Multidimensionais

Esta seção tem por objetivo descrever as principais características dos MAMs **R-tree** e **R*-tree**. Na seção 2.6.1, são descritas características relacionadas à estrutura de dados do método de acesso **R-tree**, assim como vários algoritmos de pesquisa (*intersection range query*, *containment range query e enclosure range query*) e os algoritmos de inserção, remoção e modificação de dados. Destaca-se também a descrição de três diferentes algoritmos de particionamento de nó (operação de *split*). Para o método **R*-tree**, por sua vez, são descritas na seção 2.6.2 apenas algumas de suas características particulares. Aspectos relacionados com os métodos de acesso **cell tree** e **cell tree with oversize shelves** são discutidos na seção 4.7 do capítulo 4.

2.6.1 R-tree

O método de acesso **R-tree** [Gut84] é um mecanismo de indexação espacial estruturado hierarquicamente na forma de uma árvore balanceada, similar a **B-tree** [Com79, FZ92], com suporte dinâmico para a manipulação de objetos espaciais de dimensão não-zero no espaço multidimensional. O objetivo deste método é prover suporte eficiente para consultas do tipo *range query*, típicas em sistemas *CAD* (*Computer Aided Design*) e em sistemas de informações geográficas (SIGs), através da recuperação de objetos de acordo com as suas localizações espaciais, as quais são representadas por intervalos fechados em várias dimensões.

As limitações presentes nos mecanismos de indexação existentes até 1984 motivaram a proposta da **R-tree**. Métodos voltados à manipulação de dados convencionais, tais como **hash**, **B-tree** e índices **ISAM**, não permitiam pesquisas no espaço multidimensional. Já os métodos de acesso multidimensionais eram voltados basicamente a manipulação de pontos e possuíam diversas limitações com relação: (1) à dinâmica dos dados, (2) ao tratamento de paginação em memória secundária, (3) à indexação em dimensões superiores a bidimensional, (4) ao desempenho na manipulação de grandes quantidades de dados, dentre outras.

Nos últimos anos, o método de acesso **R-tree** ganhou muita popularidade, tendo sido incorporado em vários produtos acadêmicos e comerciais, tais como, os SIGs Intergraph e MapInfo e os SGBDs Postgres, Illustra e Informix [HJR97, Min98, Gut99, Inf99a, Inf99b]. Além disto, existe na literatura uma vasta lista de pesquisas que analisam, utilizam e/ou propõem extensões a estrutura básica da **R-tree**, como exemplo a proposta de novas rotinas para o particionamento de nós (operação de *split*) [Gre89, GLL98], o acoplamento de técnicas especiais de controle de concorrência e recuperação de falhas [KB95, NK93, NK94], o suporte a consultas mais complexas (como exemplo, *nearest neighbor queries* e junção espacial) [RKV95], a proposta de técnicas específicas para construção da estrutura em ambientes estáticos [RL85], a proposta de algoritmos e arquiteturas paralelas [KF92], dentre outras.

2.6.1.1 Estrutura de Dados

A estrutura da **R-tree** é baseada em dois tipos de nó: folha (*leaf*) e interno (*non-leaf*). Um nó, independentemente de seu tipo, possui espaço para *M* entradas, sendo que cada entrada contém informações sobre: (1) uma localização espacial e (2) a localização dos dados, na memória, relativos à localização espacial representada. Além disto, cada nó deve possuir um número mínimo *m* de entradas ($m \leq M/2$), podendo tal parâmetro ser ajustado com o intuito de melhorar o desempenho da estrutura. Uma vez que a **R-tree** é usada comumente para indexação de objetos espaciais armazenados em memória secundária, um nó corresponde a uma página de disco (figura 2.22)

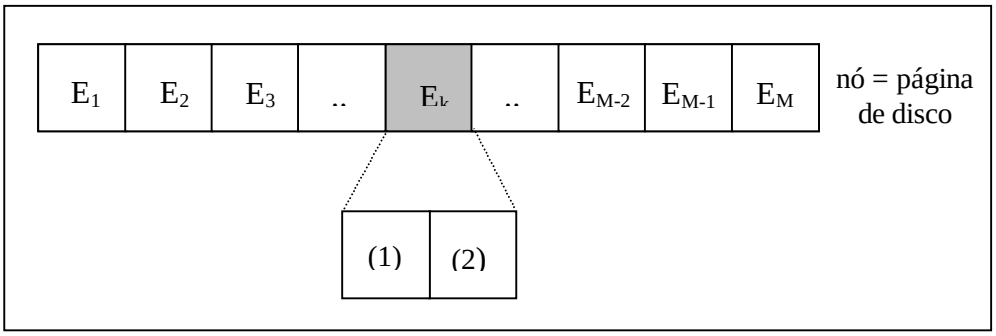

Figura 2.22 Estrutura dos nós da R-tree

A localização espacial presente nas entradas dos nós folhas e internos corresponde a um MBB *n-dimensional* e é representada na **R-tree** por um conjunto de intervalos fechados $I = (I_0, I_1, I_2, ..., I_{n-2}, I_{n-1})$, onde *n* é o número de dimensões consideradas e I_i é um intervalo fechado [a, b] que descreve o *extent* ao longo da dimensão *i*. Já a localização dos dados

corresponde a uma referência a um endereço de memória, representada geralmente através de um número.

Apesar da estrutura de nós folhas e nós internos ser idêntica, há uma diferença básica que os distinguem entre si: o tipo de elemento armazenado no nó. Nós internos armazenam informações sobre nós de nível imediatamente inferior, também conhecidos como nós filhos, enquanto que nós folhas armazenam exclusivamente informações sobre os objetos espaciais. Por ser uma estrutura balanceada, todos os nós folhas sempre estão no mesmo nível.

Um nó folha contém, portanto, entradas da forma (*I*, *id*), onde:

- **I** corresponde ao MBB *n-dimensional* do objeto espacial identificado por *id*;
- **id** é o identificador de um objeto espacial, o qual indiretamente é uma referência a um endereço de memória que possui os dados do objeto, tal como uma tupla em um banco de dados.

Já um nó interno possui entradas da forma (*I*, *p*), onde:

- **I** corresponde ao MBB *n-dimensional* do nó filho referenciado pelo ponteiro *p*, o qual engloba o MBB de todas as entradas do nó inferior e consequentemente engloba o MBB de todos os objetos espaciais contidos em nós folhas alcançados a partir desta entrada;
- **p** é o endereço de um nó filho, o qual determina uma subárvore hierarquicamente subordinada a esta entrada.

Por fim, tem-se a restrição especial que o nó raiz, quando interno, deve possuir pelo menos dois nós filhos e não *m* nós filhos como os demais nós folhas e internos. Não há restrição para o número mínimo de entradas quando o nó raiz é folha. As figuras 2.23 e 2.24 ilustram um exemplo de uma árvore **R-tree** que indexa dezoito objetos espaciais (pontos, linhas e polígonos). Na figura 2.24, é visualizada a geometria de cada objeto no *extent*, isto é, a extensão e a localização dos objetos no espaço. Já a figura 2.23 representa o conteúdo da estrutura de dados, sendo a árvore composta de 3 níveis com o valor de $M = 4$ e $m = 50\% M$.

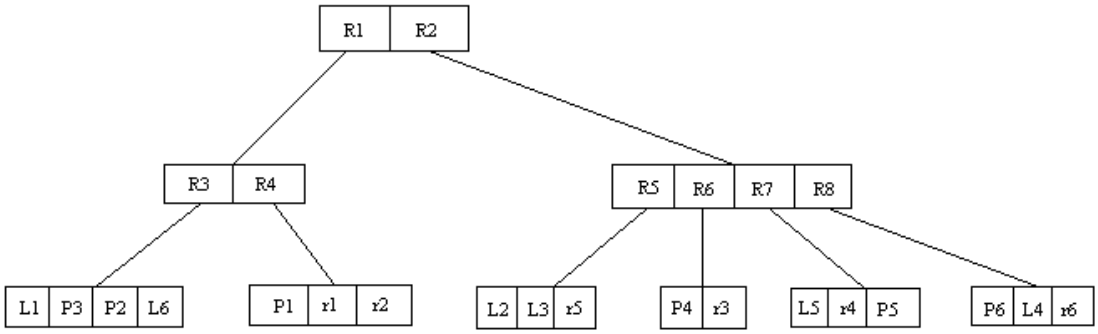

Figura 2.23 Exemplo: estrutura de dados da R-tree

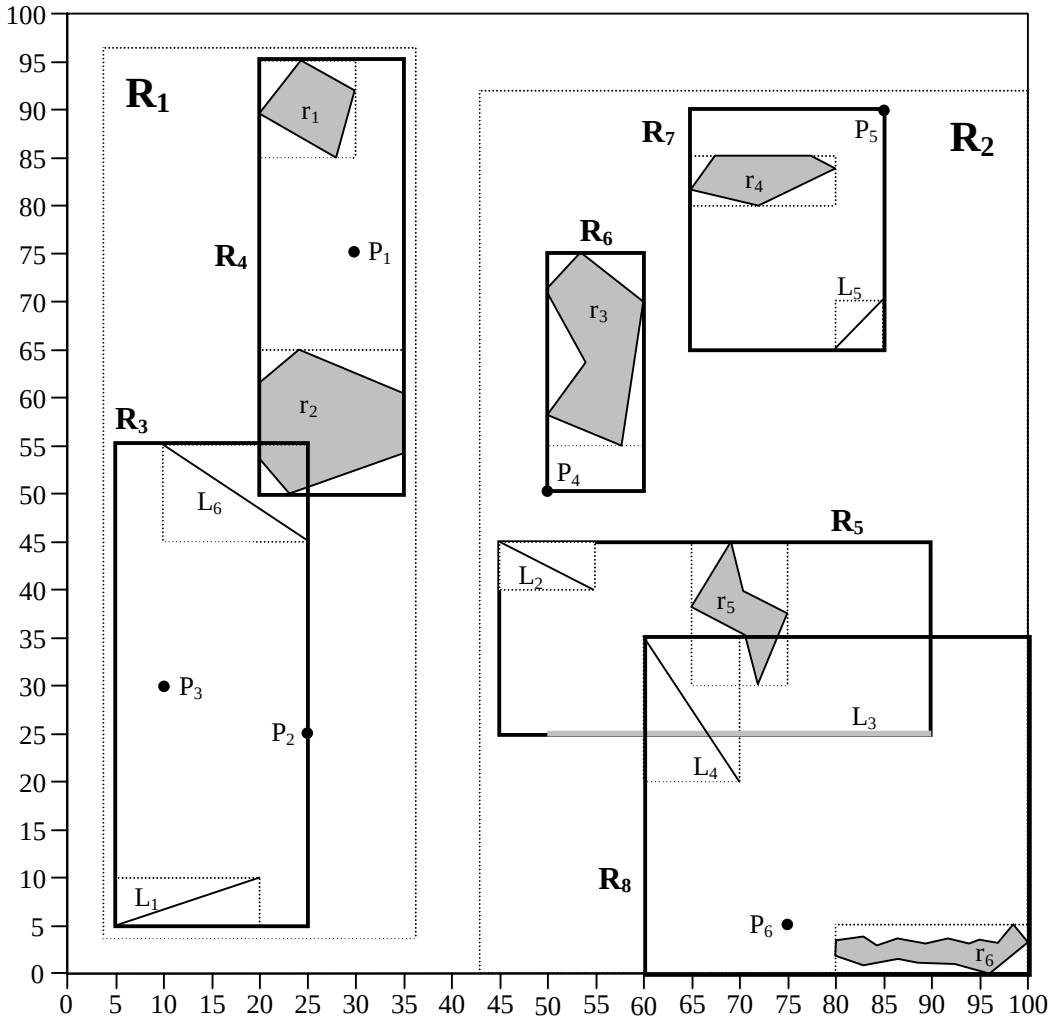

Figura 2.24 Exemplo: extensão e localização dos objetos no espaço

2.6.1.2 Consultas Espaciais

O algoritmo básico de pesquisa da **R-tree** é voltado ao suporte de consultas do tipo *intersection range query*. Para responder tais consultas, o algoritmo percorre a estrutura de forma *top-down* de maneira similar a **B-tree**, ou seja, a busca inicia-se na raiz e continua em direção às folhas. Neste percurso, para cada nó interno visitado, todas as entradas são testadas com relação à satisfação do relacionamento de interseção entre o MBB da entrada e a janela de consulta (JC). O percurso continua para as entradas cujo relacionamento for satisfeito, através da visita de seus respectivos nós filhos. Isto pode ocasionar a visita a várias subárvores alcançadas a partir de um nó interno e por conseguinte na ramificação do percurso inicial. Uma vez alcançado um nó folha, todas as entradas do nó são testadas de forma idêntica ao efetuado para nós internos, sendo o particular percurso encerrado com a possível seleção de um subconjunto do resultado da pesquisa. A busca é finalizada somente após o alcance de todos os possíveis nós folhas (ramificação do percurso). Como resultado

da pesquisa, tem-se um conjunto de objetos cujos MBBs satisfazem o relacionamento de interseção com relação à JC.

A ramificação do percurso inicial conduz a um aumento do número de caminhos de pesquisa (*search paths*) e consequentemente a visita a um maior (grande) número de nós, constituindo uma forte restrição ao desempenho do algoritmo básico de pesquisa. Desta forma, não é possível garantir um bom desempenho no pior caso, para o qual todos os nós da estrutura são visitados. Em contrapartida, a **R-tree** possui embutido nos algoritmos de inserção e remoção de dados alguns pontos de otimização que permitem a reorganização local da estrutura. Tais ajustes objetivam manter a estrutura de tal forma que o algoritmo de pesquisa consiga eliminar regiões irrelevantes do espaço multidimensional indexado, sendo examinados apenas dados próximos à JC e por conseguinte visitados um pequeno número de nós. Segundo Guttman [Gut84], isto garante um bom desempenho para a maioria dos tipos de dado espacial. Este fato, entretanto, tem sido questionado na literatura por diversos trabalhos, os quais identificaram limitações no desempenho do método de acesso **R-tree**, principalmente quando comparado a outras estruturas de indexação espacial (como exemplo, **BUDDY hash tree**, **BANG file**, **cell tree** e **R*-tree**) [KSSS89, BKSS90, Cox91, GB91, CM98].

Em especial, a ramificação do percurso inicial pode ser acentuada caso a JC intercecte o MBB de um número considerável de entradas de um nó interno. Isto pode acontecer basicamente devido a três condições: (1) sobreposição entre os MBBs das entradas de um nó interno; (2) armazenamento de MBBs grandes e (3) escolha de uma JC abrangente. Para as condições 2 e 3, respectivamente, os termos “grande” e “abrangente” são qualificados com relação ao tamanho do *extent* (espaço total indexado).

A sobreposição entre MBBs de duas ou mais entradas de um nó interno pode conduzir a ramificação do percurso caso a JC englobe parte de alguma área comum resultante da interseção dos MBBs de um subconjunto destas entradas. A ramificação do percurso acontecerá somente para as entradas cujos MBBs participam da área comum. Em especial, quanto maior o número de entradas participantes da área comum, maior será a degradação do desempenho. A ocorrência de um alto grau de sobreposição entre os MBBs das entradas de um nó interno, ou seja, de um grande número e de uma vasta área de sobreposição, aumenta a possibilidade de ramificação. Como exemplo, é ilustrado a sobreposição entre os MBBs de três entradas de um nó interno (E_1, E_2 e E_3), sendo que a JC intercecta a área comum de apenas duas destas entradas (E_1 e E_2), para as quais o percurso é ramificado (figura 2.25). Em conseqüência da **R-tree** ser uma estrutura dinâmica, torna-se impossível evitar a ocorrência de sobreposição entre os MBBs das entradas de nós internos. Para ambientes estáticos, Roussopoulos e Leifker [RL85] propõem uma técnica alternativa para a construção de uma **R-tree** que minimiza a ocorrência de sobreposições.

O armazenamento de MBBs grandes também proporciona a ramificação do percurso inicial (figura 2.26). Neste caso, a interseção entre a JC e o respectivo MBB da entrada de um nó interno é determinada basicamente pelo tamanho deste último, desde que este ocupa uma fração considerável do *extent* e por conseguinte aumenta a possibilidade de ocorrência de tal relacionamento de interseção. Um MBB grande pode ser gerado a partir da inserção de objetos espaciais grandes ou devido à alocação imprópria de um ou mais objetos em um mesmo nó folha. O primeiro caso, mais comum, é particularmente relevante para bancos de

dados que armazenam objetos de tamanhos extremamente variados (pequenos e grandes), tais como em aplicações georeferenciadas contendo dados sobre edificações de destaque e estados de uma federação. Já o segundo caso foi antecipado por Guttman, o qual adicionou certas propriedades aos pontos de otimização localizados no algoritmo de inserção (rotinas *ChooseLeaf* e *SplitNode*) de forma a reduzir a *coverage*, ou seja, a área total representada por um ou mais MBBs.

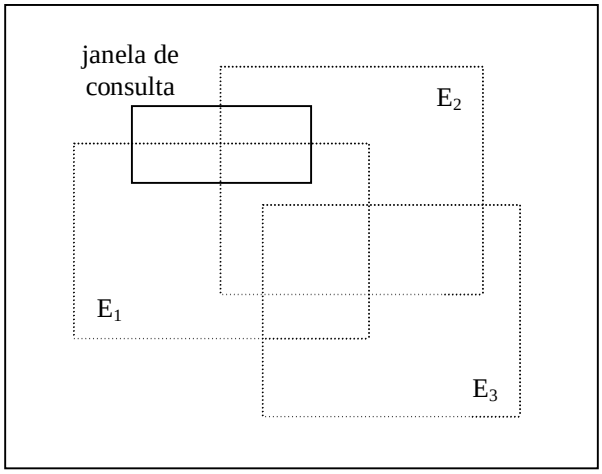

Figura 2.25 Ramificação do percurso causada pela sobreposição entre MBBs das entradas de um nó interno

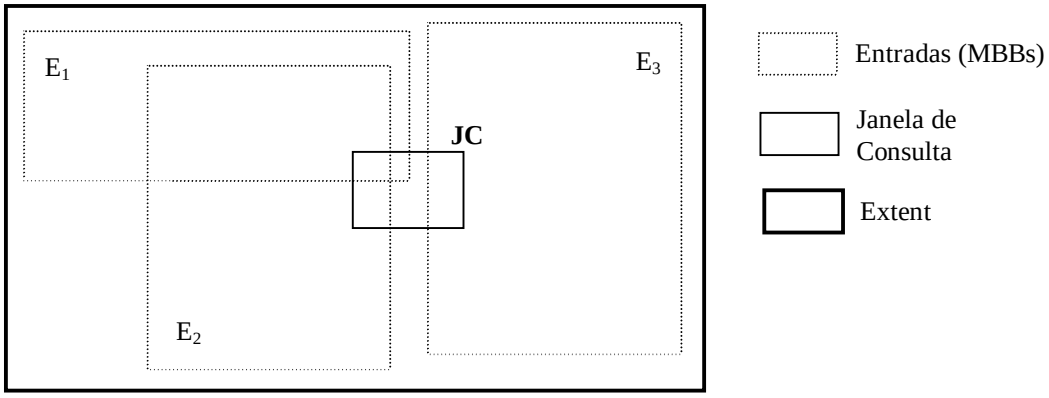

Figura 2.26 Ramificação do percurso causado pelo armazenamento de MBBs grandes

A terceira condição para ramificação do percurso inicial de pesquisa é a escolha de uma JC abrangente (figura 2.27). Neste caso, a ocorrência de uma quantidade significativa de interseções entre a JC e o MBB das entradas de um nó interno é praticamente inevitável, sendo de pouco valor os pontos de otimização, assim como outras características inerentes à estrutura.

O algoritmo básico de pesquisa pode ser adaptado para prover suporte a outros subtipos de *range query*, tais como *containment range query* e *enclosure range query*. Basicamente, o algoritmo deve ser modificado em três pontos: (1) percurso; (2) tratamento de nós internos e (3) tratamento de nós folhas. Além disto, pode ser necessário o refinamento da consulta.

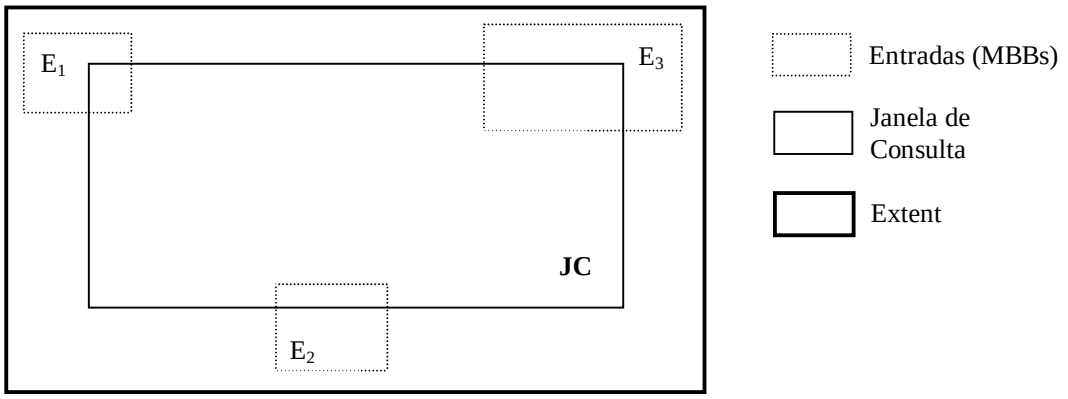

Figura 2.27 Ramificação do percurso causado pela escolha de uma janela de consulta abrangente

Consultas do tipo *containment range query* são similares a *intersection range queries* no que tange ao percurso e ao tratamento de nós internos. Para ambos os tipos de consulta, é necessário descer até as folhas, pois a interseção da JC com o MBB da entrada de um nó interno indica a possibilidade de satisfação do relacionamento topológico entre a JC e o MBB de algum objeto espacial alcançado a partir desta entrada. Isto é ilustrado na figura 2.28 e na figura 2.29, sendo que a satisfação do relacionamento de interseção em um nó interno antecipa, respectivamente, a presença do todo ou de parte do MBB do objeto obj_1 na área sobreposta e consequentemente a satisfação do relacionamento de inclusão e de interseção do MBB deste objeto com relação à JC. Com relação ao tratamento de nós folhas, consultas do tipo *containment range query* testam todas as entradas com relação à satisfação do relacionamento de inclusão do MBB da entrada na JC (MBB "está contido" na JC ?), em contraste com o teste de interseção presente em consultas do tipo *intersection range query*.

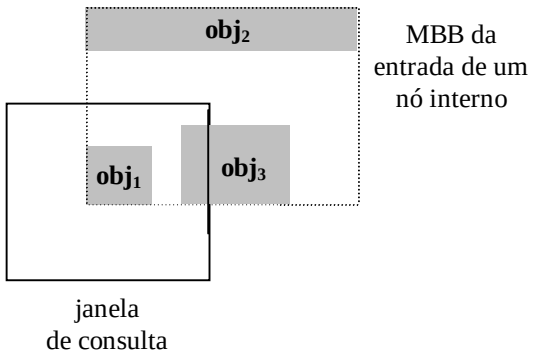

Figura 2.28 relacionamento de inclusão (está contido)

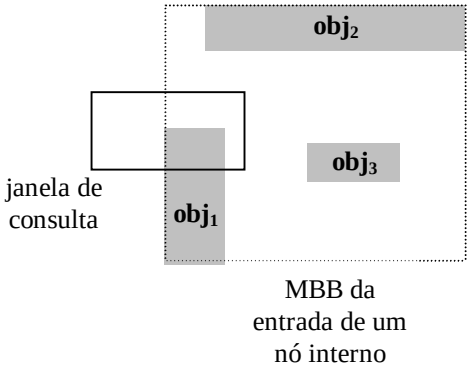

Figura 2.29 relacionamento de interseção

Consultas do tipo *enclosure range query* efetuam tratamento diferenciado com relação aos três pontos anteriormente destacados, ou seja, o percurso, o tratamento de nós internos e o tratamento de nós folhas. Tal tipo de consulta geralmente não requer a descida até as folhas, pois o fato do MBB da entrada de um nó interno não englobar a JC elimina qualquer possibilidade de um MBB de objeto espacial alcançado a partir desta entrada conter a JC, uma vez que o maior MBB de objeto terá no máximo as dimensões do MBB da entrada. Desta forma, pode-se destacar que a pesquisa concentra-se nos nós mais superiores da

estrutura e muitas vezes na própria raiz. Entretanto, o fato do MBB da entrada de um nó interno englobar a JC não garante que haverá algum MBB de objeto espacial que conterá a JC e assim a descida em direção às folhas pode ser inútil, apesar de necessária. A figura 2.30 ilustra os possíveis casos, sendo que o MBB da entrada de um nó interno: em (a) engloba a JC e também possui um MBB de objeto espacial que contém a JC; em (b) não possui nenhum MBB de objeto espacial que contém a JC, apesar da entrada englobar a JC e em (c) não engloba a JC e portanto não possui nenhum MBB de objeto espacial que possa conter a JC. Para ambos os nós internos e folhas o tratamento é o mesmo, ou seja, todas as entradas são testadas para se verificar quais destas contêm a JC.

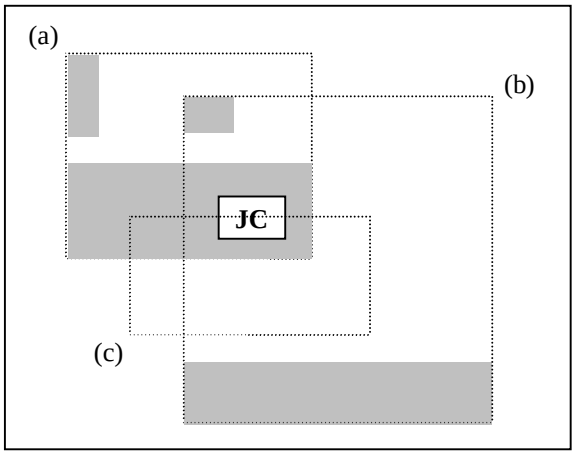

Figura 2.30 Existência do relacionamento de inclusão (contém) do MBB de um objeto espacial com relação à janela de consulta.

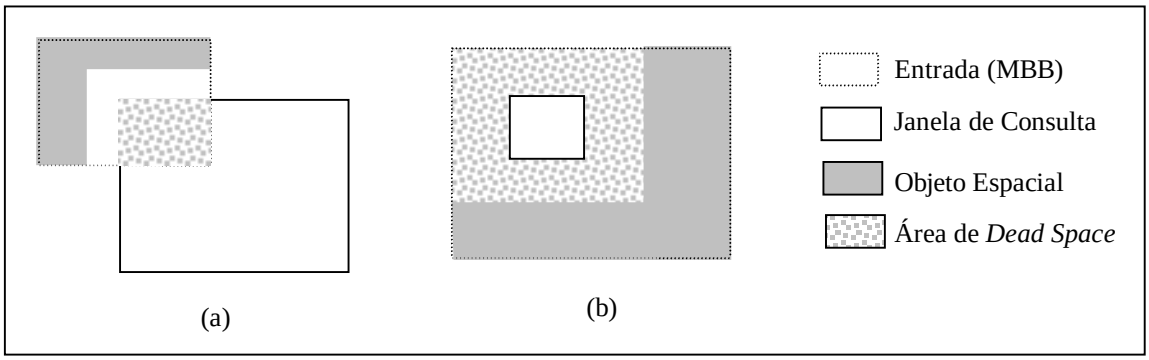

Figura 2.31 Falsos candidatos e influência da área de *dead space* na determinação dos relacionamentos topológicos de (a) Interseção e (b) Inclusão (contém)

A utilização de aproximações para representar objetos espaciais na **R-tree** exige, para consultas do tipo *intersection range query* e *enclosure range query*, um posterior acesso à representação exata dos objetos espaciais para se verificar a verdadeira satisfação do relacionamento topológico (técnica de filtragem e refinamento – seção 2.3). Isto é necessário, desde que alguns objetos espaciais pertencentes a resposta de uma pesquisa podem não satisfazer o relacionamento topológico com relação à JC (falsos candidatos), sendo este relacionamento satisfeito apenas com relação à área de *dead space* (área sem informação) contida no MBB da respectiva entrada do objeto (figura 2.31). Em particular, consultas do tipo *containment range query* não requerem o passo de refinamento, devido à propriedade transitiva do relacionamento de inclusão, ou seja, se a JC contém o MBB de uma entrada e

o MBB desta entrada contém um objeto espacial, então a JC contém o objeto espacial. Apesar da fase de refinamento da pesquisa ser muito custosa, pois requer acessos adicionais a memória secundária e considerável poder de processamento, esta não tem sido levada em consideração pela maioria dos trabalhos voltados para a análise de desempenho da **R-tree** [GG98].

2.6.1.3 Inserção de Dados

O algoritmo de inserção tem por objetivo principal realizar a alocação de novos objetos espaciais em nós folhas, de tal forma que a *coverage* (área total) das entradas de cada nó interno seja minimizado. Desta forma, procura-se reduzir a possibilidade de interseção entre os MBBs das entradas de um nó interno e a janela de consulta, e consequentemente evitar a ramificação do percurso inicial de pesquisa.

O primeiro passo realizado pelo algoritmo é a escolha de um nó folha no qual o objeto espacial será inserido. Para isto, a árvore é percorrida da raiz em direção às folhas. Em um certo nó interno, a escolha de qual entrada e portanto de qual caminho (subárvore) a ser seguido é determinada pela entrada cujo respectivo MBB sofrer o menor aumento de área, caso o objeto espacial seja inserido em um dos nós folhas alcançados a partir da entrada. Esta escolha corresponde ao **primeiro ponto de otimização** para reduzir a *coverage*. Caso haja mais de uma entrada que proporcione o mesmo aumento de área, escolhe-se a entrada que possuir o MBB de menor área.

Escolhido o nó folha, inicia-se o segundo passo do algoritmo que consiste da alocação propriamente dita, isto é, do armazenamento físico do objeto. A primeira possibilidade, mais simples, ocorre quando há espaço disponível no nó folha para o armazenamento de mais uma entrada e para isto acontecer o nó folha deve possuir no máximo *M-1* entradas. Neste caso, o armazenamento é efetuado e passa-se ao próximo passo. Por outro lado, caso não exista espaço para alocar uma nova entrada, ocorre o tratamento de *overflow* (estouro da capacidade máxima de armazenamento) por intermédio do particionamento do nó (operação de *split*). Este particionamento produz como resultado a criação de um novo nó folha e a distribuição das *M+1* entradas entre os dois nós, de modo que o parâmetro *m* seja respeitado. Os algoritmos de particionamento de nó são discutidos posteriormente, devido à sua grande importância.

O terceiro passo do algoritmo consiste na propagação de atualização dos MBBs das entradas de nós superiores, assim como a propagação dos efeitos do particionamento do nó folha, caso tenha ocorrido. Para isto, é necessário que seja percorrido o caminho inverso ao realizado no primeiro passo, ou seja, da folha em direção à raiz. A atualização dos MBBs é necessária uma vez que a inserção de um novo objeto pode alterar o MBB do nó folha e consequentemente alterar o MBB associado à entrada correspondente em um nó interno superior, a qual por sua vez também pode alterar o MBB associado ao nó interno e assim sucessivamente. Já o particionamento do nó folha requer o armazenamento de uma nova entrada no nó interno superior, a qual apontará para o novo nó folha criado e conterá o MBB das entradas distribuídas para este novo nó. Este armazenamento é tratado de forma similar ao efetuado para nós folhas, sendo possível a ocorrência de *overflow*.

O último e quarto passo tem por objetivo tratar do particionamento da raiz da **R-tree**. Neste caso, cria-se uma nova raiz, a qual terá como filhos os dois nós resultantes do particionamento.

O particionamento de nós é de suma importância para a estrutura e corresponde ao **segundo ponto de otimização**, também voltado à redução da *coverage*. Como meta para o particionamento, a área total relativa ao MBB dos dois nós deve ser minimizada. Guttman [Gut84] propõe três diferentes algoritmos de particionamento com complexidade de tempo e minimização da *coverage* distintos.

O primeiro algoritmo, exaustivo, investiga todas as possibilidades de agrupamento para as $M+1$ entradas, de forma a escolher a melhor distribuição possível. Assim, reduz-se ao máximo a *coverage*, mas o grande número de possibilidades, aproximadamente 2^{M-1}, o torna extremamente lento para valores aceitáveis de M (valores atuais variam de 50 a 400).

O segundo algoritmo possui uma complexidade quadrática em M e linear no número de dimensões. Este algoritmo, denominado de quadrático, garante encontrar uma pequena *coverage*, mas não garante obter a menor *coverage* possível. Inicialmente, o algoritmo escolhe duas das $M+1$ entradas para serem as sementes de cada um dos nós. Para isto, calcula-se a ineficiência de agrupamento para todos os pares de entrada, a qual consiste da área de *dead space* no MBB gerado por ambas as entradas quando alocadas conjuntamente, e escolhe-se o par que tiver o maior desperdício de área. Este par tende a possuir os MBBs mais distantes entre si. As demais entradas são associadas, uma por vez, a um dos nós. O critério usado é baseado no aumento de área proporcionado pela inclusão da entrada em cada um dos nós, sendo escolhida a entrada que gerar a maior diferença entre ambos os aumentos, o que demonstra uma maior preferência por um dos nós. A alocação imprópria desta entrada pode gerar uma *coverage* enorme. Visando a minimização da *coverage*, a entrada escolhida é associada ao nó que sofrer o menor aumento de área. Outros critérios adicionais de escolha do nó, para o caso de um possível empate, são: (1) a escolha do nó que possui o MBB de menor área e (2) por último, a escolha do nó que possui o menor número de entradas. Durante a associação das entradas remanescentes, verifica-se a necessidade de associação de todas as entradas a um dos nós, de forma a cumprir o valor mínimo requerido pelo parâmetro m. Isto acontece quando o outro nó atinge $M-m+1$ entradas. Esta associação, sem consideração de critérios geométricos, pode degenerar o particionamento.

Beckmann *et al.* [BKSS90] atribuem os problemas do algoritmo de particionamento quadrático à escolha de sementes pequenas, que tende a acontecer na presença de MBBs de tamanhos extremamente variados (pequenos e grandes) e/ou na ocorrência de um alto grau de sobreposição. Tal escolha pode associar inapropriadamente a primeira entrada de um dos nós, de forma a agrupar MBBs muito distantes e com grande potencial para se gerar uma *coverage* extensa. Além disto, as subsequentes associações de entradas serão na maioria das vezes efetuadas no nó que recebeu a primeira entrada, desbalanceando e prejudicando a distribuição das entradas entre os dois nós. Este desbalanceamento na distribuição, por sua vez, determina a associação das entradas remanescentes sem a consideração de critérios geométricos, conforme destacado anteriormente.

O último algoritmo de particionamento possui complexidade linear com relação a *M* e com relação ao número de dimensões. A estrutura do algoritmo é idêntica a do algoritmo quadrático, mas este utiliza rotinas simplificadas para a escolha das sementes dos nós e para a associação das entradas remanescentes. O algoritmo linear, pela simplicidade das decisões, não garante encontrar uma pequena *coverage* para ambos os nós. A escolha das sementes é baseada na distância normalizada entre o mais alto lado inferior e o mais baixo lado superior dos MBBs ao longo de cada uma das dimensões (figura 2.32). Escolhe-se o par de entradas cujos respectivos MBBs possuem a maior distância normalizada. Já a associação das demais entradas é feita ao acaso, escolhendo-se qualquer uma das entradas remanescentes.

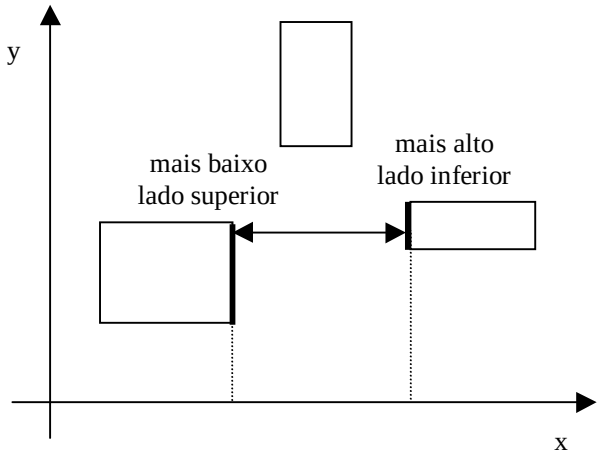

Figura 2.32 Cálculo da distância com relação à dimensão x

Cox [Cox91] discute a tentativa de tornar zero o *overlap* (área comum representada por dois ou mais MBBs) e ao mesmo tempo minimizar ao máximo a *coverage* durante o particionamento de nós, de modo a evitar a ramificação do percurso inicial de pesquisa. Embora a situação ideal fosse o atendimento simultâneo desses dois quesitos, em algumas situações este objetivo torna-se inalcançável, sendo necessário a definição de prioridades. O método **R-tree**, como visto, opta como heurística de otimização a minimização da *coverage*.

2.6.1.4 Remoção de Dados

O algoritmo de remoção é realizado segundo quatro passos seqüenciais, de modo semelhante ao algoritmo de inserção, a saber: (1) busca do nó folha que contém a entrada a ser removida; (2) remoção propriamente dita; (3) propagação de mudanças e (4) ajuste da raiz.

A busca do nó folha é similar à descrita para o algoritmo básico de pesquisa, tendo como particularidades (1) o fato da janela de consulta ser o próprio MBB da entrada a ser removida; (2) a não obrigatoriedade de se percorrer todos os possíveis caminhos de pesquisa, alcançada por intermédio da utilização de uma nova condição de parada (entrada encontrada) e (3) o uso do identificador do objeto espacial como critério adicional para identificação da entrada nos nós folhas.

Uma vez encontrado o nó folha que contém a entrada a ser removida, efetua-se a remoção propriamente dita. Inicia-se então a propagação de mudanças, da folha em direção

a raiz, a qual consiste inicialmente da propagação de *underflow*, quando necessário, seguida da propagação de atualização dos MBBs das entradas de nós superiores.

Um *underflow* ocorre quando, após a remoção de uma entrada de um nó, o número de entradas torna-se inferior a capacidade mínima de armazenamento *m*. Nesta situação, o nó deve ser eliminado, suas entradas colocadas em um conjunto *Q* para posterior reinserção na estrutura e a entrada correspondente no nó interno superior removida. A eliminação desta última entrada pode ocasionar um *underflow* no nó interno superior e assim sucessivamente. Cessada a propagação de *underflow*, é efetuado o ajuste dos MBBs das entradas de nós superiores. Isto é necessário, pois a remoção de uma entrada pode alterar o MBB do nó e por conseguinte alterar o MBB associado à entrada correspondente no nó interno superior, a qual por sua vez também pode alterar o MBB associado ao nó interno superior e assim por diante.

Por fim, a raiz é ajustada no caso desta possuir somente uma entrada. Neste caso, o nó filho passa a ser a nova raiz da estrutura.

O tratamento efetuado pela **R-tree** na ocorrência de *underflow* (reinserção de entradas) difere significativamente do efetuado em uma **B-tree**, para a qual pode-se: (1) redistribuir algumas das entradas relativas a um nó irmão (*sibling*[1]) e ao nó pai, com o intuito de impedir a remoção do nó que sofreu *underflow* ou (2) na impossibilidade de se efetuar o primeiro tratamento, devido à propagação do *underflow* aos nós irmãos, redistribuir as entradas do nó que sofreu *underflow* para um nó irmão, com o intuito de se agrupar o conteúdo de dois nós (*merge*). No contexto espacial, as entradas de nós irmãos não mantém necessariamente qualquer relacionamento de adjacência e com isto pode-se estender o segundo tratamento de modo que as entradas sejam redistribuídas para mais de um nó irmão. No entanto, a utilização de ambos os tratamentos para *merge* em uma **R-tree** pode gerar um aumento considerável do MBB dos nós irmãos, invalidando qualquer tentativa de minimização da *coverage* efetuada pelos pontos de otimização contidos no algoritmo de inserção.

A reinserção de entradas relativas aos nós eliminados (transferidas para o conjunto *Q*) permite um refinamento incremental da estrutura de dados da **R-tree**, desde que a alocação das entradas nos nós será otimizada levando-se em consideração a configuração atualizada da árvore, a qual possivelmente contém objetos espaciais inexistentes na primeira alocação destas entradas e por conseguinte novos nós internos e folhas que podem proporcionar uma maior redução da *coverage* em todos os níveis. A reinserção de cada entrada, no entanto, deve ser feita de acordo com seu respectivo nível original, ou seja, uma entrada removida de um nó folha deve ser reinserida em um outro nó folha, ao passo que uma entrada removida de um nó interno deve ser reinserida em um nó interno de mesmo nível. Carneiro [Car98] também observou que a reinserção de entradas deve iniciar pelas entradas relativas ao nível mais superior (últimas entradas adicionadas ao conjunto *Q*) e que posteriormente deve ser seguida pela reinserção das entradas relativas ao nível imediatamente inferior, e assim por diante. A utilização de uma ordem inversa da verificada na propagação de *underflow* é necessária, pois as entradas relativas aos níveis superiores, quando inseridas, poderão servir para guiar a reinserção das demais entradas de nível inferior remanescentes no conjunto *Q*. De acordo com Guttman [Gut84], o processo de reinserção de entradas é eficiente, uma vez

[1] *sibling nodes* = nós que possuem o mesmo nó pai.

que grande parte dos nós a serem manipulados na reinserção já se encontram em memória principal, correspondendo aos mesmos nós que foram acessados no percurso de busca do nó folha. A reinserção de entradas corresponde ao **terceiro ponto de otimização** da **R-tree**.

2.6.1.5 Modificação de Dados

A mudança da geometria associada a um objeto espacial indexado pela estrutura **R-tree** gera uma inconsistência no MBB da entrada do nó folha utilizado para representar o objeto. Para resolver este problema, deve-se primeiramente remover a respectiva entrada da árvore. Em seguida, deve-se atualizar o MBB associado ao objeto espacial e por fim reinserir a entrada na estrutura. Vale destacar que o simples ajuste do MBB da entrada do nó folha não é indicado, pois pode-se gerar um excessivo aumento da *coverage* do nó folha, o qual será propagado aos nós superiores. O algoritmo de modificação, portanto, basicamente utiliza os algoritmos de inserção e remoção para alcançar os seus fins.

2.6.2 R*-tree

O desenvolvimento do método de acesso **R*-tree** [BKSS90] foi motivado pelo estudo da **R-tree** quanto aos critérios utilizados por esta estrutura para prover um bom desempenho. A **R*-tree** estende os critérios heurísticos de otimização, utilizando uma combinação de fatores que, segundo análises efetuadas pelos projetistas da estrutura de indexação, proporciona um melhor desempenho na manipulação de objetos espaciais de dimensão não-zero do que o obtido com o método de aceso **R-tree**.

Pode-se definir uma **R*-tree** como um mecanismo de indexação espacial derivado da **R-tree**, e portanto estruturado hierarquicamente na forma de uma árvore balanceada, que propicia suporte dinâmico na manipulação de objetos espaciais e que proporciona um bom desempenho na recuperação destes objetos através da utilização dos critérios de *coverage*, *overlap*, *margin* e *storage*. A estrutura dos nós (folhas e internos) é a mesma da **R-tree**, assim como as restrições aplicadas aos nós e os algoritmos de pesquisa, remoção e modificação. O algoritmo de inserção, por sua vez, concentra todas as extensões definidas pela **R*-tree** e será descrito posteriormente em separado.

Quanto aos critérios, *coverage* e *overlap* já foram definidos anteriormente, na seção 2.6.1. O critério *storage* refere-se à quantidade de espaço utilizada pela estrutura, a qual é influenciada diretamente pelo parâmetro *m*. Nos experimentos, determinou-se que o valor de *m* deve ser igual a 40% de *M*, de forma a proporcionar o melhor desempenho com relação à complexidade de tempo. O último critério, *margin*, corresponde ao perímetro do MBB (isto é, a soma do tamanho dos lados do MBB). A análise destes critérios permitiu a identificação de propriedades que conduzem a um aumento no desempenho do MAM.

O controle da *coverage*, por exemplo, pode proporcionar um melhor desempenho, uma vez que decisões sobre o percurso raiz para folhas podem ser tomadas nos níveis mais altos. Deve-se, portanto, minimizar tal critério, de forma a reduzir a possibilidade de interseção entre o MBB da entrada de um nó e o MBB da janela de consulta. Tal minimização permite o agrupamento de dados espaciais próximos uns aos outros, pertencentes a uma restrita

região geográfica, além de conduzir à redução da área de *dead space*. A minimização do *overlap* também permite reduzir o número de caminhos a serem percorridos da raiz aos nós folhas (respectivas subárvores), em especial na pesquisa de entradas dos níveis inferiores (objetos espaciais). Isto ocorre devido ao fato de que a escolha de vários caminhos pode ser influenciada pela sobreposição entre os MBBs das entradas de um certo nó interno. Por intermédio de *0-overlap* (ausência de sobreposição), reduz-se a possibilidade que mais de uma subárvore abaixo de um nó seja visitada. Já a minimização do critério de espaço de armazenamento, *storage*, permite obter uma árvore mais baixa (ou rasa), o que encurta o percurso "raiz para folhas" de uma determinada árvore (menos acessos a disco). Tal critério também permite diminuir a quantidade total de nós (páginas de disco alocadas), que pode afetar drasticamente o desempenho de uma consulta que possua, por exemplo, uma grande área de interseção com os objetos espaciais armazenados, necessitando a visita a muitos nós da estrutura. Por fim, o critério *margin* permite, quando minimizado, o ajuste dos MBBs para o formato quadrático. Este ajuste facilita o agrupamento de MBBs em um nó interno e consequentemente reduz a *coverage* das entradas de nós superiores.

2.6.2.1 Inserção de Dados

A estrutura geral do algoritmo é idêntica a do algoritmo descrito para a **R-tree**. Entretanto, a lógica de determinadas rotinas é alterada, de modo a incorporar os critérios de otimização anteriormente citados. Pode-se destacar três modificações cruciais, feitas no (1) algoritmo de escolha do nó folha, nomeado *ChooseSubtree*, que corresponde ao algoritmo *ChooseLeaf* na **R-tree**, no (2) tratamento de *overflow*, o qual agora incorpora uma outra abordagem além do particionamento do nó e no (3) algoritmo de particionamento de nós.

A escolha de um caminho apropriado para a inserção de um novo objeto espacial é determinado pela análise das entradas de cada nó interno, que determinarão qual caminho (subárvore) a ser seguido no percurso raiz para folhas e consequentemente qual nó folha será escolhido. Sendo o *overlap* de uma entrada definida como:

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

O Algoritmo ChooseLeaf é modificado de forma que o passo CL3 seja:

- se as entradas de um nó apontarem para nós folhas, deve-se determinar o menor custo de sobreposição (*overlap*). Isto é feito analisando-se todas as entradas do nó e escolhendo-se a entrada que tiver o menor aumento da área de sobreposição para incluir o novo objeto espacial, de acordo com a fórmula acima descrita. Empates são resolvidos pela escolha da entrada cujo MBB sofrer o menor aumento de área;
- se as entradas de um nó não apontarem para nós folhas, deve-se determinar o menor custo de área (*coverage*). Para isto, são analisadas todas as entradas de um nó, sendo escolhida a entrada que sofrer o menor aumento de área para incluir o novo objeto espacial. Empates são resolvidos pela escolha da entrada com o menor MBB.

A primeira parte deste algoritmo, custo de sobreposição, pode ser otimizada reduzindo-se a quantidade de entradas de um nó que terão os seus custos de sobreposição calculados. Como primeiro passo, deve-se colocar em ordem crescente as entradas de acordo com o aumento de área necessário para incluir o novo objeto espacial. Escolhe-se, então, apenas as *p* primeiras entradas, para as quais o custo de sobreposição é calculado. Tal estratégia conduz a um custo de sobreposição quase mínimo. Para o espaço Euclidiano bidimensional, o valor de $p = 32$ praticamente não proporcionou nenhuma redução no desempenho da **R*-tree** nos experimentos realizados por Beckmann *et al.* [BKSS90].

Outra modificação, correspondente a segunda otimização proporcionada pela **R*-tree**, refere-se ao tratamento de *overflow*. Quando ocorre um *overflow* na inserção de uma nova entrada em um nó, este nó não é imediatamente particionado. Caso seja a primeira vez que a rotina de tratamento de *overflow* esteja sendo executada em um certo nível (o nível não pode ser o da raiz) durante a inserção de uma entrada, opta-se pela reinserção de *p* entradas (*forced reinsert*). O algoritmo de reinserção ordena as $M+1$ entradas de modo decrescente de acordo com a distância do centro do MBB da entrada com relação ao centro do MBB do nó (ou seja, do MBB que engloba todas as entradas do nó). Remove-se as *p* primeiras entradas, isto é, as entradas cujos MBBs possuem os centros mais distantes do centro do MBB do nó. A reinserção das *p* entradas escolhidas pode ser efetuada da entrada mais distante para a menos distante (técnica *far reinsert)* ou da entrada menos distante para a mais distante (técnica *close reinsert*). O valor de $p = 30\%$ de *M* proporcionou os melhores resultados de desempenho em [BKSS90]. Dentre os benefícios da reinserção de entradas pode-se citar: (1) a redução do *overlap*, devido à freqüente alocação da entrada reinserida em um outro nó; (2) o aumento da taxa de utilização dos nós; (3) a redução da ocorrência de particionamentos de nós, que são altamente custosos e (4) a formação de MBBs mais quadráticos nos nós superiores, devido à reinserção de entradas mais afastadas do centro do MBB do nó, o que indiretamente reduz a *coverage* das entradas de nós superiores.

O algoritmo de particionamento de nós apenas é executado quando, na inserção de uma entrada, ocorrer o segundo tratamento de *overflow* em um mesmo nível. Em outras palavras, este algoritmo trata especificamente de entradas que mesmo após reinseridas continuam causando *overflow* no mesmo nível, consistindo de duas partes. A primeira parte objetiva a escolha do eixo perpendicular de partição, segundo o qual as entradas serão distribuídas em dois nós. Esta escolha é baseada no critério de *margin*, o qual procura-se minimizar. Já a segunda parte tem por objetivo encontrar a melhor distribuição das $M+1$ entradas em dois grupos (nós), considerando o eixo de partição escolhido. Para isto, procura-se minimizar, em primeiro lugar, o critério de *overlap* e, em segundo lugar, o critério de *coverage*.

No processo de escolha do eixo, para cada dimensão, as $M+1$ entradas são ordenadas primeiramente pela menor coordenada e em seguida pela maior coordenada, gerando dois conjuntos distintos de ordenação. Para cada conjunto de ordenação, $M-2m+2$ distribuições das $M+1$ entradas em dois grupos são geradas. A *k-ésima* distribuição é definida como: o primeiro grupo contém as primeiras $m-1+k$ entradas e o segundo grupo contém as entradas remanescentes. Calcula-se, então, o valor de *P* para cada uma das distribuições, onde *P* consiste da soma dos perímetros dos MBBs que englobam cada um dos grupos, ou seja, $P = margin[\text{MBB}(\text{grupo}_1)] + margin[\text{MBB}(\text{grupo}_2)]$. Após, calcula-se, para cada eixo, o valor de *S* que consiste da somatória dos perímetros *P* de todas as suas distribuições. Escolhe-se o eixo que possuir o menor valor de *S*.

Uma vez definido o eixo de partição, deve-se escolher, dentre as distribuições geradas pela dimensão, a distribuição que possuir o menor valor de *overlap*, sendo que o *overlap* de uma distribuição corresponde a O = área [MBB($grupo_1$) $\cap$ MBB($grupo_2$)]. Possíveis empates são resolvidos através da escolha da distribuição com o menor valor de *coverage*, sendo a *coverage* = área [MBB ($grupo_1$)] + área [MBB ($grupo_2$)].

2.7 Sistemas de Informações Geográficas

A grande popularização dos computadores e o avanço tecnológico em áreas, como exemplo cartografia, geografia, sensoriamento remoto e a ciência da computação como um todo, levou, nos últimos anos, a uma crescente necessidade de sistemas computacionais para aplicações que permitam a manipulação (processamento, análise, armazenamento e recuperação) de dados espaciais, também chamados dados georeferenciados. Estes sistemas dedicados ao tratamento de dados que representam objetos e fenômenos e suas localizações espaciais são chamados Sistemas de Informações Geográficas (SIG).

Os SIG incluem diferentes fases do processamento de dados geográficos: aquisição, pré-processamento (a conversão dos dados e a definição do sistema de coordenadas a ser usado), gerenciamento (modelagem dos dados e definição de mecanismos para manipulação dos mesmos), análise dos dados e geração do produto final. Nesta seção, são apresentados inicialmente alguns conceitos básicos associados à área de geoprocessamento e definições relacionadas aos SIG. Em seguida abordaremos aspectos relativos à conversão de dados e apresentaremos os conceitos associados à modelagem de dados geográficos.

2.7.1 Conceitos Básicos

Para melhor entendimento dos dados geográficos, apresentamos alguns conceitos essenciais associados à cartografia, à geografia e ao sensoriamento remoto.

- **Projeção:** cada ponto do elipsóide ou esfera (globo terrestre) projetado em uma superfície plana. Esta superfície, o mapa, pode ser apresentada em diferentes escalas. As projeções cartográficas mais utilizadas são as planas ou azimutais, cônicas e cilíndricas. Existe também a projeção UTM (*Universal Transverse Mercator*) que é definida dividindo-se a Terra em 60 fusos de 6 graus de longitude.
- **Escala:** relação entre as dimensões dos elementos representados em um mapa e a grandeza correspondente, medida sobre a superfície da Terra. As escalas numéricas são descritas por frações cujos denominadores representam as dimensões naturais e os numeradores as que lhes correspondem no mapa. Para ilustrar, alguns exemplos de escalas são: 1:50.000 ou 1/20.000.
- **Sistemas de Coordenadas:** define a localização de qualquer ponto da superfície terrestre. Esta localização é estabelecida em relação a outros objetos cujas posições são previamente conhecidas e é determinada em uma rede coerente de coordenadas. Existem dois grupos de sistemas de coordenadas: