Instituto de Computação
Universidade Estadual de Campinas

# Análise de Desempenho de Métodos de Acesso Espaciais Baseada em um Banco de Dados Real

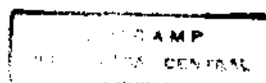
**Alexandre Pedrosa Carneiro**

Abril de 1998

**Banca Examinadora:**

- Prof. Dr. Geovane Cayres Magalhães (Orientador)
- Prof. Dr. Marcelo Gattass
  Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro
- Prof. Dr. Mário Antonio do Nascimento
  Instituto de Computação – Universidade Estadual de Campinas
- Profa. Dra. Claudia M. Bauzer Medeiros (Suplente)
  Instituto de Computação – Universidade Estadual de Campinas

99118275



UNIDADE BC
N.º CHAMADA
V.
TOMBO 37829
PROC. 229/99
C [X]
PREÇO R$11,00
DATA 10/06/99
N.º CPD

CM-00124515-3

**FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP**

Carneiro, Alexandre Pedrosa

C215a Análise de desempenho de métodos de acesso espaciais baseada em um banco de dados real / Alexandre Pedrosa Carneiro -- Campinas, [S.P. :s.n.], 1998.

Orientador : Geovane Cayres Magalhães

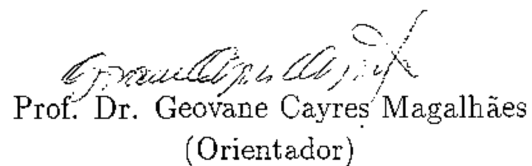
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Sistemas de informações geográficas. 2. Estruturas de dados (Computação). 3. Banco de dados - Gerência. I. Magalhães, Geovane Cayres. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

# Análise de Desempenho de Métodos de Acesso Espaciais Baseada em um Banco de Dados Real

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Alexandre Pedrosa Carneiro e aprovada pela Banca Examinadora.

Campinas, 22 de maio de 1998.

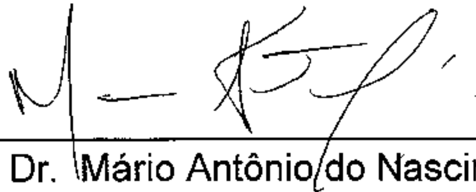


Prof. Dr. Geovane Cayres Magalhães
(Orientador)

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

Tese de Mestrado defendida e aprovada em 22 de maio de 1998 pela Banca Examinadora composta pelos Professores Doutores

Prof. Dr. Marcelo Gattass



Prof. Dr. Mário Antônio do Nascimento

Prof. Dr. Geovane Cayres Magalhães

*Aos meus pais, Lair e Rosalina.*
*à minha esposa, Cilamar,*
*e ao meu filho, Rafael.*

*Hoje me sinto mais forte, mais feliz quem sabe*
*Só levo a certeza de que muito pouco eu sei*
*Eu nada sei*

(Tocando em frente – Almir Sater e Renato Teixeira)

# Prefácio

Esta dissertação apresenta uma análise comparativa de desempenho de métodos de acesso espaciais a partir de um banco de dados real.

Embora haja um grande número de pesquisas envolvendo a comparação de desempenho de métodos de acesso espaciais, pouco até hoje se fez para que essas avaliações levem em conta as características de grupos específicos de aplicações, o que em parte se deve à dificuldade de se obter conjuntos de dados reais que as representem. A utilização de dados reais é necessária, uma vez que a geração de dados sintéticos pode resultar em conjuntos de dados com características atípicas, levando a conclusões não necessariamente extensíveis a determinado tipo de aplicação. Neste contexto, as principais contribuições deste trabalho são:

- a conversão de um conjunto de dados reais representativos para aplicações de gerenciamento de serviços de utilidade pública, tais como telefonia, eletricidade e água, para um formato em que ele pode ser facilmente repassado a outros pesquisadores;
- a avaliação do desempenho de um grupo de métodos de acesso espaciais pertencentes à família da R-tree na indexação desse conjunto de dados.

Alguns dos resultados dos experimentos divergiram de outros obtidos por um grupo de pesquisadores a partir de dados sintéticos, reforçando a necessidade do uso de dados reais representativos na comparação de desempenho de métodos de acesso espaciais.

Esta dissertação traz ainda um levantamento das diversas técnicas utilizadas na indexação de dados espaciais.

# Abstract

This dissertation presents a comparative performance analysis of spatial access methods based on a real-life database.

In spite of the large amount of research dealing with the performance comparison of spatial access methods, very little has been done when it comes to considering the properties of specific groups of applications. In part, this is due to the difficulty in obtaining real data sets to represent them. The use of real data is necessary, since synthetic data generation may result in data sets with atypical characteristics, which may lead, in turn, to conclusions that don't apply to a given application type. In this context, the main contributions of this work are:

- the conversion of a real data set that is representative of geographic applications for public utility services management to a format in which it may be easily delivered to other researchers. Public utility services include telecommunication, electricity and water supply, and the like.

- the performance comparison of a group of spatial access methods of the R-tree family with regards to the indexing of this data set.

The accomplished experiments have shown some results that disagree with other ones obtained by a group of researchers who have based on synthetic data sets, reinforcing the need of using representative real data sets.

This dissertation also presents a survey of several techniques used in spatial data indexing.

# Agradecimentos

Agradeço, em primeiro lugar, a Deus, que me trouxe aqui e me deu a força necessária para vencer os desafios.

Aos meus pais, pelo amor e apoio durante toda a minha vida.

À minha esposa e ao meu filho, pelo carinho, pela paciência e pelos bons momentos que passamos juntos em Campinas.

Ao meu sogro e minha sogra, pela ajuda que nos deram quando precisamos.

A toda a minha família, e aos meus amigos (os de Goiânia e os que conheci aqui), que fazem com a vida seja uma festa.

Aos professores do Instituto de Computação, especialmente aos professores Geovane Cayres Magalhães, meu orientador, Claudia Bauzer Medeiros e Mário Nascimento, com quem aprendi muito, e a quem considero como amigos.

Aos funcionários do Instituto de Computação, pelo pronto atendimento.

A Juliano Lopes de Oliveira, que me apoiou e guiou quando cheguei à Unicamp.

A Frederico Sidney Cox Junior, que com presteza me forneceu as implementações dos métodos de acesso.

À Telegoiás, por ter possibilitado minha estada aqui.

Ao Centro de Pesquisa e Desenvolvimento da TELEBRÁS, pelo apoio a este trabalho.

Finalmente, a todas as pessoas que, mesmo sem saberem (ou mesmo sem que eu soubesse), me ajudaram a concretizar esse sonho.

# Conteúdo

**Prefácio** vii

**Abstract** viii

**Agradecimentos** ix

**1 Introdução** 1

1.1 Sistemas de informação geográfica . . . . . 2

1.1.1 Aplicações de sistemas de informação geográfica . . . . . 2

1.1.2 Modelos de dados em SIGs . . . . . 3

1.2 Comparação de desempenho de métodos de acesso espaciais . . . . . 3

1.2.1 O trabalho de Cox . . . . . 4

1.2.2 Motivação do trabalho . . . . . 5

1.3 Organização da dissertação . . . . . 6

**2 Métodos de Acesso Espaciais — técnicas de construção, classificação e funcionamento** 8

2.1 Introdução . . . . . 8

2.2 Abstração de objetos espaciais . . . . . 9

2.3 Classificação dos Métodos de Acesso Espaciais . . . . . 13

2.3.1 Classificação baseada na dimensão dos dados . . . . . 14

2.3.2 Classificação baseada nos métodos de acesso convencionais de origem 22

2.4 Métodos que utilizam índices convencionais . . . . . 24

2.4.1 Transformação através de *space-filling curves* . . . . . 24

2.4.2 DOT . . . . . 33

2.4.3 2dMAP21 . . . . . 35

2.4.4 G-tree . . . . . 38

2.4.5 Filter Tree . . . . . 41

2.5 Métodos derivados de árvores binárias . . . . . 46

2.5.1 Point Quadtree . . . . . 47

2.5.2 K-d tree . . . . . 51
2.6 Métodos derivados de estruturas *hash* . . . . . 54
2.6.1 Grid File . . . . . 55
2.7 Métodos derivados de árvores multiárias . . . . . 60
2.7.1 R-tree . . . . . 61
2.7.2 $R^+$-tree . . . . . 67
2.7.3 $R^*$-tree . . . . . 72
2.7.4 Hilbert R-tree . . . . . 74
2.7.5 R-trees compactadas . . . . . 77
2.7.6 Outras pesquisas sobre as R-trees . . . . . 79

**3 Dados e Consultas** **81**
3.1 Introdução . . . . . 81
3.2 Origem dos dados . . . . . 81
3.3 Conversão dos dados . . . . . 82
3.4 Conjunto de MBRs utilizado nos testes . . . . . 84
3.5 Consultas . . . . . 86

**4 Comparação de desempenho** **96**
4.1 Introdução . . . . . 96
4.2 Descrição dos testes . . . . . 96
4.2.1 Configuração dos métodos de acesso . . . . . 96
4.2.2 Critérios de comparação . . . . . 98
4.2.3 Tipos de consulta . . . . . 98
4.2.4 Atualizações . . . . . 98
4.3 Alterações na implementação . . . . . 99
4.3.1 $R^*$-tree . . . . . 100
4.3.2 $R^+$-tree . . . . . 102
4.4 Apresentação e avaliação dos resultados . . . . . 103
4.4.1 Inserções e remoções . . . . . 105
4.4.2 *Point queries* . . . . . 114
4.4.3 *Range queries* para determinação de intersecção . . . . . 118
4.4.4 *Range queries* para determinação dos retângulos que incluem a janela de consulta . . . . . 124
4.4.5 Conclusões . . . . . 129
4.5 Comparação com os resultados de Cox . . . . . 129

**5 Conclusões e extensões** **131**
5.1 Conclusões . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 131
5.2 Extensões . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132

**Bibliografia** **134**

# Lista de Tabelas


2.1 Classificação dos métodos de acesso a pontos de acordo com as propriedades das regiões em que o espaço é dividido. . . . . . . . . . . . . . . . . . . . . . 16

2.2 Código binário Gray refletido de 4 bits. . . . . . . . . . . . . . . . . . . . . . 29

3.1 Número de programas de conversão por grupo de elementos. . . . . . . . . 83

3.2 Elementos convertidos dos dados de Valinhos. Grupos: MUB – mapeamento urbano básico; RA – rede aérea; CS – canalização subterrânea; ET – estações telefônicas; LGR – limites de gerência de rede. . . . . . . . . . . 89

3.3 Elementos não existentes nos dados de Valinhos para os quais foram codificados programas de conversão. Grupos: MUB – mapeamento urbano básico; RA – rede aérea. . . . . . . . . . . . . . . . . . . . . . . . . . . 90

3.4 Número de ocorrências dos elementos representados como pontos. . . . . . 90

3.5 Número de ocorrências dos elementos representados como linhas poligonais. As áreas se referem aos seus MBRs. . . . . . . . . . . . . . . . . . . . . . 91

3.6 Distribuição dos tamanhos das áreas dos MBRs em relação à área total (elementos representados como linhas poligonais). . . . . . . . . . . . . . . 91

# Lista de Figuras


2.1 Abstração de objetos através de retângulos envolventes mínimos. . . . . . . 9
2.2 Dois MBRs que se intersectam, apesar dos objetos representados serem disjuntos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.3 Intersecção dos MBRs de dois objetos que se tocam. . . . . . . . . . . . . 10
2.4 Aproximações da representação completa de objetos espaciais: (a) retângulo envolvente mínimo, (b) retângulo envolvente mínimo rotacionado, (c) casco convexo, (d) círculo envolvente mínimo e (e) elipse envolvente mínima. . . 11
2.5 Aproximações progressivas: (a) círculo incluso máximo, (b) retângulo incluso máximo, (c) segmentos de linha inclusos máximos e (d) combinação do retângulo e dos segmentos de linha inclusos máximos. . . . . . . . . . 12
2.6 Decomposição de um objeto em DBMR's para $g = 2$. As linhas pontilhadas indicam os eixos de partição. . . . . . . . . . . . . . . . . . . . . . . . 12
2.7 Outras técnicas de decomposição: (a) regular, (b) estrutural (divisão do polígono em trapezóides aproximados por MBRs). . . . . . . . . . . . . . 13
2.8 Divisão de uma região ocasionada pela inserção de um ponto. . . . . . . 14
2.9 Aumento da densidade máxima com a inclusão de um retângulo. . . . . . 18
2.10 Esquemas de representação de um objeto espacial na técnica de abstração: (a) representação por vértices e (b) representação central. . . . . . . . . . 19
2.11 Duas consultas onde objetos que intersectam as janelas de consulta (S1 e S2) são descartados na fase de filtragem devido à representação por vértices (a) e central (b). . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.12 Representação de um objeto espacial por um número maior de pontos: (a) representação por vértices e (b) representação central. (c) e (d) mostram o resultado se somente as células que intersectam o objeto forem consideradas. 21
2.13 Divisão de um objeto espacial baseada no tamanho da janela de consulta mínima. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.14 Separação de MBRs em índices diferentes de acordo com seus tamanhos. . 22
2.15 Técnicas utilizadas para derivar métodos de acesso a objetos de dimensão não zero a partir de métodos de acesso a pontos. . . . . . . . . . . . . . . 23
2.16 (a) Percurso por colunas e (b) percurso por colunas com sentidos alternados. 25

2.17 Coordenadas lineares das células de uma grade 4 × 4 geradas pela intercalação dos bits das coordenadas em cada dimensão. . . . . . . . . . . . . . 26
2.18 Curvas z de ordem 1, 2 e 3. . . . . . . . . . . . . . . . . . . . . . . . . . 27
2.19 Códigos Gray das células de uma grade 4 × 4, gerados pela intercalação dos bits dos códigos Gray das coordenadas de cada dimensão (a), e seus equivalentes binários (b). . . . . . . . . . . . . . . . . . . . . . . . . . 28
2.20 Curvas de código Gray de ordem 1, 2 e 3. . . . . . . . . . . . . . . . . . 29
2.21 Curvas de Hilbert de ordem 1, 2 e 3. . . . . . . . . . . . . . . . . . . . . 30
2.22 Alguns agrupamentos permitidos (a) e não permitidos (b) numa grade com células ordenadas por uma curva z. . . . . . . . . . . . . . . . . . . . . . 31
2.23 Processo de representação de um polígono através de coordenadas lineares em uma curva z. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
2.24 Transformação de segmentos de reta e de uma janela de consulta em um espaço unidimensional (a) para um espaço bidimensional (b). . . . . . . . . 34
2.25 Um conjunto de MBRs e suas projeções em cada eixo. . . . . . . . . . . . 36
2.26 Indexação dos retângulos da figura anterior através de duas árvores MAP21. 37
2.27 Partição do espaço na G-tree. . . . . . . . . . . . . . . . . . . . . . . . . 38
2.28 Cálculo dos níveis de um conjunto de retângulos. . . . . . . . . . . . . . 42
2.29 Estrutura de uma Filter Tree. . . . . . . . . . . . . . . . . . . . . . . . . 43
2.30 Junção espacial em uma Filter Tree. . . . . . . . . . . . . . . . . . . . . 45
2.31 Agrupamento de nós de uma árvore binária em páginas de disco. . . . . . . 47
2.32 Inserção de pontos em uma point quadtree. . . . . . . . . . . . . . . . . . 49
2.33 Área onde se localizam os nós que devem ser reinseridos com a exclusão do nó $A$ e sua substituição pelo nó $H$. . . . . . . . . . . . . . . . . . . . . . 50
2.34 Dois casos onde o primeiro critério de escolha entre os candidatos falha: (a) nenhum nó satisfaz o critério; (b) dois nós satisfazem o critério. . . . . 51
2.35 Inserção de pontos em uma k-d tree. A sub-árvore da direita corresponde a $LOSON$, e a da esquerda a $HISON$. . . . . . . . . . . . . . . . . . . . 53
2.36 Estrutura de um grid file. . . . . . . . . . . . . . . . . . . . . . . . . . . 56
2.37 Tratamento de *overflow* em um Grid File com capacidade máxima de 3 pontos por bloco. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
2.38 Diretório de um Grid File onde o tamanho das células é fixo. . . . . . . . . 58
2.39 A região em destaque na primeira grade representa um bloco que deve sofrer uma junção. As demais grades mostram as alterações permitidas nos sistemas *buddy* e *neighbor*. . . . . . . . . . . . . . . . . . . . . . . . 59
2.40 Exclusão de pontos em um Grid File com junção de blocos. . . . . . . . . . 60
2.41 Uma R-tree (a) e os retângulos e pontos indexados (b). . . . . . . . . . . 63

2.42 Duas *range queries* sobre um conjunto de retângulos e pontos (a) e os caminhos percorridos na R-tree para identificação dos retângulos que intersectam $j_1$ (b). . . . . 66
2.43 Uma R$^+$-tree (a) e os retângulos e pontos indexados (b). . . . . 69
2.44 Três situações especiais na inserção de um retângulo. Os retângulos tracejados representam os MBRs das entradas de um nó. . . . . 70
2.45 Um conjunto de entradas de um nó intermediário (retângulos tracejados) (a) e a divisão do espaço através de MaxRects (b) e (c). . . . . 70
2.46 Propagação de um split para os níveis inferiores em um nó de uma R$^+$-tree. 71
2.47 Um conjunto de retângulos indexados por uma Hilbert R-tree. . . . . 76
2.48 Estrutura correspondente aos retângulos da figura anterior. . . . . 76

3.1 Dois logradouros, cada um composto por duas linhas centrais. . . . . 84
3.2 MBRs dos dados de Valinhos. . . . . 85
3.3 Localização dos postes de Valinhos. . . . . 86
3.4 MBRs das quadras de Valinhos. . . . . 87
3.5 Janelas de consulta com 0,0001% da área total. . . . . 92
3.6 Janelas de consulta com 0,001% da área total. . . . . 92
3.7 Janelas de consulta com 0,01% da área total. . . . . 93
3.8 Janelas de consulta com 0,1% da área total. . . . . 93
3.9 Janelas de consulta com 1% da área total. . . . . 94
3.10 Janelas de consulta com 10% da área total. . . . . 94
3.11 Localização dos pontos utilizados nas *point queries*. . . . . 95

4.1 Comparação do desempenho dos dois algoritmos de determinação, na R$^+$-tree, dos retângulos que incluem uma janela de consulta. . . . . 104
4.2 Número de páginas acessadas na inserção dos dados completos de Valinhos (66.837 objetos). . . . . 106
4.3 Número de páginas armazenadas em disco — fase 1 — dados completos de Valinhos. . . . . 106
4.4 Ocupação média dos nós — fase 1 — dados completos de Valinhos. . . . . 107
4.5 Número de páginas acessadas em disco na exclusão de metade dos objetos (33.418) — dados completos de Valinhos. . . . . 107
4.6 Número de páginas armazenadas em disco — fase 2 — dados completos de Valinhos. . . . . 108
4.7 Ocupação média dos nós — fase 2 — dados completos de Valinhos. . . . . 108
4.8 Número de páginas acessadas em disco na inclusão de 10.000 objetos e exclusão de outros 10.000 — dados completos de Valinhos. . . . . 109

4.9 Número de páginas armazenadas em disco — fase 3 — dados completos de Valinhos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 109
4.10 Ocupação média dos nós — fase 3 — dados completos de Valinhos. . . . . 110
4.11 Número de páginas acessadas na inserção dos postes (13.813 objetos). . . . 110
4.12 Número de páginas armazenadas em disco — fase 1 — postes. . . . . . . . 111
4.13 Ocupação média dos nós — fase 1 — postes. . . . . . . . . . . . . . . . . . . 111
4.14 Número de páginas acessadas na inserção das quadras (2.473 objetos). . . . 112
4.15 Número de páginas armazenadas em disco — fase 1 — quadras. . . . . . . 112
4.16 Ocupação média dos nós — fase 1 — quadras. . . . . . . . . . . . . . . . . . 113
4.17 *Point queries* — fase 1 — dados completos de Valinhos. . . . . . . . . . . . 115
4.18 *Point queries* — fase 2 — dados completos de Valinhos. . . . . . . . . . . . 115
4.19 *Point queries* — fase 3 — dados completos de Valinhos. . . . . . . . . . . . 116
4.20 *Point queries* — fase 1 — postes. . . . . . . . . . . . . . . . . . . . . . . . 116
4.21 *Point queries* — fase 1 — quadras. . . . . . . . . . . . . . . . . . . . . . . . 117
4.22 *Range queries* para determinação de intersecção — fase 1 — dados completos de Valinhos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 119
4.23 *Range queries* para determinação de intersecção (apenas as variantes da R*-tree) — fase 1 — dados completos de Valinhos. . . . . . . . . . . . . . 119
4.24 *Range queries* para determinação de intersecção — fase 2 — dados completos de Valinhos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120
4.25 *Range queries* para determinação de intersecção (apenas as variantes da R*-tree) — fase 2 — dados completos de Valinhos. . . . . . . . . . . . . . 120
4.26 *Range queries* para determinação de intersecção — fase 3 — dados completos de Valinhos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
4.27 *Range queries* para determinação de intersecção (apenas as variantes da R*-tree) — fase 3 — dados completos de Valinhos. . . . . . . . . . . . . . 121
4.28 *Range queries* para determinação de intersecção — fase 1 — postes. . . . . 122
4.29 *Range queries* para determinação de intersecção (apenas as variantes da R*-tree) — fase 1 — postes. . . . . . . . . . . . . . . . . . . . . . . . . . 122
4.30 *Range queries* para determinação de intersecção — fase 1 — quadras. . . . 123
4.31 *Range queries* para determinação de intersecção (apenas as variantes da R*-tree) — fase 1 — quadras. . . . . . . . . . . . . . . . . . . . . . . . . 123
4.32 *Range queries* para determinação dos retângulos que incluem a janela de consulta — fase 1 — dados completos de Valinhos. . . . . . . . . . . . . . 125
4.33 *Range queries* para determinação dos retângulos que incluem a janela de consulta (apenas as variantes da R*-tree) — fase 1 — dados completos de Valinhos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 125

4.34 *Range queries* para determinação dos retângulos que incluem a janela de consulta — fase 2 — dados completos de Valinhos. . . . . . . . . . . . . . 126
4.35 *Range queries* para determinação dos retângulos que incluem a janela de consulta (apenas as variantes da R*-tree) — fase 2 — dados completos de Valinhos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 126
4.36 *Range queries* para determinação dos retângulos que incluem a janela de consulta — fase 3 — dados completos de Valinhos. . . . . . . . . . . . . . 127
4.37 *Range queries* para determinação dos retângulos que incluem a janela de consulta (apenas as variantes da R*-tree) — fase 3 — dados completos de Valinhos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127
4.38 *Range queries* para determinação dos retângulos que incluem a janela de consulta — fase 1 — quadras. . . . . . . . . . . . . . . . . . . . . . . . . 128
4.39 *Range queries* para determinação dos retângulos que incluem a janela de consulta (apenas as variantes da R*-tree) — fase 1 — quadras. . . . . . . . 128

# Capítulo 1

# Introdução

*Métodos de acesso*, também conhecidos como *índices*, são associações de estruturas e algoritmos que provêem um caminho otimizado aos dados, com base em predicados avaliados sobre um conjunto de atributos [Cox91]. No caso dos *dados espaciais*, esses atributos descrevem a localização de objetos ou fenômenos num espaço $k$-dimensional, onde $k > 1$, e muitas vezes também suas formas; os predicados se referem a relacionamentos como adjacência, intersecção, sobreposição e distância entre essas entidades. Entre as aplicações que manipulam esses tipos de dados estão as aplicações construídas sobre os sistemas de informação geográfica (SIGs), os sistemas de projeto e fabricação auxiliados por computador (CAD/CAM) e as aplicações de processamento analítico *on-line* (OLAP).

Embora a indexação de dados espaciais seja objeto de estudo desde a década de 70 [FB74, Ben75], somente agora alguns sistemas gerenciadores de bancos de dados comerciais começam a incorporá-los, juntamente com outros recursos, para dar suporte às aplicações espaciais. Contribuíram para essa evolução o crescimento da capacidade de processamento e de armazenamento e a queda dos custos das estações de trabalho, assim como o aumento da demanda por essas aplicações. Por outro lado, existem fatores que ainda dificultam a inclusão de índices espaciais nos SGBDs, como a ausência de algoritmos de controle de concorrência e recuperação de falhas para a maioria deles e a falta de uma melhor definição dos métodos de acesso mais apropriados para cada tipo de aplicação.

O interesse desta dissertação se concentra na indexação de dados pertencentes a aplicações geográficas urbanas, especialmente aquelas voltadas para o gerenciamento de serviços de utilidade pública, tais como telefonia, eletricidade, água, esgoto, gás e televisão a cabo. A partir de um conjunto de dados reais extraídos de uma aplicação geográfica para controle de redes de telefonia, faz-se aqui uma comparação do desempenho de um grupo de métodos de acesso espaciais. Embora os dados sejam provenientes de uma aplicação específica, sua utilidade reside no fato deles possuírem características comuns aos dados de outras aplicações de gerenciamento de serviços de utilidade pública, tais como tipos,

tamanhos e distribuição dos objetos representados. Como exemplo, pode-se citar as redes de cabos telefônicos, as redes de dutos de água e as redes de energia elétrica, que têm representações e distribuições semelhantes. Isso permite que os resultados deste trabalho sejam estendidos a tais aplicações.

## 1.1 Sistemas de informação geográfica

*Sistemas de informação geográfica* (SIGs) são sistemas automatizados usados para armazenar, analisar e manipular dados georeferenciados, ou seja, dados que representam objetos e fenômenos em que a localização geográfica é uma característica inerente à informação e indispensável para analisá-la [CCH+96]. Os SIGs fornecem o ambiente necessário à construção de aplicações geográficas, através de facilidades como:

- entrada e integração de dados provenientes de fontes heterogêneas, como fotos aéreas, imagens de satélites e digitalização de mapas;
- linguagem e funções para manipulação tanto dos dados convencionais como dos espaciais;
- interface com o usuário com capacidade de apresentação gráfica dos dados espaciais e de realizar operações sobre esses dados a partir de ações do usuário sobre as apresentações;
- armazenamento das informações em um banco de dados, garantindo persistência, segurança, concorrência no acesso e eficiência tanto na recuperação como na atualização dos dados.

### 1.1.1 Aplicações de sistemas de informação geográfica

Há uma vasta gama de aplicações geográficas. Em [CCH+96] elas são classificadas como:

- *sócio-econômicas*, subdivididas em:
  - *aplicações de uso da terra*, incluindo cadastros rurais, agroindústria e irrigação;
  - *ocupação humana*, envolvendo cadastros urbanos e regionais;
  - *sistemas para gerenciamento de serviços de utilidade pública*, para o controle de redes de telefonia, eletricidade, esgotos, transportes, etc.
- *ambientais*, que podem ser separadas em dois grupos:

  - *meio ambiente*, incluindo ecologia, clima, gerenciamento florestal e poluição;
  - *uso dos recursos naturais*, envolvendo extrativismo vegetal, extrativismo mineral, energia, recursos hídricos e oceânicos.

- *gerenciamento*: envolve a realização de estudos e projeções que determinam onde e como alocar recursos para remediar problemas ou garantir a preservação de determinadas características. Exemplos dessas aplicações são o planejamento de tráfego urbano, o planejamento e controle de obras públicas e o planejamento da defesa civil.

### 1.1.2 Modelos de dados em SIGs

Nas aplicações geográficas, o mundo real é modelado de acordo com duas concepções distintas: o modelo de *campos* e o modelo de *objetos*.

No modelo de campos, uma região geográfica é vista como uma superfície contínua, e o que se denomina como "campo" (também chamado de *geocampo*) é uma função matemática que modela um fenômeno qualquer naquela região (tipo de vegetação, altitude, distribuição de casos de dengue, etc.), e que tem como domínio o conjunto de pontos da região e como contradomínio o conjunto de valores que o fenômeno pode assumir.

No modelo de objetos, uma região geográfica é representada como uma superfície ocupada por objetos com identidade, geometria e atributos próprios. Como exemplo, se considerarmos uma região pertencente a uma cidade, alguns objetos (também chamados de *geo-objetos*) poderiam ser: um lote, uma quadra, um poste, um lago ou uma rua. Se tomarmos uma região correspondente a um país, exemplos de objetos poderiam ser: uma cidade, uma rodovia, um rio ou um estado.

As aplicações de gerenciamento de serviços de utilidade pública utilizam o modelo de objetos, de forma que esta dissertação se restringirá ao estudo da indexação das representações de geo-objetos.

## 1.2 Comparação de desempenho de métodos de acesso espaciais

Um grande número de trabalhos apresentam algum tipo de comparação de desempenho entre métodos de acesso espaciais. Normalmente, a cada vez que um novo índice é proposto, ele é comparado com um ou mais índices de conhecimento geral. Além disso, existem algumas pesquisas dedicadas exclusivamente a essas comparações, como [Gre89], [KSSS89] e [Cox91].

### 1.2.1 O trabalho de Cox

O trabalho de Cox é de especial interesse para esta dissertação, pois os métodos de acesso aqui avaliados foram implementados por ele. Além disso, seus testes de desempenho foram feitos unicamente a partir de dados sintéticos, o que torna útil uma comparação dos resultados das duas pesquisas.

Em seu estudo, Cox identifica alguns fatores, que ele chama de *determinantes de desempenho* dos métodos de acesso espaciais, enumerados a seguir:

- os tipos dos dados, que podem ser pontos ou objetos de dimensão não zero, estes normalmente representados nos métodos de acesso por retângulos;
- os tamanhos dos objetos indexados em relação à área total considerada;
- a distribuição dos objetos no espaço, que pode ser uniforme ou não;
- a dinâmica dos dados, que se refere à freqüência com que eles são atualizados, o que os classifica como dados dinâmicos ou estáticos;
- os tipos de consultas realizadas sobre os dados.

Cox divide as consultas espaciais em duas categorias:

- *point queries*, que visam determinar os objetos que contêm um determinado ponto no espaço;
- *range queries*, cujo objetivo é determinar todos os objetos que satisfazem a um determinado relacionamento sobre uma área de pesquisa (também chamada de *janela de consulta*). Esses relacionamentos podem ser:
  - intersecção, onde os objetos relevantes são aqueles que possuem pontos em comum com a janela;
  - inclusão, que se refere aos objetos que contêm a janela;
  - não-inclusão, referente aos objetos contidos na janela.

Como a expressão "*range queries* para determinação de inclusão" pode deixar margem a dúvidas com relação aos objetos que se deseja recuperar (os que incluem a janela de consulta ou os que estão incluídos nela), preferimos nos referir a esse tipo de consulta simplesmente como "*range queries* para determinação dos objetos que contêm a janela de consulta", substituindo também a expressão "*range queries* para determinação de não-inclusão" por "*range queries* para determinação dos objetos contidos na janela de consulta".

Cox deixou de incluir nessa classificação um tipo de consulta espacial importante, e que tem sido motivo de várias pesquisas devido ao esforço computacional que requerem para serem avaliadas: as *junções espaciais*. Nessa operação, dados dois conjuntos de objetos espaciais, deseja-se recuperar todos os pares de objetos (um de cada conjunto) que satisfaçam a um determinado relacionamento. Por exemplo, dados um conjunto de rios e um conjunto rodovias, deseja-se saber quais são os pares (rio, rodovia) que se intersectam. Uma junção espacial pode ser feita também entre objetos de um mesmo conjunto. Por exemplo, pode-se perguntar quais são os pares (rodovia, rodovia) que se intersectam. Não serão incluídos testes sobre junções espaciais neste trabalho também, pois esse assunto é suficientemente vasto para uma pesquisa independente (para alguns métodos de acesso há vários algoritmos diferentes de junção espacial).

Um fator que não foi citado, mas que também afeta o desempenho das consultas é o tamanho das janelas.

Cox utilizou em seus experimentos 14 arquivos de dados, 7 com distribuição uniforme e 7 com distribuição não uniforme. Para cada tipo de distribuição ele gerou um arquivo contendo pontos, um com retângulos pequenos, um contendo retângulos grandes, e nos demais ele combinou esses três tipos de dados. Sobre as estruturas geradas a partir desses arquivos foram executadas *point queries* e *range queries* para determinação de intersecção e para recuperação dos objetos que continham a janela de consulta. Também foram feitas exclusões e inclusões nos arquivos de dados, com posterior reexecução das consultas para verificar o comportamento dos índices com relação à dinâmica dos dados.

### 1.2.2 Motivação do trabalho

Enquanto a maioria das pesquisas sobre métodos de acesso espaciais utiliza dados sintéticos em seus testes, poucas usam dados reais representativos para aplicações geográficas. Dois motivos justificam o uso de dados sintéticos. Em primeiro lugar, eles são mais fáceis de se conseguir, pois o processo de obtenção dos dados reais é caro e nem sempre as organizações que os detêm os colocam à disposição de pesquisadores. Em segundo lugar, o uso de dados sintéticos permite o controle da variação dos fatores que afetam o desempenho dos métodos de acesso.

No entanto, o uso de dados sintéticos tem como inconveniente a dificuldade de se dizer se uma determinada combinação de fatores existe mesmo ou pelo menos se aproxima de alguma situação real. Por exemplo, quase todos os trabalhos utilizam dados distribuídos uniformemente no espaço considerado, o que não é uma situação típica em conjuntos de dados reais [FK94] e nem nos dados manipulados por aplicações geográficas, em particular [Fra91]. Para contornar esse problema, alguns conjuntos de dados são gerados com distribuições não uniformes que, no entanto, também são atípicas, como conjuntos de pontos ou retângulos dispostos sobre uma linha horizontal ou diagonal, ou ainda sobre

uma senóide [KSSS89, Cox91].

Por outro lado, alguns conjuntos de dados reais não têm qualquer relação com aplicações que não sejam aquelas de onde foram extraídos. Retângulos pertencentes a *layouts* de circuitos integrados [Gut84, LLE97], pontos modelando o fluxo de ar em torno da asa de um avião [LLE97] e curvas de nível [KSSS89] são exemplos de conjuntos de dados usados na comparação de métodos de acesso espaciais.

Uma fonte de dados geográficos reais muito utilizada por pesquisadores nos testes de índices espaciais é o sistema TIGER *(Topologically Integrated Geographic Encoding and Referencing)* do *U. S. Bureau of Census*, de onde são extraídos dados sobre rodovias de municípios norte americanos. Mas como foi notado em [SK96], apesar desses dados serem reais é improvável que sejam típicos, por apresentarem um baixo grau de sobreposição entre os objetos.

Em tal cenário não se pode afirmar com certeza se os resultados obtidos em uma bateria de testes são extensíveis a um determinado tipo de aplicação, de forma que se torna importante utilizar nos testes de desempenho de métodos de acesso conjuntos de dados representativos para as aplicações onde se pretende empregá-los. Nesse sentido, esta dissertação traz duas contribuições:

- a conversão de um conjunto de dados reais representativos para aplicações de gerenciamento de serviços de utilidade pública para um formato em que ele pode ser facilmente repassado a outros pesquisadores;
- a avaliação do desempenho de um grupo de métodos de acesso espaciais na manipulação desse conjunto de dados.

Os métodos de acesso testados pertencem à família da R-tree [Gut84] e estão descritos nas seções 2.7.1 a 2.7.3. Uma das variantes da R*-tree [BKSS90] (seção 2.7.3) se apresentou, durante os testes, como a melhor opção entre os métodos avaliados para indexação dos dados utilizados, considerando as condições de operação da aplicação de onde eles foram extraídos. A confrontação dos resultados dessa variante com os de outra, também descrita em [BKSS90] contrariaram a comparação que se fez das duas naquele trabalho, que usou dados sintéticos em seus experimentos.

## 1.3 Organização da dissertação

A dissertação está organizada da forma que se segue.

O capítulo 2 traz uma visão geral sobre as pesquisas desenvolvidas na área dos métodos de acesso espaciais, procurando, particularmente, descrever as propostas dos trabalhos mais recentes.

O capítulo 3 descreve as características dos dados utilizados nos experimentos e o processo de sua obtenção. Nele também são definidas as consultas executadas sobre esses dados.

O capítulo 4 detalha a preparação e a execução dos testes de desempenho, apresenta os resultados obtidos e compara o comportamento dos métodos de acesso avaliados. Ele também relata problemas encontrados na implementação de alguns desses métodos de acesso e a forma como foram solucionados.

O capítulo 5 traz as conclusões do trabalho, suas contribuições e possíveis extensões.

# Capítulo 2

# Métodos de Acesso Espaciais — técnicas de construção, classificação e funcionamento

## 2.1 Introdução

Este capítulo traz uma visão geral sobre as pesquisas desenvolvidas na área dos métodos de acesso espaciais. Ele não pretende ser uma compilação completa, mas antes mostrar as técnicas e idéias utilizadas na construção desses métodos de acesso, assim como as principais características de cada família de métodos. Procurou-se, particularmente, descrever as propostas dos trabalhos mais recentes. Embora haja alguns métodos de acesso desenvolvidos especificamente para espaços de alta dimensionalidade (dez dimensões por exemplo), estes não foram considerados, por atenderem necessidades diferentes daquelas encontradas em um SIG.

O capítulo está organizado da forma que se segue. A seção 2.2 trata das simplificações usadas para representar os objetos no processo de indexação. A seção 2.3 discute as diversas formas de classificação propostas para os métodos de acesso espaciais. As seções 2.4, 2.5, 2.6 e 2.7 descrevem, a partir de uma das classificações apresentadas na seção 2.3, a estrutura e o modo de operação de alguns representantes de cada família. A descrição de alguns métodos foi baseada nos artigos originais, embora outras revisões também tenham servido como referência [Ooi90, Cox91, OSH93, Sam95, CCH+96].

## 2.2 Abstração de objetos espaciais

Freqüentemente os objetos indexados pelos métodos de acesso espaciais são representados como linhas e polígonos com grande quantidade de pontos. Apesar de existirem estruturas específicas para o armazenamento e recuperação da geometria completa dos objetos espaciais [SK91, MCD94, Med95, MCG96], seria excessivamente dispendioso usar essa geometria no processo de indexação, devido ao tempo gasto para avaliação dos predicados e ao espaço de armazenamento que os métodos de acesso ocupariam. Assim, os índices espaciais utilizam certas simplificações da geometria dos objetos, tomando o cuidado de preservar suas propriedades geométricas essenciais, ou seja, a posição no espaço e a extensão em cada dimensão. As aproximações que garantem a manutenção dessas propriedades são chamadas de *conservativas*.

A abstração mais usada pelos métodos de acesso espaciais é o *retângulo envolvente mínimo (REM)*, usualmente chamado de *MBR* (de *minimum bounding rectangle*). Dado um objeto representado em um espaço $k$-dimensional, seu MBR é o menor retângulo $k$-dimensional que o contém e cujos lados são paralelos aos eixos de cada dimensão. A figura 2.1 mostra um conjunto de objetos num espaço bidimensional e seus respectivos MBRs.

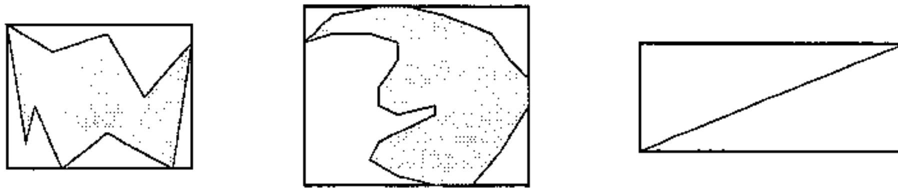


Figura 2.1: Abstração de objetos através de retângulos envolventes mínimos.

As duas grandes vantagens do uso do MBR são o pequeno espaço necessário para armazená-lo, o que pode ser feito com as coordenadas de apenas um ponto para cada dimensão, e a facilidade com que são avaliados predicados espaciais sobre retângulos. Pelo fato de se tratarem de aproximações conservativas, alguns relacionamentos entre MBRs podem ser deduzidos de relacionamentos entre os objetos que eles representam. Tomemos, como exemplo, o predicado *"intersecta"*. Se esse predicado é verdadeiro entre dois objetos, também o é entre os MBRs que os representam. A mesma observação vale para outros predicados, como, por exemplo, "contém" e "está contido". Entretanto a recíproca pode não ser verdadeira, ou seja, uma vez determinada a existência de um relacionamento entre dois MBRs não se pode afirmar a priori a sua existência entre os objetos correspondentes. Por exemplo, dados dois MBRs que se intersectam, não necessariamente os objetos representados se intersectam também (veja a figura 2.2). Isto se deve ao espaço "vago" em cada MBR, isto é, aquele que não pertence ao objeto representado, e que é conhecido como *dead space*.

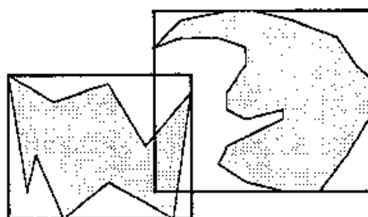


Figura 2.2: Dois MBRs que se intersectam, apesar dos objetos representados serem disjuntos.

A diferença de resultados que se pode obter entre a avaliação de predicados sobre a geometria completa dos objetos e sobre suas abstrações faz com que as consultas espaciais sejam executadas em duas fases:

- **Filtragem:** Nesta etapa é recuperado, através dos métodos de acesso, um conjunto de objetos "candidatos", ou seja, aqueles cujas abstrações satisfazem o predicado envolvido, embora não necessariamente os objetos que representam também o satisfaçam. Apesar de haver um grande número de predicados espaciais [Câm95, Cif95], nesta fase substitui-se predicados específicos por outros de cunho geral, normalmente a existência ou não de intersecção. Por exemplo, se o predicado que está sendo avaliado é *"toca"*, durante a filtragem ele é substituído por *"intersecta"*, uma vez que se dois objetos se tocam é possível que seus MBRs (ou outras abstrações usadas) se intersectem ao invés de se tocar, como mostra a figura 2.3. Se o predicado avaliado durante a filtragem fosse *"toca"*, objetos nessa situação seriam indevidamente descartados.

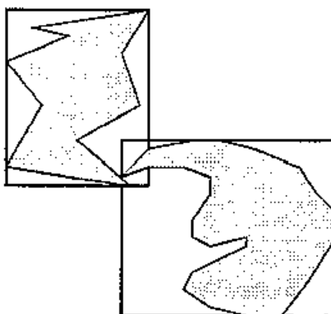


Figura 2.3: Intersecção dos MBRs de dois objetos que se tocam.

- **Refinamento:** Nesta fase verifica-se a geometria completa de cada objeto recuperado na etapa anterior. Apenas os objetos que realmente satisfazem o predicado são devolvidos como resposta.

Uma vez que a recuperação das representações exatas dos objetos e a avaliação de predicados a partir delas são operações dispendiosas, é desejável que o número de objetos verificados durante o refinamento seja o menor possível. Com o intuito de diminuir o número de objetos recuperados durante a filtragem, mas descartados no refinamento (comumente denominados *false hits*), Brinkhoff, Kriegel, Schneider e Seeger [BKSS94, BK94] propõem o uso de outras aproximações à forma dos objetos em adição ao MBR: o *retângulo envolvente mínimo rotacionado*, o *casco convexo*, o *polígono envolvente mínimo com m vértices*, o *círculo envolvente mínimo* e a *elipse envolvente mínima*. A figura 2.4 mostra algumas dessas aproximações.

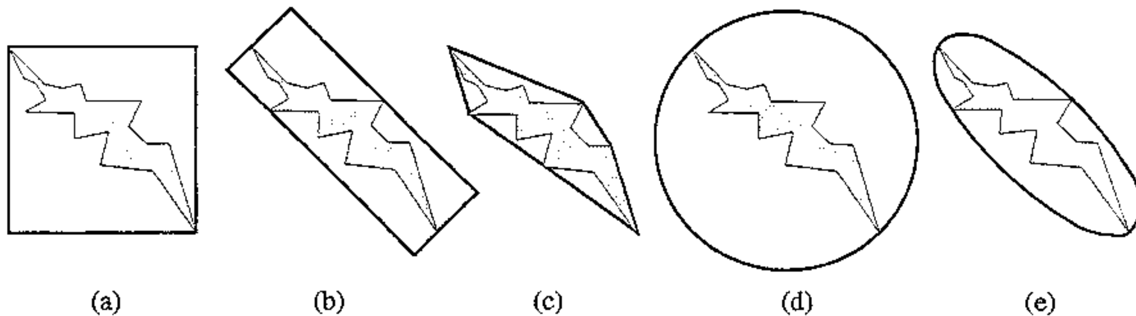



Figura 2.4: Aproximações da representação completa de objetos espaciais: (a) retângulo envolvente mínimo, (b) retângulo envolvente mínimo rotacionado, (c) casco convexo, (d) círculo envolvente mínimo e (e) elipse envolvente mínima.

Apesar dessas representações diminuírem o *dead space* na maioria das vezes, seu uso acarreta uma maior complexidade na verificação dos predicados, e quase todas requerem mais espaço de armazenamento que o MBR. Devido a isso, os autores sugerem que cada objeto seja representado nos métodos de acesso por seu MBR e por uma das abstrações citadas. Essa abstração seria usada em uma nova fase entre a filtragem e o refinamento, chamada de *aproximação*. Nessa etapa, cada objeto recuperado na filtragem (através de seu MBR) teria sua outra abstração verificada com relação ao predicado que se estivesse avaliando, e somente aqueles para os quais ele é satisfeito passariam ao refinamento. Segundo os experimentos apresentados naqueles artigos, o polígono envolvente mínimo com 5 vértices foi a aproximação que mostrou melhor relação custo/benefício (número de pontos armazenados $\times$ número de *false hits* identificados).

Os autores também propõem o uso de *aproximações progressivas* durante a fase de aproximação, para auxiliar na identificação de *false hits*. Uma aproximação progressiva de um objeto é um subconjunto de seus pontos. Decorre daí que se as aproximações progressivas de dois objetos se intersectam, obrigatoriamente os objetos se intersectam também (o mesmo vale para a aproximação progressiva de um objeto e uma janela de consulta). A figura 2.5 mostra algumas dessas aproximações, dentre as quais os autores

apontam a combinação do retângulo incluso máximo com os segmentos de linha inclusos máximos como a representação de melhor desempenho, apesar de requerer maior espaço de armazenamento.

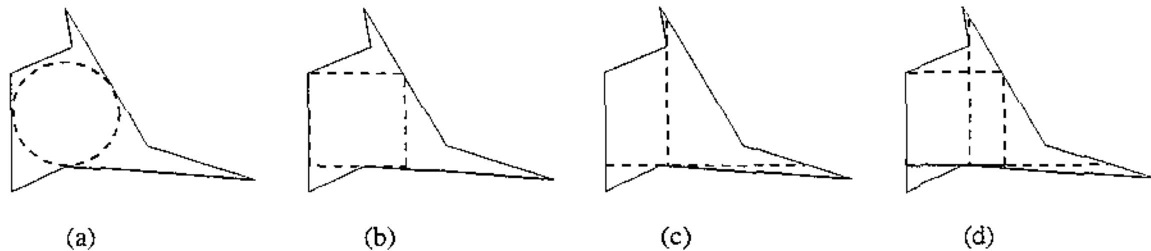



Figura 2.5: Aproximações progressivas: (a) círculo incluso máximo, (b) retângulo incluso máximo, (c) segmentos de linha inclusos máximos e (d) combinação do retângulo e dos segmentos de linha inclusos máximos.

É bom notar que o cálculo das aproximações progressivas é mais dispendioso que o das conservativas, especialmente se o que se procura são as aproximações máximas.

Outra tentativa de melhorar os desempenho dos métodos de acesso espaciais através da diminuição do *dead space* é apresentada em [LLRC96]. Nesse artigo os autores propõem a representação de um objeto através do que eles chamam de *retângulos envolventes mínimos decompostos* (ou *DMBRs* do inglês *decomposed minimum bounding rectangles*). Segundo esta técnica, o polígono (ou linha poligonal) a ser representado é dividido em dois sub-polígonos (ou duas linhas poligonais), correspondentes às sub-regiões esquerda e direita de seu MBR, e um novo MBR (DMBR) é gerado para cada. Essa operação é executada recursivamente, alternando-se os eixos de partição, até que cada DMBR satisfaça a uma dada restrição, que é controlada por um parâmetro $g$. Uma nova subdivisão só é permitida se a área do DBMR a ser dividido for maior que $2^{-g}$ da área do MBR original. A figura 2.6 ilustra o processo para $g = 2$, onde um DMBR só é particionado novamente se sua área for maior que 25% da área do MBR inicial.

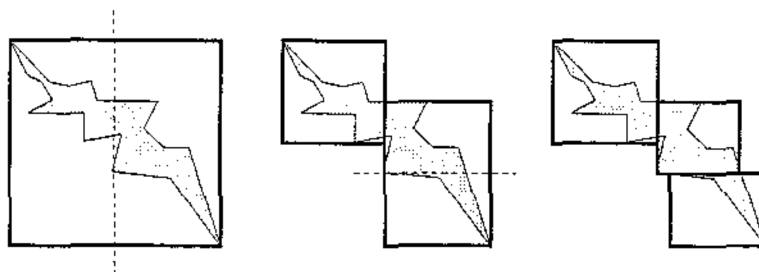


Figura 2.6: Decomposição de um objeto em DBMR's para $g = 2$. As linhas pontilhadas indicam os eixos de partição.

À medida que $g$ cresce, há uma diminuição do *dead space*, mas o número de DMBRs que representam um objeto cresce exponencialmente. Segundo os estudos apresentados em [LLRC96] os melhores desempenhos são obtidos para valores de $g$ entre 3 e 4.

É também interessante citar duas outras técnicas de decomposição usadas na representação de objetos espaciais:

- **Decomposição regular:** Nesta técnica, divide-se, através de uma grade regular, o espaço onde estão dispostos os objetos. Cada objeto é, então representado pelo conjunto de células que ele intersecta.
- **Decomposição estrutural:** Esta técnica é assim denominada porque utiliza o contorno do objeto poligonal para orientar a decomposição. Um exemplo é a utilização dos vértices de um objeto para particioná-lo em trapezóides, que depois poderiam, ou não, ser aproximados por MBRs.

A inconveniência destes dois tipos de decomposição é o grande número de componentes que podem gerar. A figura 2.7 ilustra as duas técnicas.

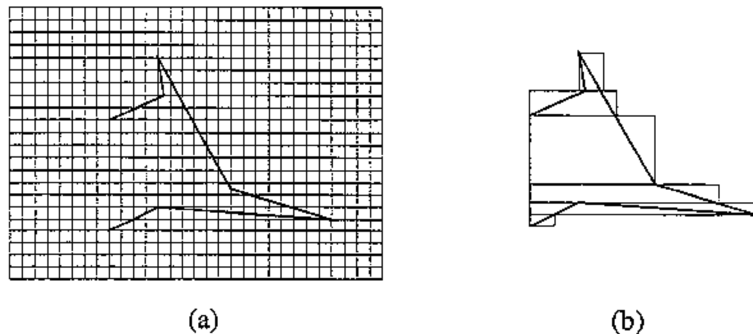



Figura 2.7: Outras técnicas de decomposição: (a) regular, (b) estrutural (divisão do polígono em trapezóides aproximados por MBRs).

Apesar das alternativas propostas, o MBR continua sendo a abstração usada na maioria dos trabalhos sobre indexação espacial, devido à facilidade com que é determinado, à simplicidade com que os predicados podem ser avaliados e ao menor espaço de armazenamento requerido.

## 2.3 Classificação dos Métodos de Acesso Espaciais

Existem, fundamentalmente, dois tipos de classificação para os métodos de acesso espaciais: um que se baseia na dimensão dos dados indexados [SRF87, Ooi90, Med95], e outro nos métodos de acesso convencionais que deram origem aos índices espaciais [Cox91]. [OSH93] apresenta ambas as classificações, que discutiremos a seguir.

### 2.3.1 Classificação baseada na dimensão dos dados

Esta classificação divide os índices espaciais em *métodos de acesso a pontos* e *métodos de acesso a objetos de dimensão não zero*. Alguns autores, como em [KSSS89, Ooi90, OSH93, SLS95], chegam mesmo a classificar como métodos de acesso espaciais apenas os do segundo grupo, se referindo aos do primeiro simplesmente como métodos de acesso a pontos. Como veremos, ambos os grupos sofrem novas subdivisões.

#### Classificação dos métodos de acesso a pontos

A idéia básica do funcionamento dos métodos de acesso a pontos é a divisão do espaço em sub-regiões disjuntas, de forma que cada região contenha, no máximo, um número predeterminado $n$ de pontos. Cada uma dessas sub-regiões é, então, associada a uma página de disco capaz de armazenar os $n$ pontos. Existem outros métodos que fazem a divisão do espaço de acordo com o posicionamento dos pontos que armazenam, e nesses casos cada nó da estrutura contém as coordenadas de apenas um ponto.

A inserção de um novo ponto no índice pode fazer com que seja ultrapassada a capacidade de armazenamento de um nó, causando o particionamento de uma região, num processo conhecido como *split*. Em métodos de acesso que subdividem o espaço de acordo com a localização de cada ponto (ou seja, onde essa capacidade é igual a 1), isso ocorre toda vez que um ponto é inserido. A divisão é feita pela introdução de um ou mais hiperplanos $k-1$-dimensionais no espaço $k$-dimensional onde estão dispostos os pontos indexados, resultando na criação de novas sub-regiões disjuntas.

A figura 2.8 ilustra a divisão de uma região associada a uma página de disco com capacidade máxima de armazenamento igual a quatro pontos, causada pela inserção do ponto $p$. $H$ é o hiperplano inserido (como o espaço em questão é bidimensional, $H$ é uma reta). Cada uma das sub-regiões resultantes é associada a uma página de disco distinta.

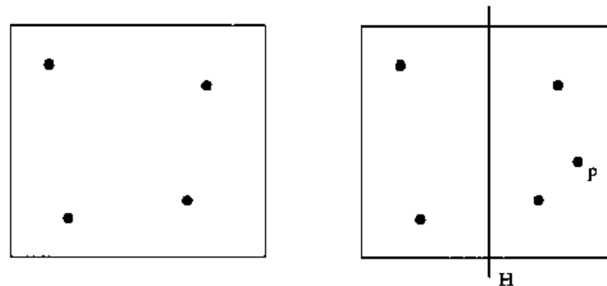



Figura 2.8: Divisão de uma região ocasionada pela inserção de um ponto.

Sellis, Roussopoulos e Faloutsos [SRF87] apresentam três classificações para os métodos de acesso a pontos, baseadas nas características dos algoritmos de *split* usados:

1. Classificação baseada na posição do(s) hiperplano(s) — os métodos de acesso são divididos em dois grupos:
   - Métodos fixos: são aqueles em que a posição do hiperplano é predeterminada. Por exemplo, aqueles em que a região a ser dividida é sempre cortada ao meio.
   - Métodos adaptáveis: são aqueles em que o ponto a ser inserido determina a posição do hiperplano.
2. Classificação baseada no número de dimensões em que são inseridos os hiperplanos — aqui também existem dois grupos:
   - Métodos que particionam o espaço em apenas uma dimensão: o *split* é feito apenas por um hiperplano. Normalmente há um rodízio de dimensões, de forma que se a região a ser dividida foi originada de uma partição na dimensão $k$, então ela é particionada na dimensão $k+1$ (se o espaço é $k$-dimensional, então ela é particionada na primeira dimensão).
   - Métodos que particionam o espaço em $k$ dimensões: neste caso, o espaço $k$-dimensional é dividido em todas as suas dimensões, através da inserção de $k$ hiperplanos.
3. Classificação baseada na localidade da divisão:
   - Métodos de grade: são aqueles em que o hiperplano divide não somente a região afetada, mas todas as regiões em sua direção.
   - Métodos de divisão hierárquica: o hiperplano divide apenas a região que está sofrendo o *split*, o que leva a uma decomposição hierárquica do espaço. Estes métodos foram chamados de *brickwall methods* em [SRF87].

Kriegel, Schiwietz, Schneider e Seeger [KSSS89] sugerem outra classificação, baseada nas propriedades das regiões em que é subdividido o espaço, a saber:

1. se as regiões são disjuntas ou não;
2. se são retangulares ou não;
3. se a união das regiões cobre todo o espaço onde estão dispostos os objetos ou não.

A partir daí, os métodos de acesso são separados em quatro classes, de acordo com a combinação de propriedades que apresentam (veja a tabela 2.1). Segundo os autores, não há métodos de acesso a pontos que se enquadrem nas classes correspondentes às outras combinações.

| | PROPRIEDADE | | |
|---|---|---|---|
| Classe | Regiões retangulares | Cobertura completa | Regiões disjuntas |
| C1 | Sim | Sim | Sim |
| C2 | Sim | Sim | Não |
| C3 | Sim | Não | Sim |
| C4 | Não | Sim | Sim |

Tabela 2.1: Classificação dos métodos de acesso a pontos de acordo com as propriedades das regiões em que o espaço é dividido.

### Classificação dos métodos de acesso a objetos de dimensão não zero

Normalmente, os métodos de acesso a objetos de dimensão não zero são divididos em subclasses de acordo com a técnica utilizada para derivá-los de métodos de acesso a pontos[1]. Embora os nomes dessas subclasses variem de trabalho para trabalho, assim como a forma de agrupá-las, seu significado é o mesmo. Segundo Ooi, Sacks-Davis e Han [OSH93] essas técnicas são:

1. Transformação, subdividida em duas outras:

   - Transformação para um espaço de maior dimensão: Aqui, objetos descritos por $n$ vértices em um espaço $k$-dimensional são mapeados para pontos em um espaço $nk$-dimensional. Por exemplo, um retângulo descrito pelos cantos inferior esquerdo $(x_1, y_1)$ e superior direito $(x_2, y_2)$, é representado como um ponto em um espaço quadridimensional, cujas coordenadas poderiam ser $(x_1, y_1, x_2, y_2)$. Outra alternativa seria usar as coordenadas do centro de cada MBR e seus deslocamentos em cada dimensão em relação ao centro $(c_x, c_y, d_x, d_y)$. Depois da transformação, os pontos gerados podem ser armazenados em métodos (espaciais) de acesso a pontos. Embora esta seja uma técnica simples, pois não requer nenhuma alteração no método base, ela tem a desvantagem de não preservar as distâncias entre os objetos indexados. Isso significa que objetos que são espacialmente próximos no espaço $k$-dimensional podem ficar distantes uns dos outros no espaço $nk$-dimensional, e vice-versa. Como conseqüência, consultas baseadas em intersecção podem se tornar ineficientes.

   - Transformação para o espaço unidimensional: Nesta técnica, o espaço onde os objetos são representados é dividido por uma grade de células do mesmo

[1] Os métodos de acesso a objetos de dimensão não zero têm sua origem em métodos convencionais ou espaciais de acesso a pontos.

tamanho. Estas células são numeradas de acordo com alguma *space-filling curve*, que são linhas poligonais que passam exatamente uma vez no ponto central de cada célula da grade, sem nunca cruzar a si mesma. A ordem em que as células são "visitadas" por essa linha é, então, usada para ordená-las, e cada objeto é representado por um conjunto de números, correspondentes às células que ele intersecta[2]. Esses números são tratados como coordenadas de pontos num espaço unidimensional e indexados através de métodos de acesso convencionais, como a $B^+$-tree, por exemplo. Uma desvantagem dessa técnica é que um objeto pode ser representado por vários conjuntos disjuntos de pontos e, portanto, tem que ser referenciado mais de uma vez no índice.

Existem dois métodos de acesso espaciais que utilizam a técnica da transformação, mas que não se enquadram muito bem em nenhum dos dois procedimentos. O primeiro é o DOT [FR91], onde os dois são combinados. Inicialmente, os MBRs dos objetos são transformados em pontos numa dimensão duas vezes maior, como na primeira técnica. Esses pontos são, depois, mapeados para um espaço unidimensional, através do uso de *space-filling curves*. Dessa forma, cada objeto é representado apenas como um ponto num índice convencional. O segundo método de acesso é a 2dMAP21 [ND96, ND97], que transforma um MBR bidimensional em dois pontos em espaços unidimensionais distintos. Cada ponto corresponde à extensão e localização da projeção do MBR em um dos eixos. A partir daí, são usadas duas $B^+$-trees, cada qual indexando os pontos correspondentes a uma das dimensões[3].

2. Divisão do espaço nativo sem sobreposição:

   Nessa abordagem, o espaço é dividido em sub-regiões disjuntas, e a abstração de cada objeto é representada em todas as sub-regiões que ela intersecta, o que pode ser feito de duas formas:

   - Duplicação do objeto: o identificador do objeto é duplicado e armazenado nas páginas correspondentes a todos os sub-espaços que ele intersecta.
   - Divisão do objeto *(object clipping)*: O objeto é decomposto em vários objetos menores e disjuntos, de forma que cada sub-objeto esteja completamente contido em um sub-espaço.

   A maior vantagem da divisão do espaço sem sobreposição é facilidade com que os métodos de acesso a pontos podem ser estendidos. No entanto, ela tem como ponto desfavorável o aumento do número de entradas correspondentes a cada objeto, o que

[2] As *space-filling curves* serão melhor discutidas na seção 2.4.1.

[3] O DOT e a 2dMAP21 serão discutidos nas seções 2.4.2 e 2.4.3 respectivamente.

requer maior espaço de armazenamento e um tempo de execução maior das rotinas de inclusão e exclusão. Outro problema é que a densidade[4] máxima no espaço indexado deve ser menor que a capacidade de armazenamento da página. Considere o espaço $S$ da figura 2.9a, indexado por um método de acesso cujas páginas de disco têm capacidade de armazenar quatro retângulos. Com a inserção do retângulo $r$ (figura 2.9b), não há como fazer uma partição de forma que cada sub-espaço contenha no máximo quatro retângulos, ainda que alguns deles sejam divididos. Isto se deve ao espaço em destaque, onde os cinco retângulos se intersectam.

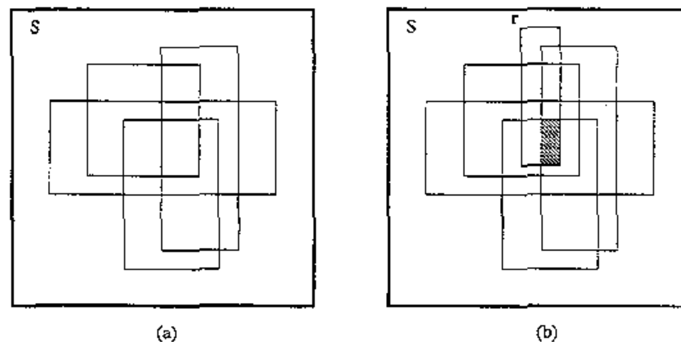



Figura 2.9: Aumento da densidade máxima com a inclusão de um retângulo.

3. Divisão do espaço nativo com sobreposição: Esta técnica, como a anterior, divide o espaço em sub-regiões, com a diferença de permitir que elas se sobreponham, de forma que cada objeto esteja totalmente contido em apenas uma delas. Essas sub-regiões são, então, organizadas em um índice hierárquico.

   A vantagem dessa abordagem sobre a última é que cada objeto é representado uma única vez. Sua desvantagem é que uma consulta sobre um objeto localizado em uma área de intersecção entre dois ou mais sub-espaços levará a uma busca em todos os ramos correspondentes no índice hierárquico, embora a representação do objeto esteja armazenada em apenas um deles.

Segundo Ooi, Sacks-Davis e Han [OSH93], alguns índices usam mais de uma técnica para estender os métodos de acesso a pontos para métodos de acesso a objetos de maior dimensão, a fim de que uma técnica compense os pontos fracos da outra. No entanto, essa manobra pode ter o efeito contrário, de modo que o método de acesso espacial pode herdar os pontos negativos de cada técnica empregada.

Em [SLS95], Stefanakis, Lee e Sellis apresentam uma nova técnica de extensão de métodos de acesso a pontos — a *abstração* — e por ser uma proposta relativamente

[4] Número de objetos que se intersectam em um determinado ponto.

recente será descrita com maior detalhe. Sua idéia básica é a representação dos MBRs dos objetos espaciais através de pontos no mesmo espaço onde eles estão dispostos, de forma que qualquer método espacial de acesso a pontos possa ser utilizado na indexação. Os autores sugerem dois esquemas de representação: *por vértices*, onde o MBR é substituído pelas coordenadas de seus quatro cantos, e *central*, onde é utilizado o ponto central do MBR (veja a figura 2.10).

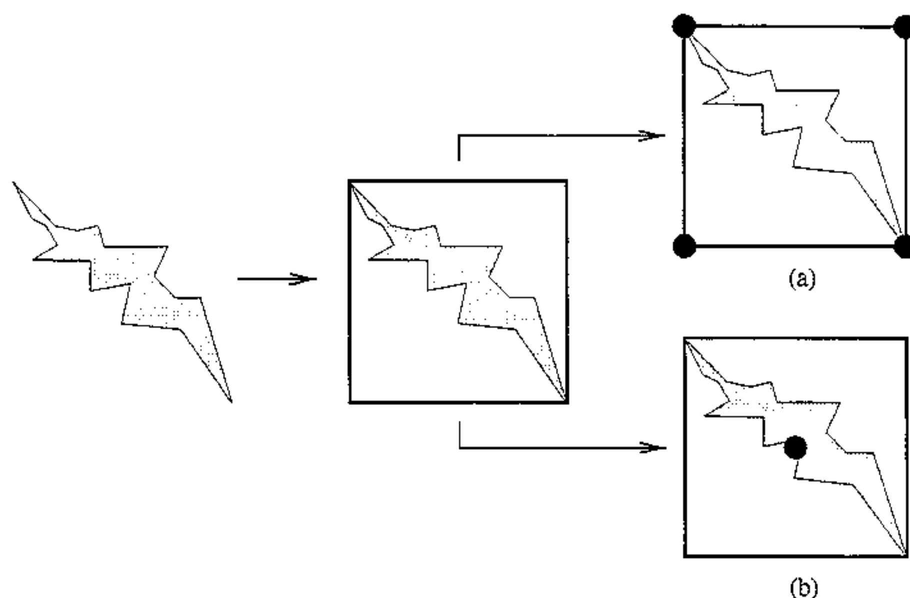



Figura 2.10: Esquemas de representação de um objeto espacial na técnica de abstração: (a) representação por vértices e (b) representação central.

Apesar dessa técnica preservar a localização de um objeto, ela não preserva sua extensão, o que pode causar erros na fase de filtragem. A figura 2.11 mostra duas consultas, uma sobre uma representação por vértices e outra central, onde dois objetos que intersectam as janelas de consultas são descartados.

A fim de resolver esse problema, os autores utilizam o conceito de *extensão da janela de consulta*. A idéia é aumentar o tamanho da janela, se necessário, de forma que todos os objetos que intersectem a janela original sejam recuperados na filtragem. O cálculo do tamanho da janela estendida se baseia nos seguintes parâmetros:

- $dX_{max}$ — o comprimento do maior lado na dimensão $X$ entre os MBRs armazenados;
- $dY_{max}$ — o comprimento do maior lado na dimensão $Y$ entre os MBRs armazenados;
- $dX_{jc}$ — o comprimento do lado da janela de consulta na dimensão $X$;
- $dY_{jc}$ — o comprimento do lado da janela de consulta na dimensão $Y$.

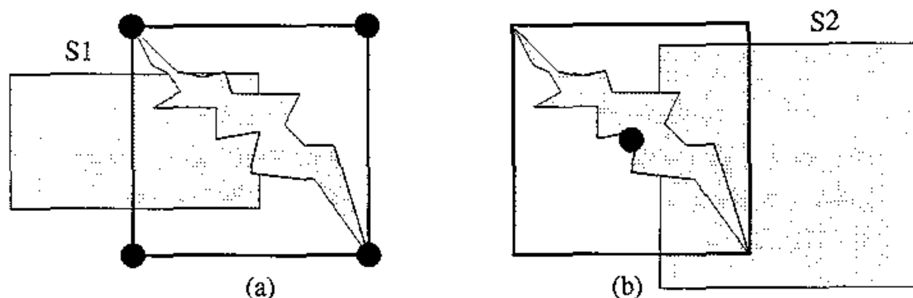



Figura 2.11: Duas consultas onde objetos que intersectam as janelas de consulta (S1 e S2) são descartados na fase de filtragem devido à representação por vértices (a) e central (b).

Para garantir a correção das consultas na representação por vértices, basta que $dX_{jc} \geq dX_{max}$ e $dY_{jc} \geq dY_{max}$. Assim, o aumento só será necessário caso um dos lados da janela de consulta seja menor que o maior lado de MBR na extensão correspondente. Na representação central é sempre obrigatório o aumento da janela de consulta, e seus lados nas dimensões $X$ e $Y$ devem ser, respectivamente, $dX_{jc} + dX_{max}$ e $dY_{jc} + dY_{max}$. Além disso, o ponto central da janela estendida deve coincidir com o da janela de consulta original.

A extensão da janela de consulta traz consigo um problema: se os tamanhos dos MBRs indexados apresentarem grande variação o desempenho será degradado pelo elevado número de objetos recuperados na filtragem, especialmente se houver uma grande quantidade de MBRs pequenos em relação ao número de MBRs maiores. Para tentar diminuir esse inconveniente, os autores discutem três formas de diminuir os *false hits*. A primeira fixa o tamanho da janela de consulta mínima e representa cada objeto por um número de pontos proporcional ao seu tamanho em relação a essa janela. Um objeto cujo MBR fosse menor ou igual a janela de consulta mínima em todas as dimensões seria representado através dos quatro vértices ou do ponto central desse MBR, de acordo com o esquema de representação adotado. Um objeto maior teria seu MBR sobreposto por uma grade cujas células teriam tamanho igual à janela de consulta mínima, e seria representado pelos vértices ou pelo ponto central de todas as células que intersectassem seu MBR. Alternativamente, poderiam ser consideradas apenas as células que intersectassem a representação completa do objeto. Veja o exemplo da figura 2.12.

A segunda estratégia seria subdividir o objeto original em objetos menores que coubessem em uma janela de consulta mínima predefinida, e representá-los através de seus vértices ou centros. A diferença desse procedimento para o anterior é que a *representação completa* do objeto seria dividida em novos sub-objetos. Uma vantagem é que a avaliação dos predicados na fase de refinamento poderia ser feita apenas sobre a representação completa de alguns fragmentos do objeto, e não de todo o objeto. A figura 2.13 ilustra o

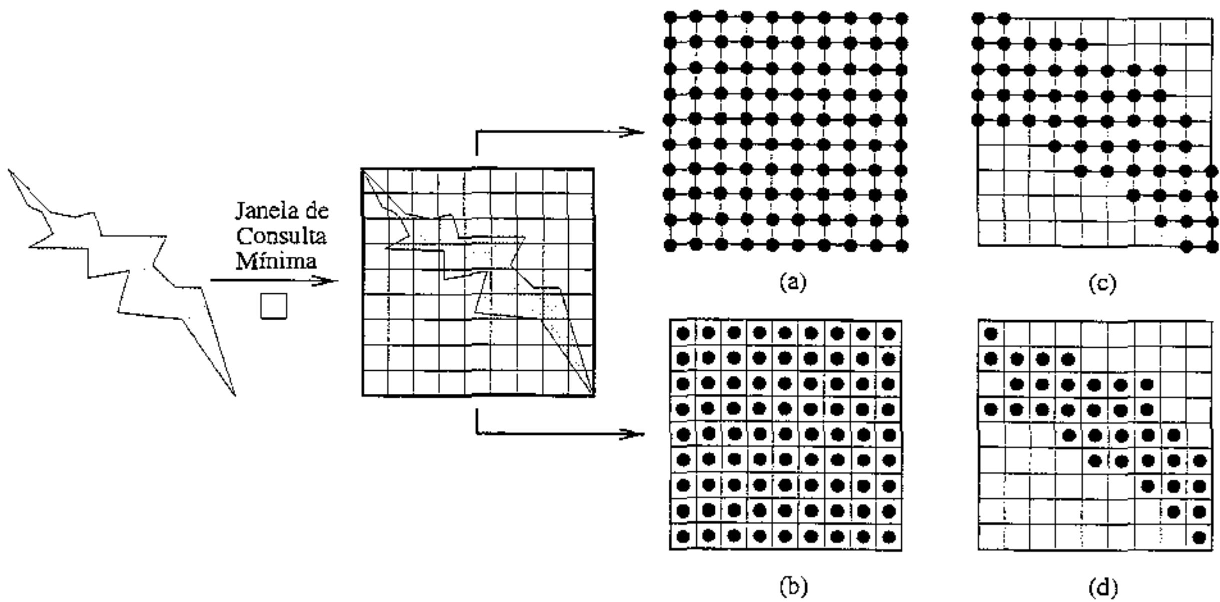



Figura 2.12: Representação de um objeto espacial por um número maior de pontos: (a) representação por vértices e (b) representação central. (c) e (d) mostram o resultado se somente as células que intersectam o objeto forem consideradas.

processo.

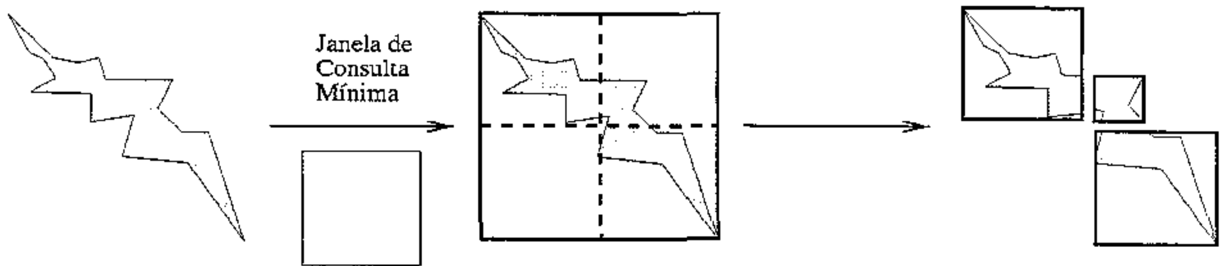



Figura 2.13: Divisão de um objeto espacial baseada no tamanho da janela de consulta mínima.

A terceira alternativa agrupa os MBRs de extensões semelhantes e cria um índice separado para cada grupo, com tamanhos mínimos de janelas de consulta apropriados, como mostra a figura 2.14. Cada consulta deve, então, ser realizada sobre todos os índices criados, o que pode ser compensado pelo fato de cada índice conter a representação de um número menor de objetos. A vantagem desta estratégia sobre as duas anteriores é que cada objeto é representado por um número mínimo de pontos e, portanto, o espaço de armazenamento necessário é menor (assim como o número de níveis do índice, caso uma estrutura hierárquica seja usada).

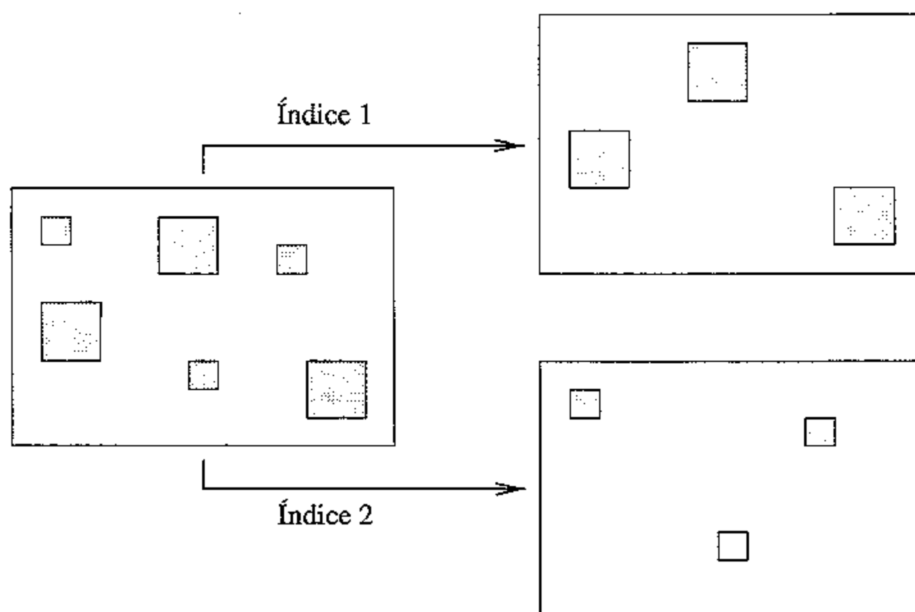



Figura 2.14: Separação de MBRs em índices diferentes de acordo com seus tamanhos.

A figura 2.15 resume a classificação dos métodos de acesso a objetos de dimensão não zero baseada nas técnicas utilizadas para estender os métodos de acesso a pontos.

### 2.3.2 Classificação baseada nos métodos de acesso convencionais de origem

Essa classificação foi proposta em [Cox91], mas aparece também em [OSH93]. Segundo Cox, as estruturas de dados utilizadas tanto nos métodos de acesso a pontos como nos métodos de acesso a objetos de dimensão não zero são extensões das estruturas usadas no gerenciamento de dados escalares (unidimensionais). Ele aponta dois motivos para se adotar essa classificação em detrimento da que se baseia na dimensão dos objetos suportados:

- Para Cox, a classificação baseada na dimensão dos objetos, rigorosamente falando, não existe, pois é possível se indexar objetos de dimensão não zero com métodos de acesso a pontos, utilizando a técnica da transformação. Da mesma forma, os métodos de acesso a objetos de dimensão não zero suportam a indexação de pontos, já que estes podem ser vistos como MBRs de extensão nula em todas as dimensões.
- Muitas das características das estruturas de dados unidimensionais, além de conhecidas, são preservadas nas estruturas estendidas. Isso facilita a determinação dos

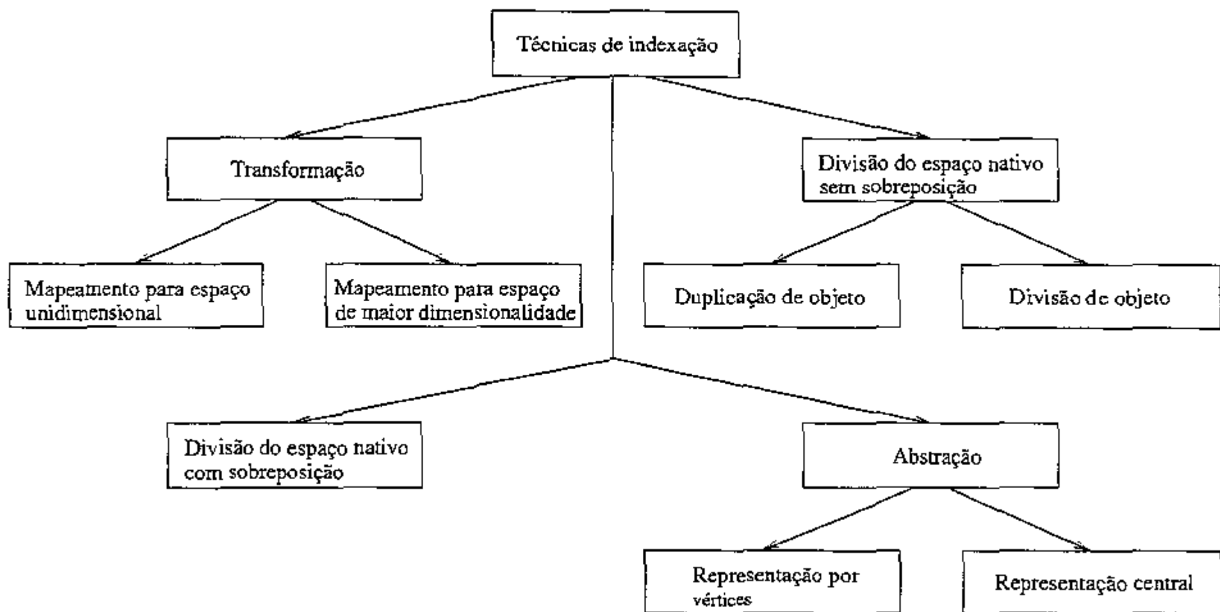



Figura 2.15: Técnicas utilizadas para derivar métodos de acesso a objetos de dimensão não zero a partir de métodos de acesso a pontos.

métodos de acesso mais apropriados a serem utilizados nos SGBDs, através das relações de desempenho, flexibilidade e simplicidade conhecidas dos métodos base.

Assim, Cox classifica os métodos de acesso espaciais como derivados de três grupos de estruturas de dados: árvores binárias, estruturas *hash* e árvores multiárias. Ooi [OSH93] chama os métodos desse último grupo de "derivados de $B^+$-trees", e acrescenta um quarto, o dos métodos que utilizam *space-filling curves*. No entanto, acreditamos que esta não seja uma boa denominação, já que existem métodos derivados de árvores multiárias que utilizam *space-filling curves* apenas para ordenar os MBRs, e não para transformá-los em pontos [KF93, KF94]. Um nome melhor seria *métodos que utilizam índices unidimensionais* (na maioria dos casos, o índice escolhido é a $B^+$-tree). Dessa forma, o grupo englobaria todos os métodos que utilizam a técnica da transformação para o espaço unidimensional, não importando se são usadas *space-filling curves* ou não.

Durante o restante desta revisão utilizaremos a classificação proposta por Cox acrescida deste último grupo, pois cremos que ela facilita a compreensão do funcionamento dos métodos de acesso.

Uma certa ênfase será dada aos métodos que utilizam índices convencionais e aos derivados de árvores multiárias, pois são desses dois grupos os métodos de acesso atualmente utilizados para o suporte a dados espaciais em SGBDs comerciais.

## 2.4 Métodos que utilizam índices convencionais

Objetos espaciais podem ser indexados através de métodos de acesso convencionais, desde que sofram uma transformação para o espaço unidimensional ou algum tipo de ordenação linear. A vantagem do uso de métodos convencionais reside no fato de que todos os SGBDs comerciais suportam pelo menos um desses índices, o que não ocorre com os métodos de acesso espaciais[5].

Os próximos tópicos descrevem alguns métodos de acesso espaciais que utilizam a estrutura de métodos de acesso a dados convencionais. Dentre estes, apenas a técnica da transformação para o espaço unidimensional usando *space filling curves* é hoje utilizada em SGBDs e SIGs comerciais. Três métodos, a 2dMAP21, a G-tree e a Filter Tree, são propostas bastante recentes.

### 2.4.1 Transformação através de *space-filling curves*

Uma *space-filling curve* é uma linha contínua que visita cada célula de uma grade $k$-dimensional exatamente uma vez, sem nunca cruzar a si mesma (nesse caso, visitar uma célula significa passar por um ponto predeterminado dessa célula, digamos, seu ponto central ou sua origem). A ordem em que as células são visitadas cria entre elas uma ordenação linear. Se fizermos o comprimento do lado das células tender a zero, cada célula tenderá a um ponto. Logo, essa ordenação pode ser realizada sobre qualquer conjunto de pontos que se queira indexar no espaço $k$-dimensional, bastando para isso que se utilize células cujo lado seja suficientemente pequeno para que cada uma contenha somente um ponto.

A ordenação gerada por uma *space-filling curve* será tanto melhor quanto mais eficaz ela for na preservação das distâncias, ou seja, pontos que são próximos no espaço $k$-dimensional devem estar próximos também no espaço unidimensional. O motivo é que objetos espacialmente próximos têm maior probabilidade de serem recuperados juntos numa consulta do que aqueles que estão distantes entre si. Por conseguinte, deseja-se que objetos próximos sejam representados por pontos tão próximos quanto possível uns dos outros no espaço unidimensional, de forma a serem armazenados, se não na mesma página, pelo menos em páginas contíguas ou não muito distanciadas, diminuindo o tempo de acesso a disco.

A seguir são apresentadas algumas dessas curvas. Embora elas possam ser usadas em espaços de qualquer dimensão, por simplicidade somente espaços bidimensionais serão considerados.

[5]Os únicos SGBDs comerciais de que temos informação a fazerem uso de um método de acesso espacial que não utiliza índices convencionais são o Illustra, da Informix, e o OpenIngres, da Computer Associates, que suportam a R-tree (descrita na seção 2.7.1).

### Percurso por colunas

A maneira mais simples de se ordenar as células de uma grade é através de um percurso por colunas (ou por linhas), como mostra a figura 2.16a. A coordenada linear de uma célula é definida por sua posição seqüencial na linha de percurso. Considere uma célula bidimensional $c$ de coordenadas $(x, y)$ dadas, respectivamente, pela coluna e pela linha que ela ocupa na grade, e seja $nc_y$ o número de coordenadas diferentes na dimensão $y$, isto é, o número de linhas da grade. A coordenada linear de $c$ é dada por $x * nc_y + y$ (a primeira coordenada em cada dimensão é 0, assim como a primeira coordenada linear).

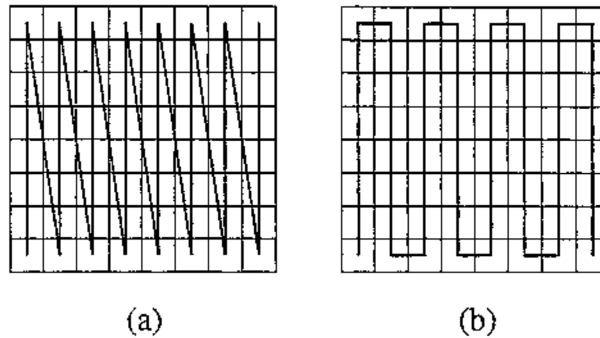



Figura 2.16: (a) Percurso por colunas e (b) percurso por colunas com sentidos alternados.

Outra forma de se percorrer as colunas de uma grade é mostrada na figura 2.16b, onde o sentido do percurso é invertido a cada mudança de coluna. Essa curva preserva melhor as distâncias do que a anterior, no sentido de que dois pontos consecutivos na ordenação linear são sempre consecutivos também no espaço bidimensional. A coordenada linear de uma célula bidimensional, nesse caso, é dada por $x * nc_y + y$ para valores pares de $x$ e $(x + 1) * nc_y - y - 1$ para valores ímpares de $x$.

A ordenação por colunas é ótima se a maioria das consultas envolverem áreas de formato vertical. No entanto, se as janelas de consulta tiverem sua maior dimensão no eixo horizontal, teremos um desempenho ruim, pois várias colunas serão acessadas apenas para se recuperar alguns dos elementos de cada. Obviamente as mesmas considerações podem ser feitas para uma ordenação por linhas.

### Curva de Peano ou Curva Z

O uso das curvas z no mapeamento de células multidimensionais para um espaço unidimensional é descrito em [OM84]. A coordenada unidimensional de uma célula é obtida pela intercalação dos bits das representações binárias de suas coordenadas em cada dimensão. A ordem em que os bits são intercalados, ou seja, de que coordenada é tomado cada bit da intercalação, depende de uma função denominada *função de embaralhamento*.

A única exigência a ser cumprida por essa função é a de manter a ordem entre os bits de cada coordenada. Como exemplo, tomemos duas cadeias com 2 bits: $X =' 00'$ e $Y =' 11'$. Algumas das possíveis funções de embaralhamento são:

- Tomar alternadamente 1 bit de cada cadeia, iniciando com a cadeia $X$. O resultado seria '0101'.
- Tomar alternadamente 1 bit de cada cadeia, iniciando com a cadeia $Y$. Resultado: '1010'.
- Concatenar os bits da cadeia $Y$ à direita dos bits da cadeia $X$: '0011'.
- Tomar 1 bit da cadeia $Y$, concatenar os bits da cadeia $X$, e, finalmente, o segundo bit da cadeia $Y$: '1001'.

Existem várias outras funções que poderiam ser usadas, mas a mais comum é mesmo a primeira, isto é, tomar um bit de cada dimensão alternadamente. Para uma grade bidimensional $4 \times 4$, o resultado desta função é mostrado na figura 2.17 (o primeiro bit é retirado da dimensão $X$).

| | | | | |
|---|---|---|---|---|
| 11 | 0110 | 0111 | 1101 | 1111 |
| 10 | 0100 | 0101 | 1100 | 1100 |
| 01 | 0001 | 0011 | 1001 | 1011 |
| 00 | 0000 | 0010 | 1000 | 1010 |
| | 00 | 01 | 10 | 11 |

Figura 2.17: Coordenadas lineares das células de uma grade $4 \times 4$ geradas pela intercalação dos bits das coordenadas em cada dimensão.

A curva que corresponde a uma grade $2 \times 2$ é chamada *curva de ordem 1*, e a partir dela se pode gerar recursivamente qualquer curva de ordem $i$, que percorre uma grade $2^i \times 2^i$. Para isto, basta que se substitua cada vértice da curva de ordem 1 pela curva de ordem $i - 1$. A figura 2.18 mostra as curvas z para grades $2 \times 2$, $4 \times 4$ e $8 \times 8$, que correspondem, respectivamente, às curvas de ordem 1, 2 e 3.

### Curva de Código Binário Gray Refletido

Em um código binário Gray *(Gray code)*, as cadeias de bits são seqüenciadas de tal forma que duas consecutivas tenham exatamente um bit divergente. Uma das várias maneiras de

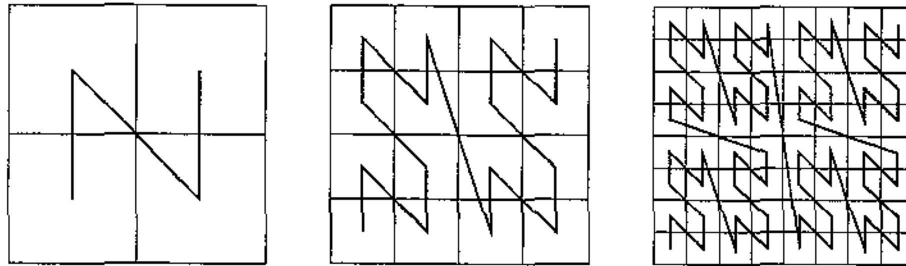


Figura 2.18: Curvas z de ordem 1, 2 e 3.

se organizar as cadeias atendendo a este requisito é dada pelos chamados *códigos binários Gray refletidos.* Um código binário Gray refletido com cadeias de $n$ bits, denotado por $G(n)$, pode ser representado como uma matriz binária $2^n \times n$, onde cada linha $G_{n,i}$ é uma cadeia de bits do código.

A matriz que define o código para cadeias de 1 bit é:

$$G(1) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Seja a matriz que representa um código Gray com cadeias de $n$ bits

$$G(n) = \begin{bmatrix} G_{n,0} \\ G_{n,1} \\ \dots \\ G_{n,2^n-1} \end{bmatrix}.$$

Então o código binário Gray refletido para cadeias com $n+1$ bits é obtido recursivamente, fazendo

$$G(n+1) = \begin{bmatrix} 0G_{n,0} \\ 0G_{n,1} \\ \dots \\ 0G_{n,2^n-1} \\ 1G_{n,2^n-1} \\ \dots \\ 1G_{n,1} \\ 1G_{n,0} \end{bmatrix}.$$

Em [Fal86], Faloutsos propõe o uso dos códigos binários Gray refletidos em funções de *hashing* e na ordenação linear de células multidimensionais, em substituição à curva z. A nova curva é obtida da seguinte forma:

- As linhas e colunas da grade são numeradas usando o código binário Gray refletido.

| | | | | |
|---|---|---|---|---|
| 10 | 0100 | 0110 | 1110 | 1100 |
| 11 | 0101 | 0111 | 1111 | 1101 |
| 01 | 0001 | 0011 | 1011 | 1001 |
| 00 | 0000 | 0010 | 1010 | 1000 |
| | 00 | 01 | 11 | 10 |

(a)

| | | | |
|---|---|---|---|
| 0111 | 0100 | 1011 | 1000 |
| 0110 | 0101 | 1010 | 1001 |
| 0001 | 0010 | 1101 | 1110 |
| 0000 | 0011 | 1100 | 1111 |

(b)

Figura 2.19: Códigos Gray das células de uma grade $4 \times 4$, gerados pela intercalação dos bits dos códigos Gray das coordenadas de cada dimensão (a), e seus equivalentes binários (b).

- O código de cada célula é derivado pela intercalação dos bits de cada coordenada (veja a figura 2.19).
- Para calcular a posição ocupada por uma célula na curva é feita a transformação de seu código Gray para seu equivalente binário (ou decimal). Este equivalente é o que realmente é usado nas entradas do método de acesso unidimensional, e não o código Gray. As fórmulas de conversão de um código binário Gray refletido de $n$ bits $(g_n g_{n-1} \ldots g_1)$ para seu correspondente binário $(b_n b_{n-1} \ldots b_1 b_0)$ são [Fal86]:

$$b_n = 0 \quad (2.1)$$

$$b_j = \sum_{m=j+1}^{n} g_m \ (mod 2), \ 0 \leq j < n. \quad (2.2)$$

A tabela 2.2 mostra o código binário Gray refletido para cadeias com 4 bits e seus equivalentes binários e decimais.

A figura 2.20 apresenta as curvas de código Gray para grades $2 \times 2$, $4 \times 4$ e $8 \times 8$, correspondentes às curvas de ordem 1, 2 e 3, respectivamente. Uma curva de ordem $i$ pode ser gerada pela substituição de cada vértice da curva básica pela curva de ordem $i - 1$, rotacionando $180^o$ as curvas que substituem os vértices superiores.

### Curva de Hilbert

Uma desvantagem da maioria das curvas anteriores é que nem sempre duas células consecutivas na ordenação linear estão próximas no espaço multidimensional (veja os "saltos" entre colunas no percurso por colunas e entre quadrantes nas curvas z e de código Gray).

| Gray | Binário | Decimal |
|---|---|---|
| 0000 | 0000 | 0 |
| 0001 | 0001 | 1 |
| 0011 | 0010 | 2 |
| 0010 | 0011 | 3 |
| 0110 | 0100 | 4 |
| 0111 | 0101 | 5 |
| 0101 | 0110 | 6 |
| 0100 | 0111 | 7 |
| 1100 | 1000 | 8 |
| 1101 | 1001 | 9 |
| 1111 | 1010 | 10 |
| 1110 | 1011 | 11 |
| 1010 | 1100 | 12 |
| 1011 | 1101 | 13 |
| 1001 | 1110 | 14 |
| 1000 | 1111 | 15 |

Tabela 2.2: Código binário Gray refletido de 4 bits.

Apesar do percurso por colunas com sentidos alternados não apresentar este ponto desfavorável, outro deve ser levado em consideração: células próximas no espaço multidimensional podem ficar bastante distanciadas no unidimensional. Veja, por exemplo, as células de menor coordenada $y$ em duas colunas consecutivas.

Na tentativa de minimizar estes problemas, Faloutsos e Roseman [FR89] e Jagadish [Jag90] propõem o uso da curva de Hilbert para a ordenação de células multidimensionais. A curva de Hilbert de ordem 1 tem a mesma seqüência de visitação das células que a curva de código Gray . A curva de ordem $i$ é obtida pela substituição de cada vértice da curva de ordem 1 pela curva de ordem $i-1$, com os seguintes cuidados:

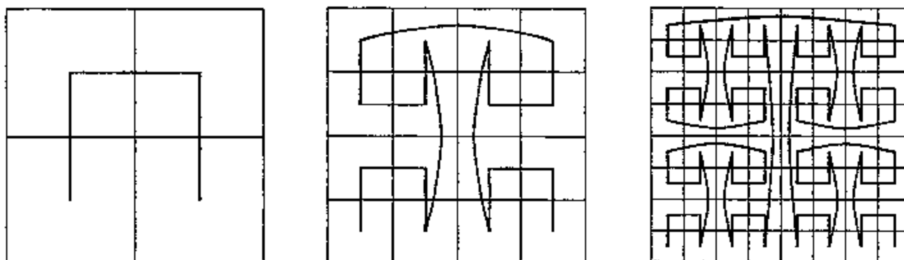


Figura 2.20: Curvas de código Gray de ordem 1, 2 e 3.

- a curva que substitui o vértice inferior esquerdo deve ser rotacionada 90° no sentido horário, e o sentido de percurso é invertido (o antigo ponto inicial passa a ser o final e vice-versa);
- a curva que substitui o vértice inferior direito deve ser rotacionada 90° no sentido anti-horário, e a mesma inversão de sentido do caso anterior deve ser realizada.

A figura 2.21 mostra as curvas de Hilbert de ordem 1, 2 e 3.

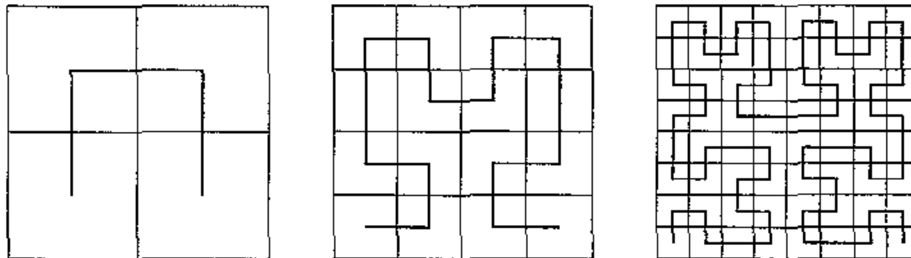


Figura 2.21: Curvas de Hilbert de ordem 1, 2 e 3.

A determinação da coordenada linear de uma célula é realizada pela intercalação dos bits das coordenadas em cada dimensão, com posterior decodificação da cadeia obtida, de acordo com as rotações necessárias. Tanto [FR89] como [Jag90] apresentam algoritmos de cálculo da coordenada linear numa curva de Hilbert para pontos representados num espaço bidimensional. O primeiro artigo traz também algoritmos para espaços tridimensionais e quadridimensionais.

### Uso de *space-filling curves* para indexação de objetos de dimensão não zero

Após a transformação para o espaço unidimensional, a indexação de pontos representados em um espaço multidimensional por um método de acesso convencional, como a $B^+$tree, é feita diretamente. No entanto, indexação de objetos de dimensão não zero requer outros procedimentos, já que cada um pode se sobrepor a mais de uma célula da grade geradora da *space-filling curve*.

Para se representar um polígono (ou linha poligonal) é usada a técnica da decomposição regular (veja a seção 2.2), onde o tamanho das células da grade é definido de acordo com a precisão desejada para a aproximação do objeto. O polígono é representado, então, pelas coordenadas lineares de cada célula que ele intersecta.

Como já foi dito, uma desvantagem dessa técnica é que a cada objeto podem corresponder, no índice, um grande número de entradas. Para diminuir esse problema, as células são, na medida do possível, agrupadas em "supercélulas".

Consideremos uma grade formada pela partição regular e recursiva do espaço em quatro quadrantes, contendo $2^n \times 2^n$ células, onde a coordenada linear de cada célula é representada por $2n$ bits. Cada supercélula é um subconjunto maximal $S_i$ do conjunto de células $P$ que representa um objeto, tal que:

- $|S_i| = 2^{k_i}$, para $1 \leq k_i < 2n$;
- Os $2n - k_i$ bits mais significativos são comuns às coordenadas lineares de todos os elementos de $S_i$.

Assim, cada polígono passa a ser representado pela coordenada linear de cada célula de $P$ que não pertença a nenhuma supercélula, mais a coordenada linear de cada supercélula $S_i$, que é dada pelos $2n - k$ bits mais significativos comuns a todas as suas células. É importante notar que aqui as coordenadas lineares são representadas como *cadeias* de bits, e que somente os comprimentos de duas cadeias não são suficientes para ordená-las. Por exemplo, '110' > '101011', pois a comparação é feita a partir dos bits mais significativos de cada cadeia.

A figura 2.22a mostra alguns agrupamentos possíveis e suas coordenadas lineares numa grade 4×4 com células ordenadas por uma curva z, e a figura 2.22b, agrupamentos não permitidos.

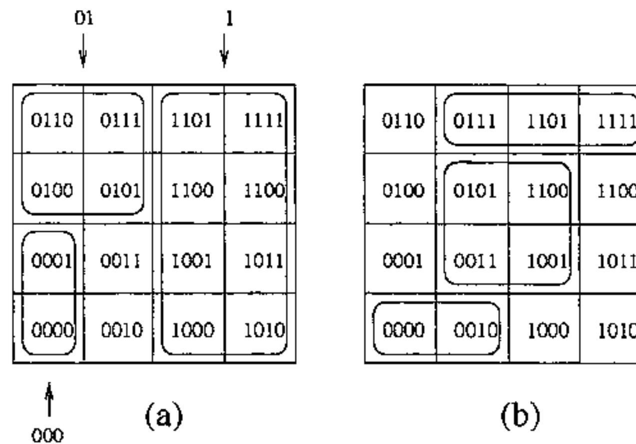



Figura 2.22: Alguns agrupamentos permitidos (a) e não permitidos (b) numa grade com células ordenadas por uma curva z.

A figura 2.23 ilustra o processo de transformação de um polígono (a) utilizando uma curva z. Em (b) o espaço é particionado por uma grade com 8 × 8 células, e aquelas que intersectam o polígono são identificadas. Como resultado dessa operação, o polígono é representado pelas coordenadas lineares de 27 células (c). Após o agrupamento (d), apenas 5 coordenadas são necessárias:

- '000101'
- '0011', resultante do agrupamento das células '001100' a '001111';
- '01101', resultante do agrupamento das células '011010' e '011011';
- '10', resultante do agrupamento das células '100000'a '101111';
- '1100', resultante do agrupamento das células '110000' a '110011'.

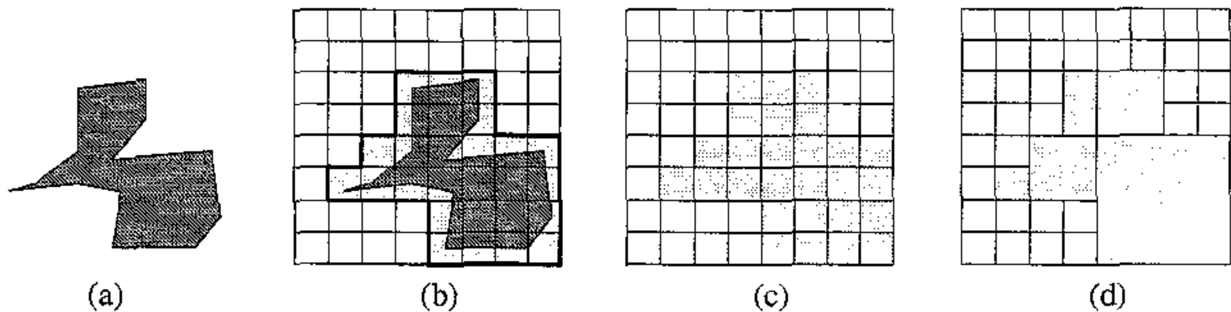



Figura 2.23: Processo de representação de um polígono através de coordenadas lineares em uma curva z.

### Operações

A inclusão ou exclusão de um objeto é feita através do cálculo das coordenadas dos pontos correspondentes no espaço unidimensional, seguida da inclusão ou exclusão das entradas no método de acesso convencional escolhido.

A consulta para verificação da existência de um objeto no índice é mais simples, pois no caso de um polígono ou linha poligonal não é necessária a busca de todas as coordenadas lineares que o representam. Pode ser usada a coordenada linear de qualquer ponto contido em uma das células que representam o objeto, como o seu centróide, por exemplo. A coordenada linear desse ponto será uma cadeia de bits que forçosamente estará "contida" na coordenada de alguma célula ou supercélula que representa o objeto (isto é, os $m$ bits que compõem a coordenada da célula ou supercélula que contêm o ponto são também os $m$ bits mais significativos de sua coordenada linear).

Uma vez escolhido o ponto que representará o objeto, a consulta é traduzida na procura, no índice convencional, de uma cadeia de bits que seja igual a qualquer número de bits mais significativos da coordenada do ponto. Como dois ou mais objetos podem conter o mesmo ponto, é necessária uma posterior verificação dos identificadores dos objetos recuperados para se chegar à resposta correta da consulta.

A determinação dos objetos que intersectam (ou incluem, ou são incluídos em) uma janela de consulta é feita pela decomposição da janela nos mesmos moldes usados para polígonos. Depois desse passo, são recuperados do índice todas as entradas que tenham um número qualquer de bits mais significativos iguais ao mesmo número de bits mais significativos da coordenada de qualquer uma das células ou supercélulas que representam a janela de consulta. Após a eliminação de identificadores duplicados, os objetos recuperados passam por uma etapa de refinamento para se chegar ao resultado final.

### 2.4.2 DOT

O DOT (de DOuble Transformation) é um método de acesso proposto por Faloutsos e Rong em [FR91] com o intuito de permitir a indexação de objetos espaciais utilizando índices convencionais sem o inconveniente da possível geração de várias entradas para um mesmo objeto, como ocorre com a transformação através de *space-filling curves*.

Cada objeto a ser inserido passa por duas transformações. Primeiro, seu MBR no espaço $k$-dimensional, representado em cada dimensão $d$ por $ri_d$ e $rf_d$, respectivamente os pontos de início e fim de sua extensão, é mapeado para um ponto num espaço $2k$-dimensional, de coordenadas $(ri_1, rf_1, \ldots, ri_k, rf_k)$. Depois esse ponto é transformado em outro, agora num espaço unidimensional, pelo uso de uma *space-filling curve*. A coordenada linear gerada pode, então, ser diretamente indexada por um método de acesso convencional.

O procedimento de exclusão segue o mesmo raciocínio do algoritmo de inclusão. As coordenadas do MBR do objeto que se deseja excluir passam pelas duas transformações descritas e, depois, a coordenada linear obtida é utilizada para se encontrar, no índice convencional, a entrada correspondente. O objeto e sua entrada são, então, removidos.

A execução de *range queries* exige que as janelas de consulta sejam transformadas para o espaço $2k$-dimensional, para depois se tornarem um conjunto de intervalos no espaço unidimensional. A figura 2.24 mostra um exemplo da primeira transformação usando segmentos de reta dispostos em um espaço unidimensional, os quais são mapeados para pontos no espaço bidimensional. A figura também apresenta a transformação de uma janela de consulta $(j_i, j_f)$. As coordenadas mínima e máxima do espaço unidimensional considerado são $c_{min}$ e $c_{max}$, respectivamente. No espaço bidimensional, o eixo horizontal $(x_i)$ representa as coordenadas mínimas dos segmentos, e o vertical $(x_f)$ as máximas. A área em destaque é o espaço onde os pontos podem representar retângulos que intersectam a janela de consulta.

A janela no espaço bidimensional é limitada pelas retas:

- $x_i = c_{min}$, pois não existe segmento que comece antes do início do espaço considerado;

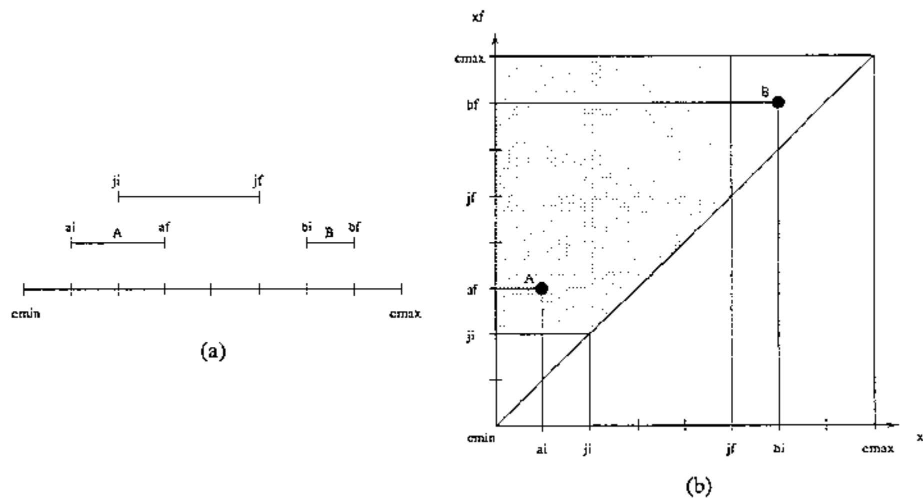



Figura 2.24: Transformação de segmentos de reta e de uma janela de consulta em um espaço unidimensional (a) para um espaço bidimensional (b).

- $x_i = j_f$, pois qualquer segmento deve começar no máximo em $j_f$ para intersectar a janela;
- $x_f = j_i$, pois qualquer segmento deve terminar no mínimo em $j_i$ para que intersecte a janela;
- $x_f = c_{max}$, pois não existe segmento que termine depois do final do espaço considerado;
- $x_i = x_f$, pois em qualquer segmento $x_i \leq x_f$.

### Comentários

Em [FR91] todos os exemplos se basearam na indexação de segmentos de reta num espaço unidimensional, onde eles podem ser considerados retângulos unidimensionais. Na primeira transformação, eles foram mapeados para pontos num espaço bidimensional, para depois serem ordenados por uma curva de Hilbert. Os segmentos de reta foram utilizados para facilitar a exposição das idéias e permitir a representação gráfica das transformações, o que não seria possível se os objetos originais estivessem em um espaço bidimensional.

O mesmo procedimento foi adotado nos testes para determinação de desempenho descritos no artigo, onde o DOT foi comparado com a R-tree e saiu vencedor. O problema é que não se pode afirmar que o DOT é também melhor que a R-tree na indexação de retângulos bidimensionais com base apenas nesses experimentos. Testes adicionais seriam necessários.

### 2.4.3 2dMAP21

A 2dMAP21, proposta por Nascimento e Dunham em [ND96, ND97], se baseia na MAP21 [NDK96], uma estrutura concebida originalmente para a indexação de intervalos de tempo representados como segmentos de reta. Assim, faz-se necessária uma breve descrição do funcionamento desta.

#### MAP21

Na MAP21, uma função $F(.)$ é usada para transformar os pontos inicial e final, $I_i^k$ e $I_f^k$, de cada intervalo indexado $I^k$ em um único ponto, que é armazenado em uma B$^+$-tree. O número máximo de dígitos necessários para representar a coordenada final de qualquer intervalo, denotado por $\alpha$, deve ser conhecido, e os valores dos pontos que delimitam cada intervalo devem ser não negativos. Assim, para qualquer intervalo $I^k = [I_i^k, I_f^k]$, temos $0 \leq I_i^k \leq I_f^k \leq 10^\alpha - 1$. A função $F(I^k)$ que mapeia o intervalo $I^k$ para um único valor é definida como $F(I^k) = F(I_i^k, I_f^k) = I_i^k \times 10^\alpha + I_f^k$.

A função $F(.)$ tem as seguintes propriedades:

- mapeia intervalos distintos em pontos distintos, isto é, dados dois intervalos $I^k = [I_i^k, I_f^k]$ e $I^l = [I_i^l, I_f^l]$, então $F(I^k) = F(I^l) \Leftrightarrow I_i^k = I_i^l$ e $I_f^k = I_f^l$;
- cria uma ordenação lexicográfica dos intervalos, ou seja, dados dois intervalos $I^k = [I_i^k, I_f^k]$ e $I^l = [I_i^l, I_f^l]$, se $I_i^k < I_i^l \Rightarrow F(I^k) < F(I^l)$; e se $I_i^k = I_i^l$ e $I_f^k < I_f^l \Rightarrow F(I^k) < F(I^l)$.

O intervalo original $I^k$ pode ser obtido a partir do ponto gerado por $F(I^k)$, fazendo-se $I_i^k = \frac{F(I^k) - [F(I^k) \ mod \ 10^\alpha]}{10^\alpha}$ e $I_f^k = F(I^k) \ mod \ 10^\alpha$.

Cada entrada de um nó folha da B$^+$-tree armazena a coordenada de um ponto e um ponteiro para uma lista ligada dos registros representados por ele (ou seja, intervalos que têm os mesmos extremos).

A rotina de exclusão usa a função $F(.)$ para mapear as coordenadas do intervalo a ser removido para o ponto que o representa. É feita, então, uma consulta à B$^+$-tree para se localizar a entrada que armazena a coordenada desse ponto, e o registro relacionado ao intervalo é procurado na lista correspondente e excluído. Se a lista se torna vazia, a entrada que a referencia na B$^+$-tree também é removida.

A execução de consultas sobre um conjunto de intervalos para determinar os que intersectam uma janela exige o conhecimento prévio do comprimento do maior intervalo indexado ou que se deseja indexar, denotado como $\Delta$. Assim, $\Delta \geq max_k\{I_i^k - I_f^k\}$.

Dada uma janela de consulta $Q = [Q_i, Q_f]$, os prováveis intervalos que a intersectam são os que têm seu ponto inicial no intervalo $[Q_i - \Delta, Q_f]$. Então os pontos candidatos

a satisfazer a consulta têm valores entre $(Q_i - \Delta) \times 10^\alpha + Q_i$ e $Q_f \times 10^\alpha + (Q_f + \Delta)$ inclusive. Esses pontos são recuperados através de uma pesquisa na $B^+$-tree, reconvertidos nos intervalos originais, e os ponteiros correspondentes àqueles que realmente fazem parte da resposta (intersectam a janela de consulta) são incluídos em um conjunto $P$, que é devolvido como resultado. Os demais são descartados.

As consultas para identificar os intervalos contidos em uma janela são mais simples, pois apenas aqueles representados por pontos com valores entre $Q_i \times 10^\alpha + Q_i$ e $Q_f \times 10^\alpha + Qf$ inclusive, precisam ser considerados. Os demais procedimentos são idênticos aos descritos na consulta para determinação de intersecção, com a diferença de que são descartados os ponteiros correspondentes aos intervalos não contidos na janela.

### Extensão da MAP21 para o espaço bidimensional

A 2dMAP21, estrutura criada para indexar retângulos, usa uma MAP21 para cada dimensão. Os retângulos são decompostos em suas projeções nos eixos $X$ e $Y$, como mostra a figura 2.25, e os intervalos obtidos em cada eixo são transformados em pontos e inseridos na $B^+$-tree correspondente. É bom notar que cada MAP21 tem seu próprio valor de $\Delta$, que está relacionado ao comprimento do maior intervalo que indexa. A figura 2.26 esquematiza as entradas correspondentes aos retângulos da figura 2.25 em cada MAP21.

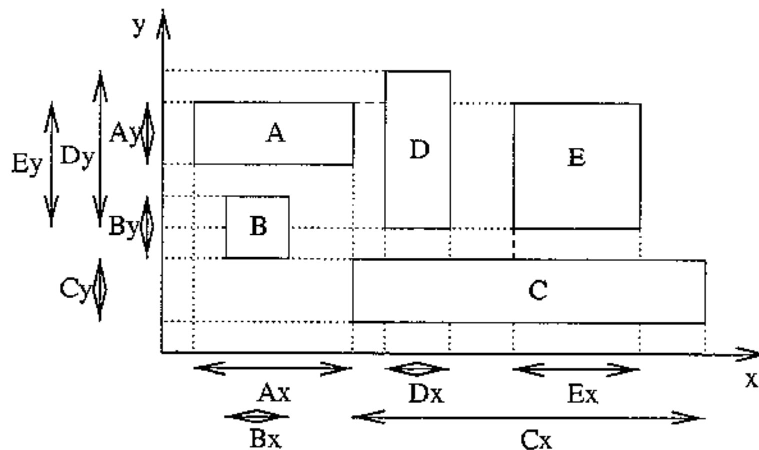



Figura 2.25: Um conjunto de MBRs e suas projeções em cada eixo.

A exclusão de um retângulo da 2dMAP21 consiste da remoção do registro que o representa em cada uma das MAP21.

As consultas para determinação de intersecção são executadas na 2dMAP21 da seguinte forma:

- a janela de consulta $Q$ é decomposta em suas projeções nos eixos $X$ e $Y$, $Q_x$ e $Q_y$;

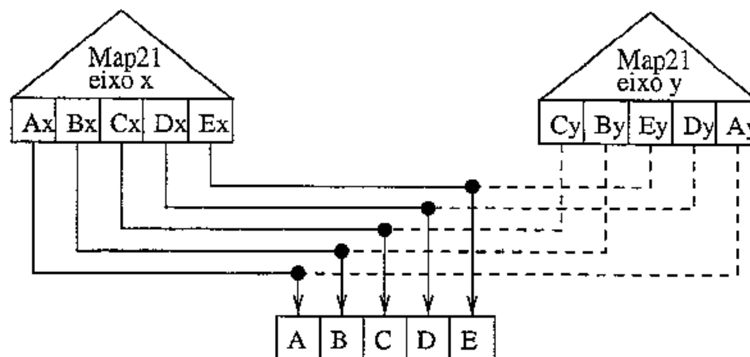



Figura 2.26: Indexação dos retângulos da figura anterior através de duas árvores MAP21.

- cada projeção é usada para realizar uma consulta para determinação de intersecção na MAP21 correspondente, gerando dois conjuntos de ponteiros para os registros de dados, $P_x$ e $P_y$, relativos aos pontos recuperados nos eixos $X$ e $Y$, respectivamente;
- O conjunto $P = P_x \cap P_y$ é devolvido como resultado (isto é, os ponteiros para os retângulos cujas projeções em ambos os eixos intersectam as projeções da janela de consulta).

As consultas para identificação dos retângulos contidos numa janela de consulta são feitas da mesma maneira, apenas substituindo-se, no segundo passo, as consultas para determinação de intersecção por consultas para identificação dos intervalos contidos nas projeções $Q_x$ e $Q_y$ da janela.

Pelo fato da MAP21 se basear no comprimento do maior intervalo indexado para determinar o tamanho da janela de consulta, um conjunto de dados composto por um grande número de retângulos "pequenos" e um pequeno número de retângulos "grandes" pode degradar o desempenho da 2dMAP21 nas consultas. O procedimento adotado para resolver esse problema é uma das soluções usadas também na técnica de extensão de métodos de acesso a pontos por abstração (seção 2.3.1): o agrupamento, em cada dimensão, de projeções de comprimentos semelhantes e a criação de uma MAP21 para cada grupo com seu próprio valor de $\Delta$. O aumento do número de MAP21 na composição da 2dMAP21 pode ser compensado pelo menor número de entradas contidas em cada uma.

### Paralelismo

Além de herdar todo o suporte existente para a B$^+$-tree, a 2dMAP21 tem, ainda, outra propriedade: é facilmente paralelizável, pois as MAP21 que indexam os intervalos de cada eixo podem ser armazenadas em discos diferentes, de forma que a consulta pode ser feita paralelamente em ambas. No caso do uso de mais de uma MAP21 por dimensão (para separar os intervalos de tamanhos diferentes), o grau de paralelismo pode ser ainda maior.

### 2.4.4 G-tree

A G-tree *(grid tree)* [Kum94] é uma estrutura criada para a indexação de pontos que se baseia na partição hierárquica do espaço em regiões disjuntas. Cada partição corresponde a uma página de disco ou *bucket*, e comporta um número máximo de entradas denotado pelo parâmetro $m$.

Se a inserção de um novo ponto deve ser feita em uma região com a capacidade máxima já atingida, ela é dividida em duas novas sub-regiões de mesmo tamanho. Essa divisão é feita a cada vez em apenas uma das dimensões do espaço $k$-dimensional, a qual é escolhida ciclicamente, de forma que se uma região foi gerada por uma divisão na dimensão $i$ ela será particionada na dimensão $i + 1$ (ou na dimensão 1, se $i = k$).

Inicialmente, o espaço é dividido em duas partições de mesmo tamanho, cujos identificadores são '0' e '1'. Cada vez que uma partição é subdividida, as duas novas regiões herdam os bits do identificador da partição original e acrescentam um '0' ou um '1' à sua direita, para formar os identificadores correspondentes à primeira e à segunda metade, respectivamente. A figura 2.27 ilustra o processo de particionamento para um espaço bi-dimensional. O valor assumido para $m$ é 2. Note que os identificadores são strings de bits — os mesmos que seriam obtidos pela transformação das partições usando uma curva z. Portanto, as partições podem ser ordenadas linearmente com base nesses identificadores, o que permite que eles sejam utilizados como entradas em uma B$^+$-tree. Para economizar espaço, as partições sem pontos não são armazenadas. Cada entrada nas folhas da B$^+$-tree aponta para uma página de dados onde estão armazenados os pontos contidos na partição correspondente.

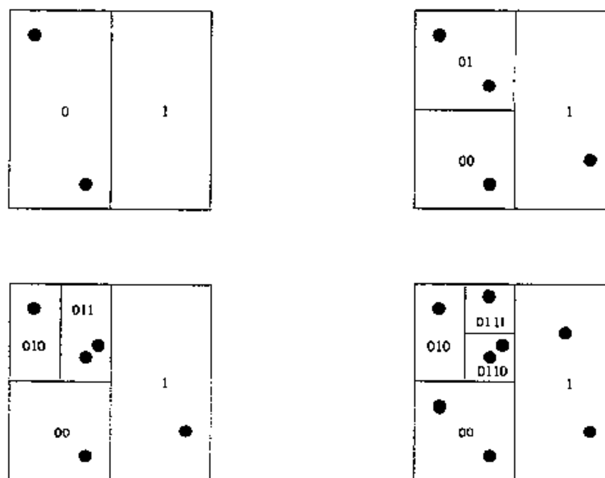



Figura 2.27: Partição do espaço na G-tree.

Algumas definições são úteis para a discussão das operações. Dadas duas partições $P_1$ e $P_2$, com $b_1$ e $b_2$ bits respectivamente, e $b = min(b_1, b_2)$, os $b$ bits mais significativos de

$P_1$ e $P_2$ são denotados, respectivamente, como $BMS(P_1, b)$ e $BMS(P_2, b)$. A partir daí, define-se:

- $P_1 > P_2$ se $BMS(P_1, b) > BMS(P_2, b)$.
- $P_1 < P_2$ se $BMS(P_1, b) < BMS(P_2, b)$.
- $P_1 \subset P_2$ se $BMS(P_1, b) = BMS(P_2, b)$ e $b_1 > b_2$. Esse é o caso em que $P_1$ é uma subpartição de $P_2$.
- $P_1 \subseteq P_2$ se $P_1 \subset P_2$ ou $P_1 = P_2$.
- $P_1 \supset P_2$ se $P_2 \subset P_1$.
- $P_1 \supseteq P_2$ se $P_1 \supset P_2$ ou $P_1 = P_2$.
- A partição pai de $P$, denotada como $pai(P)$, é determinada removendo-se o bit menos significativo de $P$. Por exemplo, $pai('010') = '01'$.
- O complemento da partição $P$, denotada como $compl(P)$, é determinado invertendo-se o bit menos significativo de $P$. Por exemplo, $compl('010') = '011'$. Note que $pai(P) = pai(compl(P))$.

**Inserção**

Para se inserir um ponto na estrutura é necessário descobrir se a partição a que ele pertence já existe na B$^+$-tree. Como não se sabe a priori que partição é essa, calcula-se uma partição inicial aproximada, denominada $P_{apr}$, assumindo que o ponto será inserido numa partição de tamanho igual à menor partição criada até o momento (e, portanto, com o mesmo número de bits no identificador).

O próximo passo é procurar, na B$^+$-tree, uma partição $P$ tal que $P_{apr} \subseteq P$. Se essa partição é encontrada e sua capacidade máxima ainda não foi atingida, o ponto é simplesmente inserido na página de dados correspondente. Se um *split* é necessário, a partição é dividida, excluída da B$^+$-tree, e os pontos que ela continha são redistribuídos (pode ser que todos os pontos sejam alocados a apenas uma das subpartições). Cada subpartição é, então inserida na $B^+$-tree, desde que não esteja vazia. A partir daí, um novo valor de $P_{apr}$ é associada ao ponto a ser inserido, desta vez com o mesmo número de bits das subpartições criadas, e todo o processo se repete até a inserção do ponto.

Caso a partição procurada não exista é necessário criá-la. Essa nova partição é a própria $P_{apr}$ ou sua ancestral de maior área que não intersecte partições já existentes na B$^+$-tree, e é identificada da seguinte forma: enquanto $P_{ant} < pai(P_{apr}) < P_{prox}$ retira-se um bit menos significativo de $P_{apr}$. $P_{ant}$ e $P_{prox}$ são, respectivamente, os identificadores

de partições *existentes na $B^+$-tree* imediatamente menor e imediatamente maior que $P_{apr}$ no início da execução do algoritmo. O novo valor de $P_{apr}$ obtido é, então, inserido na $B^+$-tree, e o ponto é armazenado na página de dados correspondente.

### Exclusão

O primeiro passo da exclusão de um ponto é se encontrar a partição $P$ a que ele pertence. O procedimento é idêntico ao adotado na inclusão.

Após a retirada do ponto, uma junção de partições será necessária se o número de pontos em $P$ mais o número de pontos em $compl(P)$ se tornar menor ou igual à capacidade máxima de uma página $(m)$. Neste caso, a partição $pai(P)$ é inserida na $B^+$-tree, e os pontos restantes de $P$ (caso existam) e de $compl(P)$ são associados a ela. $P$ é excluída da $B^+$-tree, assim como $compl(P)$, se for o caso (se $compl(P)$ não contém pontos ela não existe na $B^+$-tree). Como $compl(pai(P))$ pode não estar na $B^+$tree por não conter pontos (assim como os complementos de outros de seus ancestrais), a junção de partições pode continuar recursivamente, até que a soma do número de pontos de um ancestral de $P$ e do complemento desse ancestral seja maior que $m$, ou até que o primeiro nível seja atingido, isto é, aquele onde só existem as partições '0' e '1'.

### Consulta

A consulta para verificação da existência de um ponto se resume na identificação da partição a que ele pertence, seguida de uma busca na página associada.

Para a execução de *range queries* a estratégia básica consiste da identificação das partições com menor e maior identificador que podem intersectar a janela de consulta. Todas as partições cujos identificadores estão nesse intervalo podem conter pontos que estão dentro da janela. Em seguida, a G-tree é pesquisada, e cada partição contida no intervalo é testada para se determinar se ela está totalmente contida na janela, se a intersecta, mas não está contida, ou se a partição e a janela são disjuntas. Se a partição está completamente contida na janela, todos os seus pontos satisfazem a consulta. Por outro lado, se a partição intersecta a janela mas não está contida, então cada ponto da página correspondente deve ser examinado para se verificar se é pertencente à janela. Finalmente, se uma partição é disjunta da janela, seus pontos não precisam ser considerados.

Dada uma janela de consulta $q$, cuja extensão em cada dimensão $d$ de um espaço $k$-dimensional é dada por suas coordenadas inicial e final, $q_d^i$ e $q_d^f$ respectivamente, seu ponto de menor coordenada linear, $q_{min}$, é dado pelas coordenadas $(q_1^i, \ldots, q_k^i)$, e seu ponto de maior coordenada linear, $q_{max}$, pelas coordenadas $(q_1^f, \ldots, q_k^f)$. Esses são os pontos usados para se determinar o intervalo de identificadores das partições que podem intersectar a janela. Os valores dos extremos desse intervalo correspondem aos identificadores das

partições aproximadas que contêm $q_{min}$ e $q_{max}$, calculadas como nos algoritmos de inclusão e exclusão.

### 2.4.5 Filter Tree

A Filter Tree [SK96, KS97] é um método de acesso idealizado com o objetivo de executar eficientemente junções espaciais. Ela se baseia em três princípios:

- representação hierárquica do espaço;
- separação de objetos por tamanhos;
- localidade de acessos.

Para indexação de um conjunto de dados, os objetos são representados por seus MBRs, e o espaço onde eles se encontram é mapeado para outro, cujas coordenadas mínima e máxima em cada dimensão são, respectivamente, 0 e 1. As coordenadas dos MBRs e das janelas de consulta também são transformados nas mesmas proporções.

Em linhas gerais, o que este método faz é dividir o espaço através de uma hierarquia de grades regulares. A hierarquia possui $L+1$ níveis (de 0 a $L$), e em um dado nível $j$ a grade possui $4^j$ células de tamanho $\frac{1}{2^j} \times \frac{1}{2^j}$.

Cada objeto é associado ao nível mais baixo (aquele com o maior número) em que ele pode ser completamente sobreposto por uma única célula. Dado um objeto cujo MBR é representado pelos cantos inferior esquerdo $(x_1, y_1)$ e superior direito $(x_2, y_2)$, pode-se determinar o nível a que ele pertence a partir das representações binárias de $x_1$, $x_2$, $y_1$ e $y_2$. Seja $b_x$ o número de bits mais significativos comuns a $x_1$ e $x_2$, e $b_y$ o número de bits mais significativos comuns a $y_1$ e $y_2$. Então $n = min(b_x, b_y)$ é o nível ao qual o objeto deve ser associado. A figura 2.28 mostras três retângulos, suas coordenadas binárias e os níveis a que cada um pertence.

Com esse esquema, os retângulos maiores são associados aos níveis mais altos da hierarquia, enquanto os menores tendem a ser associados aos níveis mais baixos. No entanto, alguns retângulos pequenos podem ser associados aos níveis superiores, no caso de intersectarem as linhas de alguma grade logo nos primeiros níveis (veja o caso do retângulo $C$, na figura 2.28).

Para cada objeto é armazenada em um arquivo uma entrada denominada *descritor de entidade*, com as seguintes informações:

- a especificação do MBR do objeto $(x_1, y_1)$, $(x_2, y_2)$;
- a coordenada linear na curva de Hilbert associada ao centro do MBR $H(x_c, y_c)$, onde $x_c = \frac{x_2+x_1}{2}$ e $y_c = \frac{y_2+y_1}{2}$;

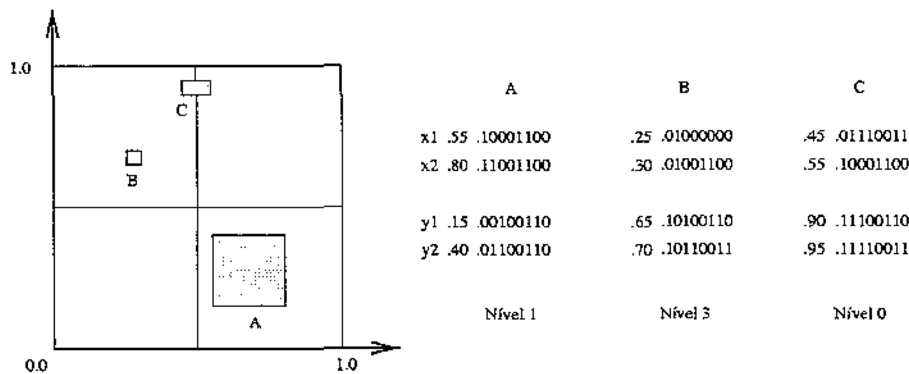



Figura 2.28: Cálculo dos níveis de um conjunto de retângulos.

- um ponteiro para a página de disco na qual estão armazenados os dados completos do objeto.

O arquivo de descritores de entidade está organizado de forma que:

- Os descritores de todas as entidades associadas a um determinado nível são armazenados juntos.
- Em cada nível, os descritores são ordenados pelas coordenadas lineares dos pontos centrais dos MBRs correspondentes, segundo uma curva de Hilbert. Dessa forma, os descritores de objetos pertencentes a uma determinada célula de um dado nível são armazenados juntos. Esse procedimento visa a otimização dos acessos a disco, já que objetos espacialmente próximos serão, muito provavelmente, recuperados juntos em uma consulta. Embora seja interessante que os registros de dados dos objetos também estejam ordenados da mesma forma, isso não é obrigatório.
- Os descritores são armazenados em blocos.
- Para cada nível da hierarquia há um índice de células, isto é, uma B-tree que armazena a coordenada linear do último descritor em cada bloco.

A figura 2.29 esquematiza a estrutura de uma Filter Tree.

Embora Sevcik e Koudas tratem apenas da construção da Filter Tree sobre conjuntos de dados estáticos, os autores sugerem, para o caso de dados dinâmicos, que um espaço livre seja deixado em cada bloco para permitir a execução eficiente de inclusões e alterações.

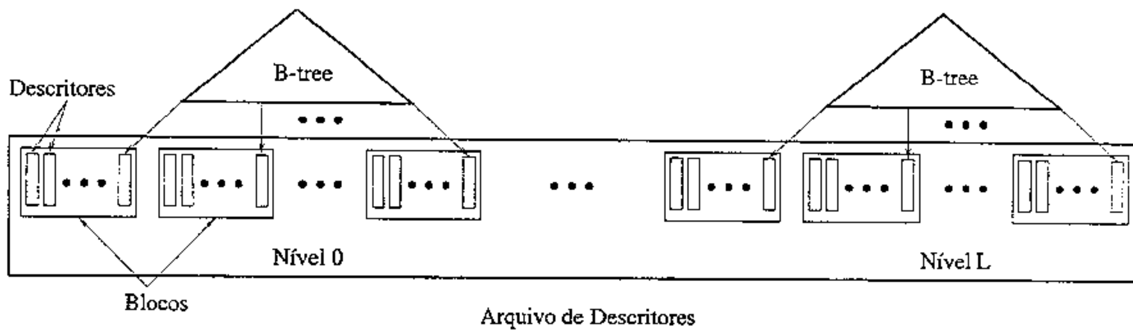



Figura 2.29: Estrutura de uma Filter Tree.

### Indexação de pontos

A indexação de pontos deve ser tratada de forma especial na Filter Tree, pois as coordenadas mínima e máxima de um ponto são iguais em cada uma das dimensões. Assim, se cada coordenada fosse representada por um número infinito de bits, não haveria como se determinar o nível correspondente na hierarquia. Embora as coordenadas sejam representadas por um número limitado de bits, de forma que sempre há um nível a ser associado aos pontos, o tratamento destes pode aumentar muito o número de níveis.

Para resolver o problema, Sevcik e Koudas propõem a adição de um novo nível na base da hierarquia, chamado de *nível infinito*, onde os pontos seriam acomodados.

### Consultas

A busca de um objeto na Filter Tree é feita pela identificação do nível em que ele se encontra, e por uma busca na B-tree correspondente, usando a coordenada linear do ponto central de seu MBR. Uma vez identificado e lido o bloco que contém o descritor do objeto no arquivo de descritores, basta encontrar o descritor associado ao objeto, o qual contém o apontador para sua área de dados.

A execução de *range queries* requer a verificação de todos os níveis da Filter Tree. Em cada nível, toda célula que intersecta a janela de consulta deve ser examinada. As outras não precisam ser verificadas, porque as células de um mesmo nível são disjuntas e os objetos associados a cada nível estão completamente contidos em uma de suas células. Então se uma célula não intersecta a janela de consulta, os objetos que ela contém também não intersectam. O conjunto de células a serem examinadas em cada nível forma um conjunto de intervalos de valores na curva de Hilbert. Uma vez que os intervalos foram identificados, o índice de células do nível correspondente pode ser usado para identificar os blocos do arquivo de descritores que contêm as entradas cujas coordenadas lineares estão contidas nos intervalos.

Segundo os autores, esse procedimento apresenta problemas para níveis muito baixos

na hierarquia (em torno de 10 para baixo), devido ao grande número de células, o que poderia gerar um grande número de intervalos. A alternativa é escolher um nível $c$, e o conjunto de intervalos gerados nesse nível é usado em todos os outros. A escolha de um alto valor para $c$ (correspondente a um nível baixo na hierarquia) faz com que o intervalos selecionados contenham uma menor área fora da janela de consulta que para valores pequenos. Em compensação, o número de células e, conseqüentemente, de intervalos, será maior. Os autores afirmam que a escolha apropriada depende mais do número de entidades armazenadas que das dimensões das janelas de consulta, e apresentam uma análise para a escolha do nível $c$.

**Junções espaciais**

O algoritmo de junção espacial é apropriado para a avaliação de predicados cuja provável satisfação é detectável a partir da existência de intersecção entre os MBRs dos objetos pertencentes aos conjuntos de dados envolvidos. Ele se baseia no fato de que, dados dois conjuntos de objetos representados por duas Filter Trees, $F_1$ e $F_2$, os objetos pertencentes a uma dada célula de $F_1$ só podem intersectar objetos de $F_2$ que estejam numa célula de mesmo nível e posição na grade, ou nas células de níveis superiores que a contêm, ou ainda nas células de níveis inferiores nela contidas. Mais uma vez, isto se deve ao fato das células de cada nível serem disjuntas e os objetos de um mesmo nível estarem inteiramente contidos em uma única célula.

A figura 2.30 ilustra o que foi dito. Cada hierarquia tem 4 níveis. Para determinar todos os objetos de $F_2$ que intersectam os objetos da célula 0 do nível 3 de $F_1$, apenas a célula 0 do mesmo nível em $F_2$ e as células que a contêm nos níveis 0, 1 e 2 precisam ser consideradas. A recíproca é verdadeira para identificar os objetos de $F_1$ que intersectam os da célula 0 do nível 3 de $F_2$. Da mesma forma, os objetos da célula 15 do nível 2 de $F_1$ só precisam ser testados contra os da sua correspondente em $F_2$ e os das células que a contêm, nos níveis 0 e 1 (se considerarmos que as células dos níveis inferiores foram processadas antes e que, portanto, qualquer intersecção entre objetos da célula 15 do nível 2 de $F_1$ e objetos de células do nível 3 de $F_2$ já foram detectados).

O algoritmo de junção espacial das Filter Trees é projetado para processar cada célula de cada árvore dessa forma e ainda permitir que cada bloco dos arquivos de descritores seja lido somente uma vez. Sejam $F_1$ e $F_2$ duas Filter Trees sobre as quais deve ser feita uma junção, e seja ${e_{lj}}^{F_i}$ a maior coordenada na curva de Hilbert de um descritor de entidade no $j$-ésimo bloco do nível $l$ da Filter Tree $F_i$. Em outras palavras, ${e_{lj}}^{F_i}$ é o marcador do fim desse bloco. Então existem tantos marcadores de fim de bloco quantos são os blocos nos arquivos de descritores de $F_1$ e $F_2$.

O algoritmo trabalha da seguinte forma. Suponha que todos os marcadores de ambas as hierarquias são classificados juntos, e os valores duplicados são excluídos, já que podem

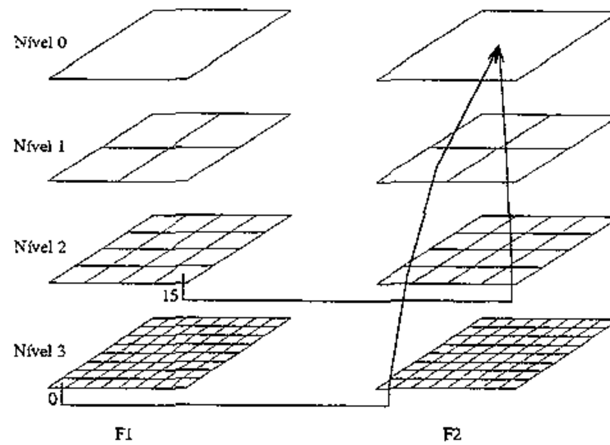



Figura 2.30: Junção espacial em uma Filter Tree.

haver marcadores iguais em ambas as Filter Trees. Após a classificação, cada par de marcadores sucessivos delimita um intervalo de coordenadas na curva de Hilbert que tem a propriedade de estar completamente contido em um único bloco em qualquer nível de qualquer das hierarquias. De fato, na pior das hipóteses, um intervalo é delimitado por dois marcadores do mesmo nível da mesma Filter Tree e, portanto, tem a extensão exata de um bloco dessa árvore. Como conseqüência, cada intervalo pode ser processado mantendo-se na memória apenas um bloco de cada nível de cada Filter Tree. Supondo que cada árvore tenha $L+1$ níveis, será necessário um *buffer* de leitura de $2L+2$ páginas. Na verdade, não há necessidade de fazer realmente a classificação, já que os marcadores de fim de bloco estão indexados por B-trees. Através desses índices é possível fazer a intercalação dos marcadores de cada árvore, a fim de se obter os intervalos.

A cada intervalo obtido, uma varredura é feita no arquivo de descritores das hierarquias em todos os níveis. Cada vez que um valor ${e_{lj}}^{F_i}$ é atingido na varredura e processado, o bloco $j$ do nível $l$ da árvore $F_i$ é substituído pelo bloco $j+1$ do mesmo nível e árvore. Seja ${S_l}^{F_i}(e_n, e_{n+1})$ o conjunto de objetos no nível $l$ da árvore $F_i$, cujas coordenadas lineares na curva de Hilbert estão no intervalo $(e_n, e_{n+1})$. Como cada intervalo está totalmente contido em um único bloco em todos os níveis das árvores, todos os descritores pertencentes a todos os conjuntos ${S_l}^{F_i}(e_n, e_{n+1})$ de um dado intervalo $(e_n, e_{n+1})$ estão na memória enquanto ele é processado. Então, quando esse intervalo é atingido na varredura, as seguintes ações são realizada para cada nível $l = 0, \ldots, L$:

- Os MBRs dos objetos pertencentes a ${S_l}^{F_1}(e_n, e_{n+1})$ são verificados contra os dos pertencentes a ${S_{l-j}}^{F_2}(e_n, e_{n+1})$, para $j = 0, \ldots, l$.

- Os MBRs dos objetos pertencentes a ${S_l}^{F_2}(e_n, e_{n+1})$ são verificados contra os dos pertencentes a ${S_{l-j}}^{F_1}(e_n, e_{n+1})$, para $j = 1, \ldots, l$. Aqui os valores de $j$ começam de

1 porque os objetos de $S_l^{F_2}(e_n, e_{n+1})$ já foram testados contra os de $S_l^{F_1}(e_n, e_{n+1})$ no passo anterior.

Como já foi dito, assim que o processamento do intervalo $(e_n, e_{n+1})$ é concluído, todos os objetos cujos descritores estão armazenados no bloco finalizado por $e_{n+1}$ já foram processados e, portanto, esse bloco já pode ser substituído pelo próximo de seu nível e árvore. O processamento continua, então, com o intervalo $(e_{n+1}, e_{n+2})$, e assim sucessivamente, até que o último intervalo tenha sido processado.

Esse algoritmo corresponde apenas à fase de filtragem da junção espacial, já que o predicado é avaliado apenas sobre os MBRs dos objetos. Na fase de refinamento são usados os mesmos algoritmos que em outros métodos de acesso.

## 2.5 Métodos derivados de árvores binárias

Os métodos derivados de árvores binárias estendem as características básicas dessas estruturas para o espaço $k$-dimensional, ou seja, a divisão recursiva e hierárquica do espaço em sub-regiões disjuntas e a representação dessa hierarquia através de uma árvore.

Apesar de existirem várias propostas de índices espaciais derivados de árvores binárias na literatura, algumas deficiências herdadas da estrutura de origem dificultam sua utilização em sistemas gerenciadores de bancos de dados, a saber:

- As árvores binárias não são balanceadas[6], o que significa que a profundidade de cada sub-árvore depende da distribuição e da ordem em que os dados são inseridos. Numa árvore mal construída, o número médio de nós visitados para se encontrar um determinado ponto pode ser $O(n)$, onde $n$ é o número de pontos indexados, contra $O(log\ n)$ numa árvore balanceada. Embora existam algoritmos para balanceamento de árvores binárias, assim como para algumas de suas derivadas espaciais, sua execução exige o conhecimento prévio dos dados a serem indexados.

- As árvores binárias são estáticas, isto é, o desempenho da estrutura pode ser degradado com as inserções e exclusões no conjunto de dados. Assim, mesmo que a árvore seja inicialmente balanceada, dependendo das atualizações que sofre pode degenerar até mesmo para uma lista ligada de nós, e, mais uma vez, o tempo de consulta pode se tornar $O(n)$. Para que este problema seja minimizado é necessário se fazer reorganizações periódicas na estrutura, o que nem sempre é possível, pois essa operação indisponibiliza o índice. Essa característica e a anterior tornam as árvores

[6] Uma árvore é balanceada quando a profundidade da sub-árvore esquerda de cada nó nunca difere de mais de um nível da profundidade de sua sub-árvore direita.

binárias, e conseqüentemente os métodos espaciais delas derivados, impróprias para o gerenciamento de dados que sofram atualizações freqüentes.

- As árvores binárias têm baixo *fan-out*[7], o que as faz inapropriadas para a indexação de dados armazenados em disco. Tomemos como exemplo os métodos de acesso espaciais específicos para a indexação de dados no espaço bidimensional. Esses métodos têm, dependendo de sua estrutura, no máximo dois ou quatro entradas por nó, cada uma correspondendo a um nó filho. Como cada entrada tem poucos *bytes*, o tamanho total de um nó é pequeno demais para que ele seja armazenado como uma página de disco. Uma alternativa para reduzir o problema é armazenar vários nós em uma mesma página, como está esquematizado na figura 2.31.

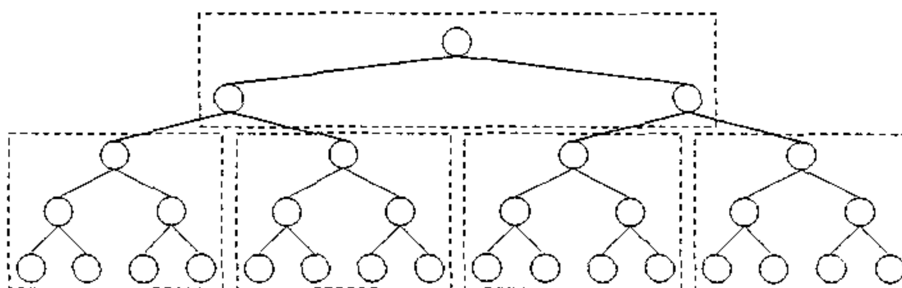


Figura 2.31: Agrupamento de nós de uma árvore binária em páginas de disco.

O funcionamento de dois métodos pertencentes a este grupo, a point quadtree e a k-d tree, é discutido a seguir. Projetados para a indexação de pontos, estes dois métodos de acesso são bastante limitados, pois armazenam apenas um ponto por nó, mas serviram como base para a criação de índices mais elaborados como a MX-CIF Quadtree e a Spatial k-d tree [Ooi90], desenvolvidos para a indexação de retângulos.

## 2.5.1 Point Quadtree

A Point Quadtree, inicialmente chamada simplesmente de Quadtree, é um método de acesso proposto por Finkel e Bentley em [FB74] para a indexação de pontos dispostos num espaço bidimensional. Embora a generalização para espaços de maior dimensionalidade seja direta, toda a exposição será feita supondo que o espaço a ser indexado é bidimensional, por simplicidade.

A Point Quadtree se baseia na decomposição recursiva do espaço em quatro sub-regiões, denominadas "quadrantes". Utilizando a classificação de Sellis, Roussopoulos e Faloutsos [SRF87][8], esse método de acesso pode ser caracterizado da seguinte forma:

[7] Número máximo de filhos de um nó.
[8] Veja a seção 2.3.1.

- é adaptável, isto é, cada ponto inserido determina a posição das linhas que dividem o espaço;
- particiona o espaço em todas as dimensões;
- faz uma decomposição hierárquica, ou seja, a divisão acarretada pela inserção de um ponto é restrita ao sub-espaço que o contém.

Os pontos são organizados em uma árvore onde cada nó armazena:

- as coordenadas de um único ponto, que servem de base para a divisão do espaço;
- um apontador para o registro de dados associado ao ponto. Finkel e Bentley partem do princípio de que não existem dois pontos com as mesmas coordenadas. Para os casos em que colisões são permitidas os autores sugerem que o apontador referencie alguma estrutura (como uma lista encadeada, por exemplo) que armazene os pontos cujas coordenas coincidam com as armazenadas no nó.
- apontadores para 4 sub-árvores que representam os 4 quadrantes em que o espaço é dividido. A esses quadrantes são dados os nomes de nordeste, noroeste, sudoeste e sudeste, correspondentes às sub-árvores 1, 2, 3 e 4 respectivamente[9].

**Inserção**

O primeiro ponto inserido na estrutura é armazenado na raiz da árvore, e suas coordenadas servem como origem para a primeira divisão do espaço. Para se inserir os pontos subseqüentes, a árvore é percorrida de acordo com o posicionamento do ponto em questão com relação aos quadrantes de cada nó visitado. Os pontos que porventura recaiam exatamente sobre uma das linhas de divisão são considerados como pertencentes ao quadrante 1 ou ao 3, dependendo do caso. Cada novo ponto inserido divide o sub-espaço onde se encontra em quatro novos sub-espaços.

Em linhas gerais, o procedimento de inserção pode ser descrito da seguinte forma:

```
se a árvore está vazia então
   insira o ponto na raiz
senão {
   faça da raiz o nó corrente;
   enquanto o ponto ainda não tiver sido inserido faça {
       determine o quadrante em que está o ponto a ser inserido
```

[9] Num espaço $k$-dimensional, cada nó de uma quadtree tem $2^k$ filhos, correspondentes às $2^k$ regiões em que é dividido o espaço em cada nível da estrutura.

```
            em relação ao ponto representado pelo nó corrente;
            se a sub-árvore do nó corrente correspondente a este
               quadrante está vazia então
               insira um nó nessa sub-árvore para o ponto em questão
            senão
               faça da raiz dessa sub-árvore o nó corrente;
    }
}
```

A figura 2.32 mostra o processo de construção de uma Point Quadtree.

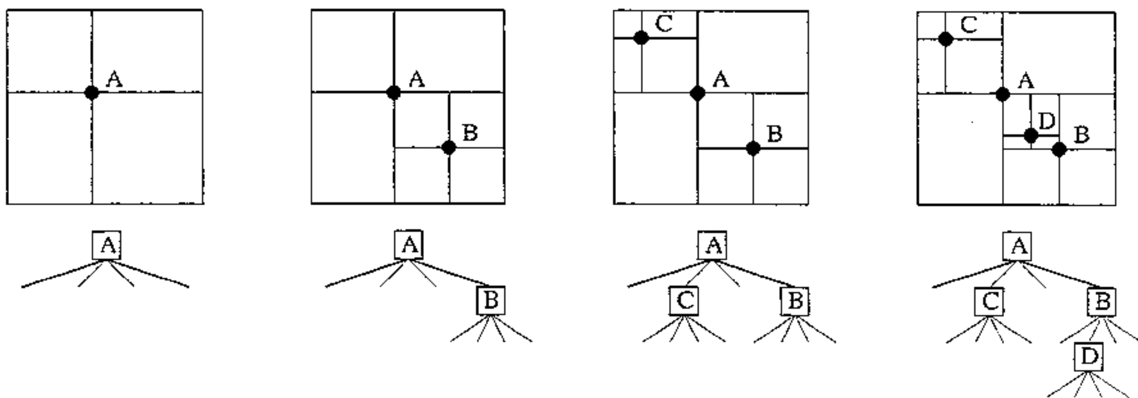



Figura 2.32: Inserção de pontos em uma point quadtree.

### Consulta

A verificação da existência de um ponto na estrutura é feita através de uma pesquisa na árvore, nos mesmos moldes daquela realizada pelo algoritmo de inserção.

O algoritmo para a avaliação de *range queries* é simples, e consiste de dois passos executados a partir da raiz:

- verificar se o ponto armazenado no nó corrente pertence à janela de consulta, e em caso afirmativo, reportá-lo como parte da resposta;
- chamar recursivamente o procedimento para cada sub-árvore não vazia e cujo quadrante correspondente intersecte a janela.

### Exclusão

O algoritmo de exclusão sugerido em [FB74] faz a reinserção de todos os nós pertencentes às sub-árvores do nó excluído, o que pode ser bastante ineficiente, dependendo da posição

do nó removido (o pior caso é a exclusão da raiz, onde todos os outros nós da árvore são reinseridos).

Uma alternativa melhor foi apresentada por Samet em [Sam80]. O algoritmo procura minimizar o número de nós a serem reinseridos substituindo o nó excluído por outro suficientemente "próximo", escolhido entre aqueles pertencentes às suas sub-árvores. A figura 2.33 ajuda a entender o motivo. Seja $A$ o nó a ser excluído, e $H$ o nó que irá substituí-lo como raiz da sub-árvore. Todos os nós pertencentes à área em destaque deverão ser reinseridos, pois mudarão de quadrante quando $H$ passar a ser a raiz. O nó $I$, por exemplo, que pertence à sub-árvore sudoeste de $A$ passará a estar localizado no quadrante noroeste de $H$, e deverá ser reinserido na sub-árvore correspondente. Os demais continuarão nos quadrantes em que se encontram e, portanto, não precisarão ser reinseridos. Assim, quanto mais próximos estiverem o ponto a ser removido e seu substituto, menor será essa área e, conseqüentemente, menor o número de nós a serem reinseridos (se considerarmos uma distribuição uniforme de pontos no espaço).

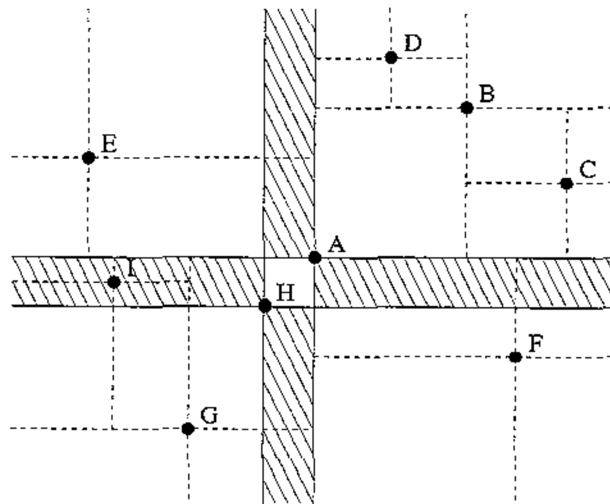



Figura 2.33: Área onde se localizam os nós que devem ser reinseridos com a exclusão do nó $A$ e sua substituição pelo nó $H$.

A fim de se escolher o substituto, inicialmente são identificados quatro pontos candidatos, um em cada sub-árvore do nó a ser excluído. O entendimento do processo de escolha dos candidatos requer a introdução do conceito de quadrante oposto. Dado um quadrante $i$, $oposto(i) = ((i + 1) \ mod \ 4) + 1$, ou seja, o primeiro e o terceiro quadrantes são opostos entre si, assim como o segundo e o quarto. Para encontrar o candidato de um quadrante $i$ de um nó a ser excluído, o algoritmo percorre a árvore a partir da raiz desse quadrante, sempre pela sub-árvore correspondente a $oposto(i)$, até encontrar uma sub-árvore vazia. O último nó visitado é o candidato do quadrante $i$. Dois critérios são, então, usados na escolha entre os candidatos:

- Escolhe-se o candidato mais próximo de ambos os eixos com relação aos candidatos dos quadrantes adjacentes. Existem condições em que esse critério é ineficaz, ou porque nenhum dos pontos o satisfaz, como mostra a figura 2.34a, ou porque ele é satisfeito por mais de um ponto, como $B$ e $D$, na figura 2.34b. Nestes casos, é utilizado o segundo critério.
- Escolhe-se o candidato com menor soma das distâncias horizontal e vertical para o nó a ser excluído.

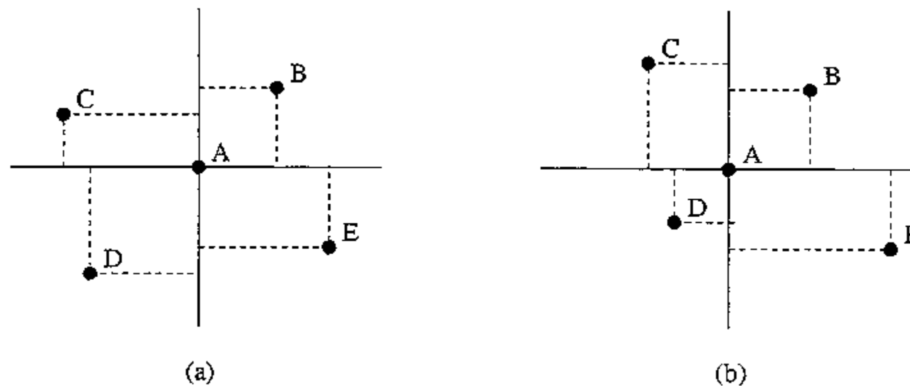



Figura 2.34: Dois casos onde o primeiro critério de escolha entre os candidatos falha: (a) nenhum nó satisfaz o critério; (b) dois nós satisfazem o critério.

Uma vez escolhido o substituto, este passa a ser a raiz da sub-árvore do nó excluído, e os nós pertencentes à referida área são reinseridos.

### 2.5.2 K-d tree

A k-d tree, proposta por Bentley em [Ben75], é um método de acesso a pontos dispostos em um espaço $k$-dimensional. A exemplo da point quadtree, ela se baseia na decomposição recursiva do espaço, orientada pela posição de cada ponto inserido e restrita ao sub-espaço que o contém. A diferença é que na k-d tree a divisão é feita em apenas uma das dimensões a cada nível da árvore, ao contrário da point quadtree, onde essa partição é feita em todas as dimensões. Assim, a cada ponto inserido, o sub-espaço a que ele pertence é dividido em dois novos sub-espaços.

Os pontos são armazenados em uma árvore, onde em cada nó são mantidas as seguintes informações:

- as $k$ coordenadas de um ponto, que aqui chamaremos de $P$ (assim como na point quadtree, cada nó de uma k-d tree armazena somente um ponto);

- um apontador para o registro de dados associado àquele ponto. Bentley não trata dos casos em que dois ou mais pontos tenham as mesmas coordenadas, mas uma solução semelhante à apresentada em [FB74] para as point quadtrees poderia ser utilizada;

- um discriminador, denominado $DISC(P)$, que indica em que dimensão o espaço está sendo particionado naquele nível da árvore. O armazenamento deste discriminador não é obrigatório, pois seu valor pode ser deduzido do nível que o nó ocupa na árvore. Supondo que o espaço considerado tenha $k$ dimensões, numeradas de 0 a $k-1$, e considerando que a raiz se encontra no nível 0, o valor do discriminador em um nó do nível $i$ é $(i \ mod \ k)$.

- dois apontadores para suas sub-árvores:

  - $LOSON(P)$, que referencia a sub-árvore de $P$ onde estão armazenados os nós cujo valor de coordenada na dimensão denotada por $DISC(P)$ é menor que o valor da coordenada de $P$;
  - $HISON(P)$, que referencia a sub-árvore de $P$ onde estão armazenadas os nós cujo valor de coordenada na dimensão denotada por $DISC(P)$ é maior que o valor da coordenada de $P$.

No caso de um ponto $Q$ subordinado a $P$ ter a mesma coordenada deste na dimensão $DISC(P)$, a decisão de armazená-lo em uma ou outra sub-árvore é feita a partir da aplicação de uma função sobre as coordenadas de $P$ e $Q$ em todas as dimensões, e da comparação dos dois resultados. Sejam $C_0(P), \ldots, C_{k-1}(P)$, as $k$ coordenadas do ponto $P$. Define-se como *superchave* de $P$ na dimensão $i$ como $S_i(P) = C_i(P)C_{i+1}(P) \ldots C_{k-1}(P)C_0(P) \ldots C_{i-1}(P)$, isto é, a concatenação cíclica de todas as coordenadas de $P$ começando com $C_i$. Então se $DISC(P) = i$ e $C_i(P) = C_i(Q)$, $Q$ será colocado na árvore apontada por $LOSON(P)$ se $S_i(Q) < S_i(P)$. Se $S_i(Q) > S_i(P)$, $Q$ será colocado na árvore apontada por $HISON(P)$.

### Inserção

Para se inserir um ponto numa k-d tree percorre-se a árvore comparando a coordenada do ponto a ser inserido com a do ponto armazenado em cada nó, na dimensão dada pelo seu discriminador $DISC$. Lembrando que o algoritmo de inserção pressupõe que não existem dois pontos com as mesmas coordenadas, pode-se resumi-lo da seguinte forma:

```
INSERE(P, RAIZ); /* P - ponto a ser inserido
                    RAIZ - ponteiro para a raiz da k-d tree */
```

```
se RAIZ = NULO então
   insira P na raiz
senão {
   no_corrente ← RAIZ;
   enquanto P ainda não tiver sido inserido faça {
      d ← DISC(no_corrente);
      se C_d(P) < C_d(no_corrente) ou S_d(P) < S_d(no_corrente) então
         se LOSON(no_corrente) = NULO então /* sub-árvore vazia */
            insira um nó para P na sub-árvore apontada por
            LOSON(no_corrente)
         senão
            no_corrente ← LOSON(no_corrente)
      senão
         se HISON(no_corrente) = NULO então /* sub-árvore vazia */
            insira um nó para P na sub-árvore apontada por
            HISON(no_corrente)
         senão
            no_corrente ← HISON(no_corrente);
   }
}
```

A figura 2.35 exemplifica a inserção em uma k-d tree.

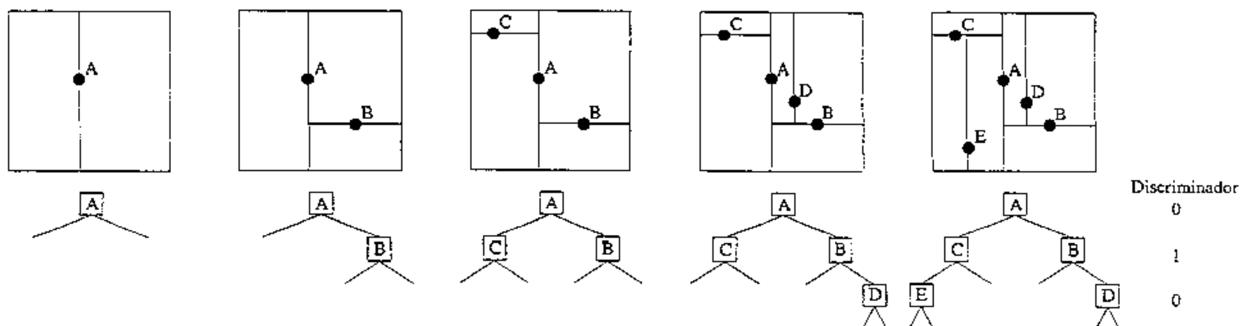



Figura 2.35: Inserção de pontos em uma k-d tree. A sub-árvore da direita corresponde a $LOSON$, e a da esquerda a $HISON$.

### Consulta

O algoritmo para determinar a existência de um ponto na estrutura percorre a árvore de maneira semelhante ao algoritmo de inserção até encontrar um nó com as coordenadas

especificadas, caso em que retorna com sucesso, ou até encontrar um apontador nulo, o que indica a ausência do ponto procurado.

Na execução de *range queries*, a pesquisa na árvore é feita de forma que, a cada nível, apenas as sub-árvores que representam sub-espaços que intersectam a janela de consulta sejam percorridas. A cada nó visitado, o algoritmo verifica, então, se o ponto que ele armazena pertence à área da janela.

**Exclusão**

A exclusão de um nó cujas sub-árvores são ambas vazias não apresenta qualquer complicação, podendo ser feita diretamente. No entanto, ao se remover um nó $P$ que possui descendentes e que, portanto, é a raiz de uma sub-árvore, deve-se substituí-lo por um de seus descendentes, digamos $Q$, de tal forma que, após a substituição, todos os nós que estavam na sub-árvore apontada por $LOSON(P)$ estejam na sub-árvore apontada por $LOSON(Q)$, e todos os que estavam na sub-árvore apontada por $HISON(P)$ estejam na sub-árvore apontada por $HISON(Q)$. Para que isto seja possível, o nó $Q$ deve ser aquele com o valor de coordenada na dimensão designada por $DISC(P)$ o mais próximo possível do valor coordenada de $P$ em qualquer uma de suas sub-árvores. Em outras palavras, $Q$ deve ser o nó com o maior valor de coordenada na dimensão dada por $DISC(P)$ na sub-árvore apontada por LOSON(P), ou o nó com o menor valor dessa coordenada na sub-árvore apontada por HISON(P). Como qualquer um dos dois pode ser escolhido (se existirem os dois), Bentley sugere que a escolha do substituto seja feita alternadamente entre uma sub-árvore e outra, cada vez que é feita uma exclusão.

Como a substituição de $P$ por $Q$ implica na exclusão deste da posição onde ele se encontrava, é necessário achar um substituto também para $Q$ em suas próprias sub-árvores, caso elas não sejam vazias. Assim, o algoritmo de exclusão é recursivo, e as chamadas recursivas só terminam quando um nó escolhido como substituto de seu ascendente tem sub-árvores vazias. Neste caso ele é simplesmente excluído de sua posição e, de baixo para cima, cada nó na cadeia formada é substituído por seu descendente escolhido, até que o nó $P$ que se desejava excluir tenha sido substituído.

## 2.6 Métodos derivados de estruturas *hash*

A idéia básica das organizações de arquivos que utilizam *hashing* é a aplicação de uma função sobre os valores de um ou mais atributos de cada registro, resultando no endereço do bloco ou *bucket* de disco em que ele está (ou deve ser) armazenado [EN94]. Os atributos cujos valores são usados no cálculo desse endereço compõem o que se chama de *chave de randomização*.

O inconveniente deste tipo de endereçamento é que o número de blocos ou *buckets* deve ser mantido fixo, pois é usado como parâmetro da função de randomização. Para permitir a expansão e redução dinâmica do arquivo, em algumas técnicas de *hashing* a correspondência entre o valor da chave de randomização e o endereço do bloco é feita por intermédio de uma estrutura conhecida como *diretório*, normalmente implementada como um vetor ou uma árvore binária. Nesses casos, a função de *hashing* é utilizada para o cálculo, a partir da chave, da entrada correspondente no diretório, que por sua vez contém o endereço do bloco apropriado. Toda vez que é ultrapassada a capacidade de armazenamento de um bloco, ele é dividido em dois, e o diretório é reorganizado para refletir a mudança. Da mesma forma, dois blocos podem ser substituídos por um, caso estejam com a ocupação abaixo de um limite mínimo.

Os métodos de acesso espaciais derivados de estruturas *hash* usam como chave de randomização as coordenadas dos objetos, e têm a preocupação de mapear objetos espacialmente próximos para o mesmo bloco ou *bucket*, na medida do possível, de forma a otimizar a execução de *range queries*. Aqui discutiremos o funcionamento de um desses métodos: o Grid File.

### 2.6.1 Grid File

O Grid File [NHS84] é um método de acesso a pontos projetado com dois objetivos: permitir a execução de *point queries* em um espaço $k$-dimensional com apenas dois acessos a disco e executar *range queries* eficientemente. Os princípios básicos dessa estrutura são:

- a divisão do espaço em células hiper-retangulares, chamadas *grid blocks*, pela inserção de hiperplanos $(k-1)$-dimensionais paralelos aos eixos de cada dimensão;
- a associação de cada célula a um bloco (página de dados) ou *bucket* de disco através de um *diretório*. Todos os pontos pertencentes a uma determinada célula são armazenados no bloco a ela associado.

O diretório é composto por dois tipos de estruturas:

- as *escalas*, que são $k$ vetores unidimensionais, cada qual armazenando a posição dos hiperplanos que cortam o espaço em uma determinada dimensão;
- o *vetor de grade*, uma matriz $k$-dimensional onde cada elemento corresponde a uma célula em que o espaço foi particionado. Cada entrada do diretório mantém o endereço de um bloco ou *bucket* de disco onde estão armazenados os pontos da célula correspondente. Para evitar a sub-utilização dos blocos de disco, duas ou mais células do diretório podem referenciar o mesmo bloco, desde que o número total de

pontos que elas contêm não ultrapasse a capacidade máxima de armazenamento, e que a região coberta por elas seja um hiper-retângulo. O conjunto de todas as células associadas a um bloco $B$ é chamada *região de B*. Devido à possibilidade de o vetor de grade requerer um grande espaço, normalmente ele é armazenado em memória secundária, enquanto que as escalas são mantidas na memória primária.

A figura 2.36 esquematiza a organização de um Grid File com capacidade máxima de armazenamento de três pontos por bloco.

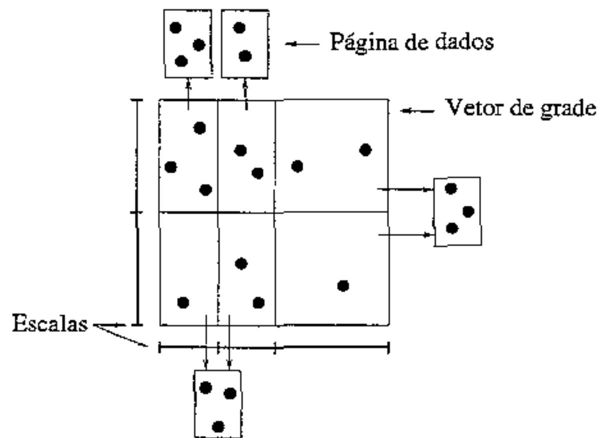



Figura 2.36: Estrutura de um grid file.

### Inserção

Para identificar o bloco onde um dado ponto $P$ deve ser inserido, as coordenadas deste são comparadas com as coordenadas de inserção dos hiperplanos armazenadas nas escalas, de forma a determinar a célula que o contém. A entrada correspondente no diretório é, então, lida do disco e, de posse do endereço que ela armazena, o bloco é também recuperado. Como se pode ver, são necessários apenas dois acessos a disco para se obter o bloco onde um ponto deve ser armazenado, o que, como foi dito, é um dos objetivos da estrutura.

Se o bloco em questão, que chamaremos de $B$, já está totalmente ocupado, a região de $B$ deve ser dividida em duas sub-regiões, e cada uma dessas deve ser associada a um bloco diferente. Os pontos pertencentes a cada uma das sub-regiões são, então, armazenados nos blocos correspondentes. Dois casos podem ocorrer:

- Se a região de $B$ contém apenas uma célula, então é necessário se inserir um novo hiperplano dividindo-a. A escolha da dimensão onde este hiperplano será inserido é feita de forma cíclica, isto é, se a última partição foi feita na dimensão $d$, então a dimensão $(d + 1)$ *mod* $k$ será usada. De acordo com o proposto em [NHS84],

o hiperplano é inserido no ponto médio do intervalo a ser particionado naquela dimensão. Com a inserção do novo hiperplano, deve-se criar uma entrada na escala correspondente à dimensão escolhida para armazenar a coordenada em que o espaço é dividido.

- Caso a região de $B$ tenha mais de uma célula, então já existe pelo menos um hiperplano que a corta, e a inserção de um novo hiperplano não é necessária. A operação de split se resume, então na criação do novo bloco de disco e na realocação dos pontos de acordo com o hiperplano já existente.

A figura 2.37 ilustra a inserção de pontos nos dois casos citados. A inserção do ponto $p_1$ na região do bloco $C$ causa a inserção de um novo hiperplano, pois ela contém apenas uma célula. Note que, como conseqüência, a região do bloco $A$ passou a conter duas células. O ponto $p_2$ inserido na região do bloco $B$ não acarreta nova divisão do espaço, uma vez que esta é composta por duas células, já separadas por um hiperplano.

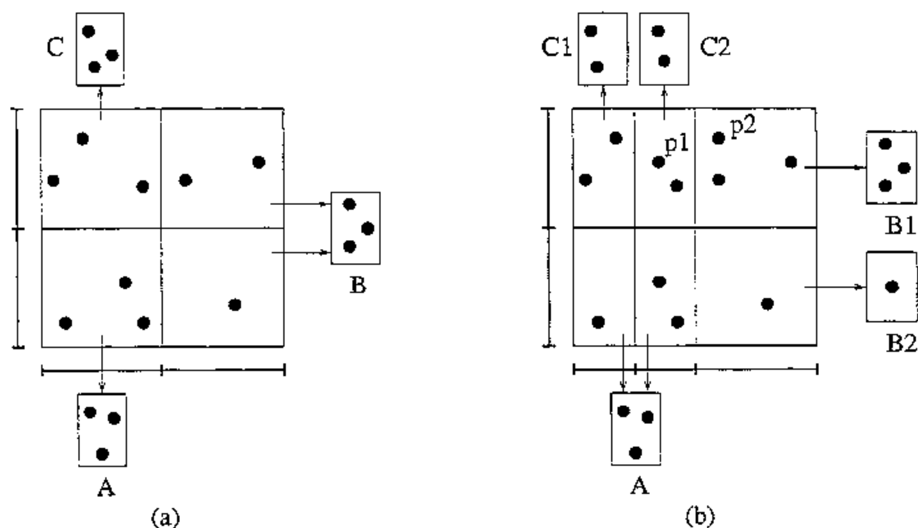



Figura 2.37: Tratamento de *overflow* em um Grid File com capacidade máxima de 3 pontos por bloco.

Para permitir o cálculo do endereço no disco de uma entrada do vetor de grade a partir das escalas, Nievergelt, Hinterberger e Sevcik sugerem que todas as células tenham um só tamanho. Isto significa que cada vez em que é necessário se inserir um hiperplano dividindo uma célula em uma determinada dimensão, todas as outras células também terão que ser divididas nessa mesma dimensão. Como conseqüência, a cada vez que é necessário se realizar um *split* de uma célula, o diretório dobra seu tamanho. A figura 2.38 exemplifica esse procedimento. Quando o bloco $C$ sofre um *split*, sua região é dividida em duas pela inserção de um hiperplano perpendicular ao eixo horizontal. Isto acarreta a

divisão também das células da região do bloco $B$, para manter a uniformidade no tamanho das células.

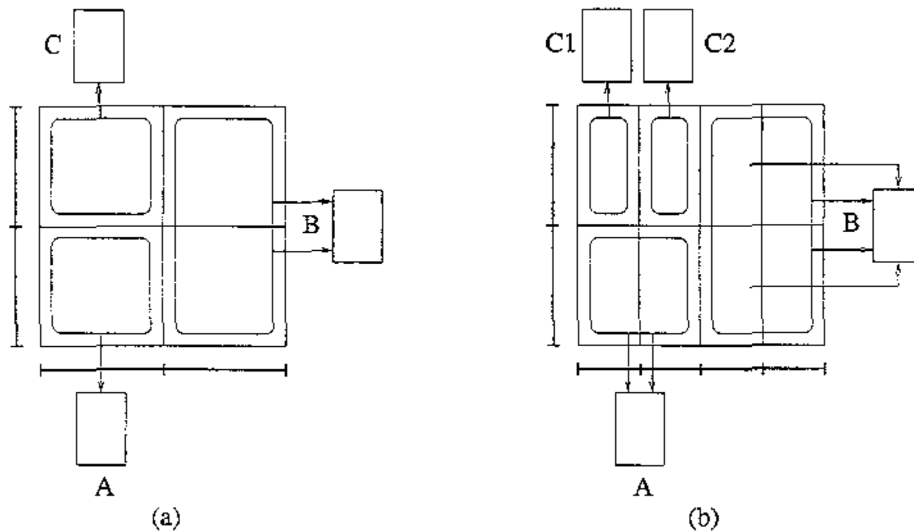



Figura 2.38: Diretório de um Grid File onde o tamanho das células é fixo.

### Consulta

A verificação da existência de um ponto na estrutura é feita da mesma forma que na determinação do bloco em que um ponto deve ser incluído, ou seja, a determinação da célula que o contém, através das escalas, a leitura da entrada correspondente no diretório e, finalmente, a recuperação da página de dados. Como na inserção, apenas dois acessos a disco são necessários.

A execução de *range queries* se resume na identificação, por meio das escalas, das células que intersectam a janela de consulta, com posterior recuperação das entradas do diretório e das páginas de dados correspondentes. Nas páginas associadas a células totalmente contidas na janela de consulta, todos os pontos são reportados, enquanto que naquelas associadas a células apenas parcialmente cobertas, cada ponto deve ser verificado antes de ser incluído na resposta.

### Exclusão

Durante a remoção de um ponto, o bloco onde ele estava armazenado pode, caso atinja uma baixa utilização, sofrer uma junção com outro bloco. As condições impostas em [NHS84] para que esta junção aconteça são:

- a nova região a ser formada pela junção das regiões associadas aos blocos a serem unidos deve ser um hiper-retângulo;

- o soma do número de pontos armazenados nos dois blocos não deve ultrapassar um limite pré-determinado. Em [NHS84] o limite proposto é de 70% da ocupação máxima de um bloco. A intenção é não criar uma região que logo teria que ser novamente dividida.

Em princípio, um bloco pode sofrer uma junção com qualquer bloco associado a uma região vizinha à sua[10] desde que as condições citadas sejam satisfeitas. No entanto, [NHS84] apresenta duas políticas para a escolha dos blocos *candidatos* a sofrerem a junção com o bloco de onde o ponto foi excluído:

- *neighbor system:* o bloco pode sofrer uma junção com qualquer dos seus vizinhos nas $k$ dimensões, observadas as condições anteriores;

- *buddy system:* neste caso um bloco só pode ser unido a um bloco específico em cada dimensão: seu *buddy*. Dois blocos são considerados *buddies* um do outro se suas regiões podem ser obtidas pela divisão de uma região superior ao meio.

A figura 2.39 mostra algumas junções válidas em cada sistema. Note que as opções existentes no *buddy system* são um subconjunto das possibilidades do *neighbor system.*

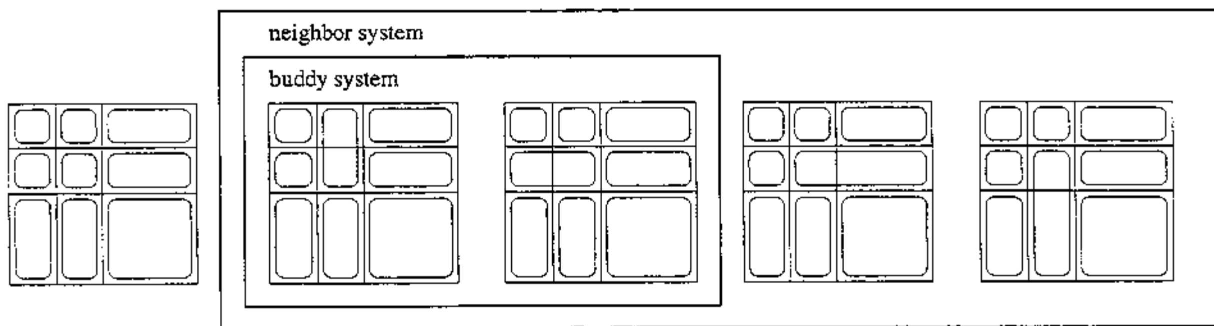



Figura 2.39: A região em destaque na primeira grade representa um bloco que deve sofrer uma junção. As demais grades mostram as alterações permitidas nos sistemas *buddy* e *neighbor.*

A junção de dois blocos torna desnecessária a existência do hiperplano que separava suas regiões quando ele não separa nenhum outro par de regiões. A figura 2.40 apresenta esta situação. Quando os pontos $p_1$ e $p_2$ são excluídos, a soma do número de pontos nos blocos $D$ e $E$ fica abaixo de 70% da ocupação máxima de um bloco, que é de 3 pontos, e eles são, portanto, unidos. O hiperplano que separava suas regiões torna-se redundante, pois as outras duas células a ele adjacentes pertencem a uma mesma região (a do bloco

[10] Chamaremos esses blocos de *blocos vizinhos.*

$A$). Dessa forma, a coordenada de inserção do hiperplano deve ser retirada da escala correspondente, e o vetor de grade ajustado. Por outro lado, a exclusão de $p_3$ e $p_4$, apesar de causar a junção dos blocos $B$ e $C$, não acarreta a retirada do hiperplano que separava as duas regiões, já que ele ainda separa as regiões dos blocos $A$ e $D$.

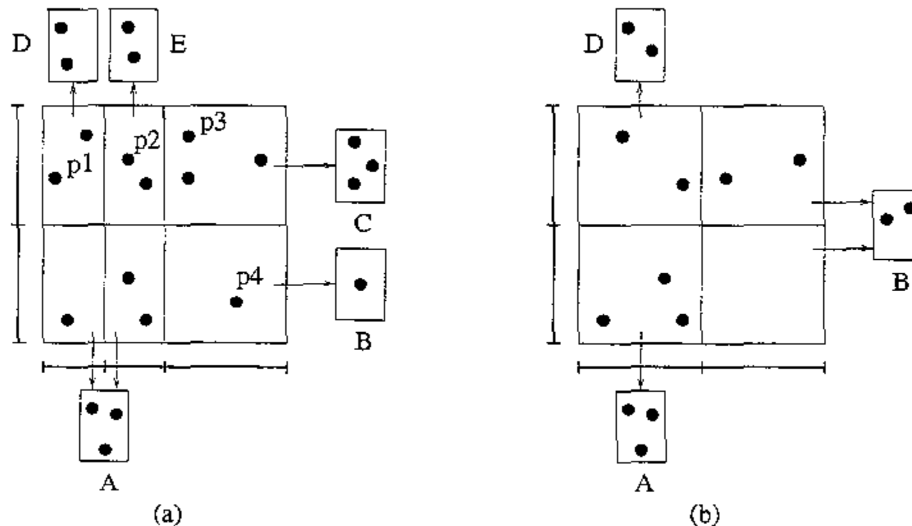



Figura 2.40: Exclusão de pontos em um Grid File com junção de blocos.

É bom notar que se o diretório é implementado de forma que todas as células tenham o mesmo tamanho, a retirada de um hiperplano que corta uma determinada dimensão só é possível se a toda a grade puder ser reorganizada para que o novo comprimento de intervalo (maior) seja usado na escala correspondente.

**Indexação de objetos de dimensão não zero**

O Grid File pode ser utilizado para indexar objetos de dimensão não zero através da transformação de seus MBRs em pontos num espaço $2k$-dimensional. Outra possibilidade seria usar a técnica da *abstração*, apresentada na seção 2.3.1.

## 2.7 Métodos derivados de árvores multiárias

As árvores multiárias se caracterizam pela capacidade de seus nós armazenarem entradas referentes a vários filhos, em contraposição às árvores binárias, cujos nós têm no máximo dois. Esta propriedade torna as árvores multiárias adequadas ao gerenciamento de dados em memória secundária, pois cada nó pode ser facilmente associado a uma página de disco, bastando para isso que a escolha do número máximo de entradas por nó seja baseada no tamanho da página e no tamanho de cada entrada. Além desta característica, duas

outras contribuíram para que as árvores multiárias se tornassem largamente utilizadas na indexação de dados convencionais:

- as árvores multiárias são balanceadas, isto é, todas as folhas aparecem no mesmo nível. Isto garante que mesmo em distribuições irregulares de dados um registro pode ser encontrado na estrutura em tempo logarítmico;
- as árvores multiárias são dinâmicas, ou seja, seu desempenho não é degradado por atualizações no conjunto de dados, de forma que reorganizações periódicas não são necessárias.

Os principais métodos de acesso espaciais derivados de árvores multiárias pertencem à família das R-trees, que são uma generalização das B-trees para espaços $k$-dimensionais. O funcionamento dos membros dessa família são discutidos nas próximas seções.

### 2.7.1 R-tree

A R-tree, proposta por Guttman [Gut84], é uma estrutura que agrupa os MBRs dos objetos representados em um espaço $k$-dimensional através de uma hierarquia de retângulos. A árvore possui dois tipos de nó:

- nós folha, que armazenam as informações referentes aos objetos indexados. Contêm um conjunto de entradas da forma *(identificador, MBR)*, onde o primeiro campo é o identificador de um objeto, e o segundo é o retângulo envolvente mínimo $k$-dimensional desse objeto;
- nós não-folha, que contêm entradas da forma *(ponteiro, MBR)*, onde *ponteiro* é o endereço de um nó subordinado, e *MBR* é o menor retângulo $k$-dimensional com lados paralelos aos eixos que envolve os retângulos de todas as entradas desse nó subordinado.

Cada nó é armazenado em uma página de disco. O número mínimo de entradas em um nó é dado por $m \leq \frac{M}{2}$, onde $M$ é o número máximo de entradas que um nó comporta. As seguintes propriedades são observadas na construção de uma R-tree:

- cada nó folha contém entre $m$ e $M$ entradas, a menos que seja a raiz;
- para cada entrada *(identificador, MBR)* em um nó folha, *MBR* é o menor retângulo que contém espacialmente o objeto $k$-dimensional por ela representado;
- cada nó não-folha tem entre $m$ e $M$ entradas, a menos que seja a raiz;

- para cada entrada *(ponteiro, MBR)* em um nó não-folha, *MBR* é o menor retângulo que contém espacialmente os retângulos das entradas do nó subordinado que ela referencia;
- a raiz tem pelo menos duas entradas, a menos que seja uma folha;
- todas as folhas aparecem no mesmo nível.

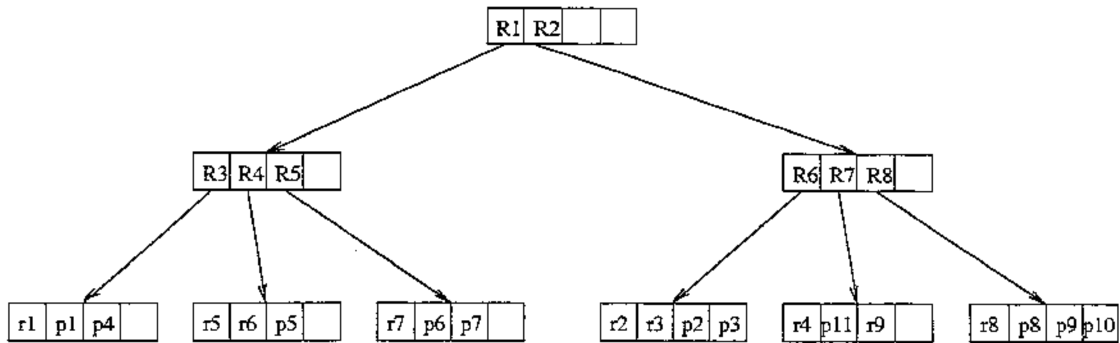
A figura 2.41 mostra a estrutura de uma R-tree. Note que os retângulos de um mesmo nível podem se intersectar.

### Inserção

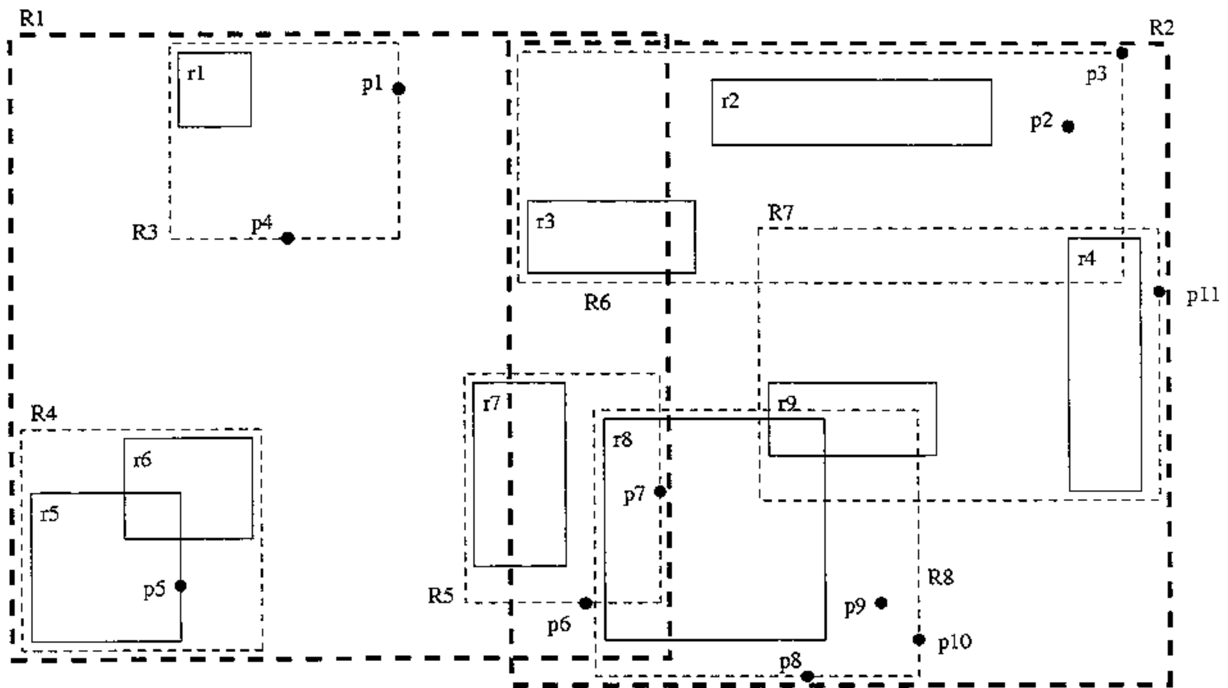
Uma entrada referente a um novo objeto é inserida sempre nas folhas da estrutura. A escolha da folha onde ela deve ser armazenada é feita percorrendo-se a árvore a partir da raiz, seguindo, em cada nível, o ponteiro da entrada cujo MBR necessitar do menor aumento de área para conter o MBR do novo elemento. Empates são resolvidos escolhendo-se a entrada cujo MBR tiver a menor área. Este procedimento é repetido até que se atinja uma folha.

Se a folha escolhida ainda não está saturada, a nova entrada é inserida e, se necessário, os MBRs das entradas acessadas nos nós superiores durante o percurso até aquela folha são atualizados para incluir a área do MBR do novo objeto. Caso a folha $i$ que deve receber a nova entrada já tenha atingido sua capacidade máxima, ela deve sofrer uma operação de *split*, isto é, uma nova folha $j$ é criada, e as entradas de $i$ mais a entrada a ser inserida são divididas entre $i$ e $j$. O *split* implica na atualização do MBR da entrada referente a $i$ em seu nó pai, assim como na criação, nesse nó, de uma nova entrada referente a $j$. Se o nó pai de $i$ estiver com sua capacidade de armazenamento esgotada, ele também sofrerá uma divisão. Assim, o *split* será propagado para os nós ancestrais de $i$ que estiverem também completamente ocupados, até que se encontre um que não esteja. Desse nível para cima, apenas ajustes nos MBRs das entradas pertencentes ao caminho da raiz até $i$ serão feitos, se for preciso, para incluir o MBR do objeto inserido. Se for necessário se fazer um *split* da raiz, um nó é criado para ser a nova raiz, passando a conter duas entradas, uma para cada nó resultante da divisão da antiga raiz.

A rotina de *split* da R-tree busca dividir as entradas de tal forma que as áreas dos MBRs que envolvem cada um dos dois grupos formados sejam as menores possíveis. Assim como na rotina que escolhe o nó folha onde será feita uma inclusão, o objetivo de se tentar minimizar as áreas dos MBRs que envolvem as entradas de cada nó é reduzir o número de nós visitados em uma consulta, partindo do princípio de que quanto menor for um MBR, menor será a chance de que ele intersecte uma janela de consulta. Além disso, a minimização da área dos retângulos diminui o *dead space*, responsável pela detecção de




(a)




(b)

Figura 2.41: Uma R-tree (a) e os retângulos e pontos indexados (b).

falsas intersecções e, portanto, pelo aumento do custo das operações de consulta. Guttman apresenta três algoritmos para a execução da rotina de *split*:

- um exaustivo, que garantidamente encontra a melhor divisão, mas que tem tempo de execução exponencial no número de entradas por nó, sendo, portanto, proibitivo para valores práticos de $M$;
- um quadrático, que escolhe, entre as $M+1$ entradas a serem redistribuídas, as duas que resultariam no MBR com maior área caso fossem armazenadas no mesmo nó, e as separa como *sementes* de dois grupos. Das entradas restantes, uma é escolhida a cada vez e inserida no grupo em que ela causa menor aumento de área, e no caso de empate, no grupo com menor área. A entrada escolhida é aquela que apresenta a maior diferença entre o aumento de área que causaria sua inserção no primeiro grupo e o aumento de área que causaria sua inserção no segundo. Se um grupo atinge $M+1-m$ entradas, todas as entradas restantes são incluídas no grupo com menor número de elementos. Cada grupo é, então, armazenado em um nó distinto.
- um linear, que escolhe como sementes os dois retângulos que apresentem a maior distância entre si. Em cada dimensão, são considerados o retângulo com menor coordenada máxima e o retângulo com maior coordenada mínima, e a separação entre eles é normalizada dividindo-se a distância entre essas coordenadas pela extensão total do espaço naquela dimensão. Os dois retângulos com maior distância normalizada em qualquer das dimensões são os eleitos. As entradas subseqüentes são tomadas em qualquer ordem, e inseridas no grupo em que causarem o menor aumento de área. Como no caso anterior, empates são resolvidos inserindo-se a entrada em questão no grupo de menor área.

Segundo Guttman, o algoritmo linear é tão bom quanto os outros dois, por ser mais rápido e não afetar consideravelmente o desempenho das consultas. No entanto, em [BKSS90] os autores relataram um desempenho bem melhor do algoritmo quadrático nos experimentos que desenvolveram.

Em [Gre89], Greene apresenta uma implementação que usa outro algoritmo de *split* quadrático. Após a escolha das sementes dos dois grupos, o que é feito da mesma forma que no algoritmo quadrático de Guttman, determina-se a dimensão a ser particionada. Esta será a que apresentar a maior distância normalizada entre os MBRs usados como sementes. As entradas são, então, classificadas pela menor coordenada de seus MBRs nessa dimensão. As primeiras $(M+1)$ *div* 2 são alocadas a um grupo, e as demais ao outro.

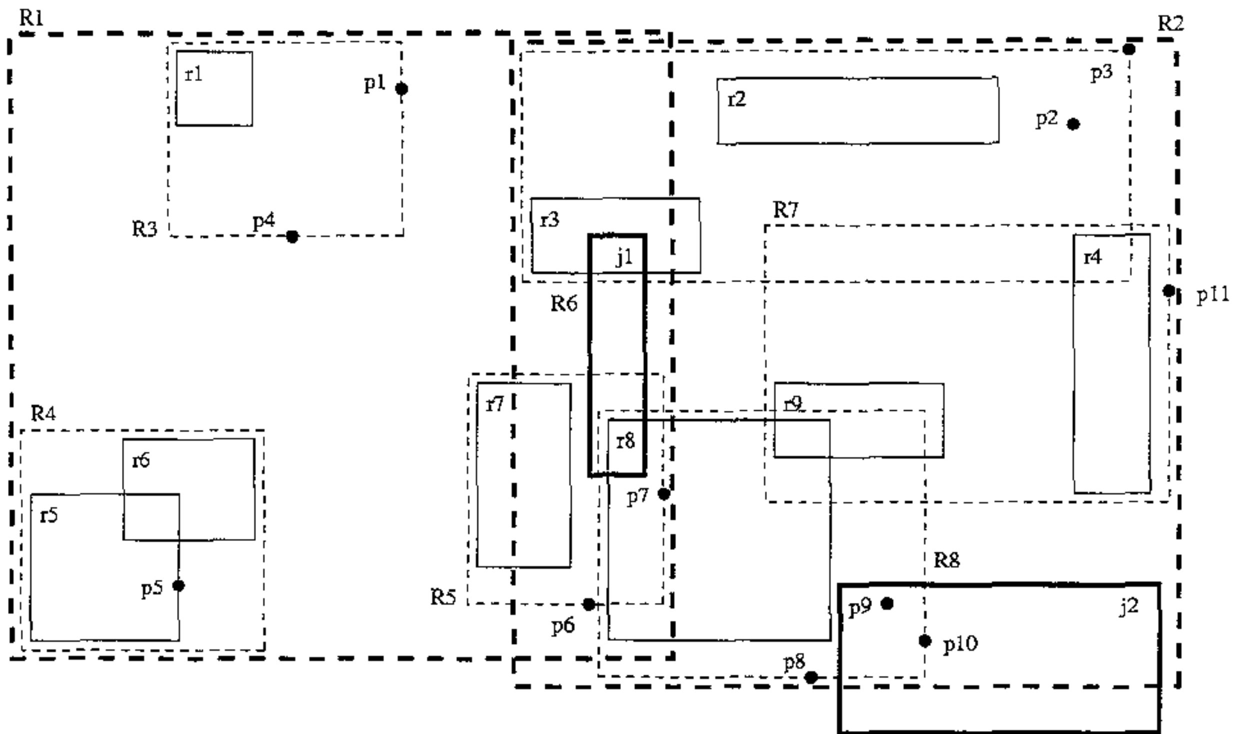
### Consulta

As consultas na R-tree são feitas através da avaliação de predicados sobre a janela de consulta e os retângulos armazenados nas entradas dos nós. Para se determinar os objetos que intersectam uma janela de consulta, a árvore é percorrida a partir da raiz, e cada nó visitado tem suas entradas verificadas. Se o nó é não-folha, as entradas cujos MBRs intersectam a janela de consulta indicam os nós filhos que deverão também ser visitados. Se o nó corrente é uma folha, as entradas cujos MBRs intersectam a janela de consulta contêm os identificadores dos objetos procurados, e são reportadas como parte do conjunto resposta. A figura 2.42a apresenta duas janelas de consulta aplicadas sobre o conjunto de retângulos e pontos da figura 2.41b. A figura 2.42b mostra os caminhos percorridos na R-tree para se chegar aos retângulos que intersectam a janela $j_1$. Note que apesar do nó correspondente ao retângulo $R_5$ ser visitado, nenhuma de suas entradas faz parte da resposta. O motivo é que a intersecção entre $R_5$ e $j_1$ ocorre numa região de *dead space*.

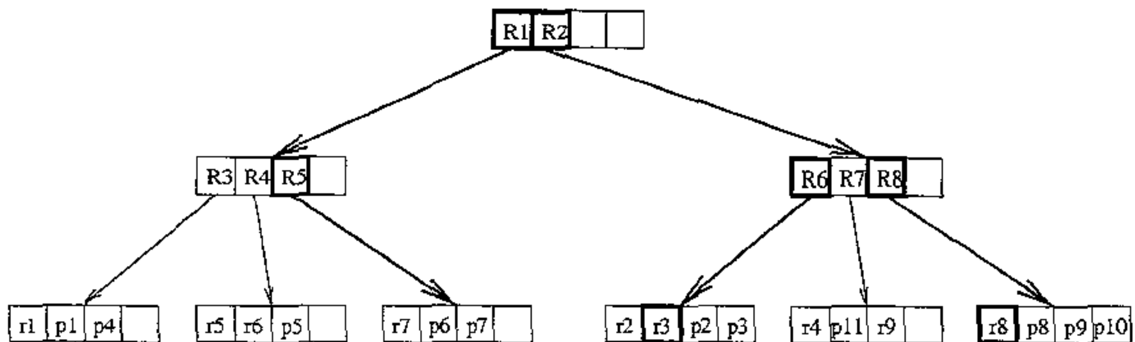
A determinação dos retângulos que incluem uma janela de consulta segue um processo semelhante. A diferença é que são consideradas apenas as entradas cujos MBRs se sobrepõem totalmente à janela. As *point queries* são um caso especial de *range queries* para determinação de inclusão onde as janelas têm área zero, e portanto o mesmo algoritmo é usado.

A consulta para identificação dos objetos contidos em uma janela é, quando o nó visitado é não-folha, idêntica à de determinação de intersecção, ou seja, as entradas consideradas são aquelas cujos MBRs *intersectam* a janela de consulta, e não somente aquelas cujos MBRs nela estão contidos. Somente nas folhas as entradas reportadas são as que realmente têm seus MBRs inclusos na janela. Isto se deve ao fato de que se o espaço representado por uma entrada intersecta uma janela de consulta existe a possibilidade de que o retângulo de alguma das entradas do nó filho que ela referencia esteja contido nessa intersecção. Como exemplo, veja a janela $j_2$ na figura 2.42a. Apesar dos retângulos $R_2$ e $R_8$ apenas a intersectarem, $j_2$ contém os pontos $p_9$ e $p_{10}$, cujas entradas estão armazenadas em sub-árvores representadas por esses retângulos.

Para se recuperar a entrada correspondente a um determinado objeto, a estrutura é pesquisada seguindo-se as entradas cujos retângulos incluem seu MBR. Embora o objeto esteja armazenado em uma única folha, pode ser que várias sub-árvores tenham que ser percorridas, devido às sobreposições entre os retângulos de cada nível. Ao se atingir uma folha, uma comparação entre os identificadores das entradas e o do objeto deve ser feita, pois objetos distintos podem ter MBRs de coordenadas idênticas.




(a)




(b)

Figura 2.42: Duas *range queries* sobre um conjunto de retângulos e pontos (a) e os caminhos percorridos na R-tree para identificação dos retângulos que intersectam $j_1$ (b).

**Exclusão**

A exclusão de um retângulo da R-tree começa pela identificação do nó folha que o contém, como descrito nos procedimentos de consulta. Após a remoção, se a ocupação do nó ainda estiver acima da ocupação mínima exigida, os MBRs das entradas que definem o caminho entre esse nó e a raiz devem ser verificados e, se necessário, ajustados.

Se a remoção retângulo faz com que a ocupação do nó caia abaixo do número mínimo de entradas permitido $m$ (evento conhecido como *underflow*), esse nó é retirado da árvore, e as entradas que lhe restavam são colocadas em um conjunto $Q$ para serem reinseridas ao final do processo. A remoção do nó folha implica na exclusão da entrada correspondente no seu nó pai, o que pode acarretar um *underflow* também nesse nó, fazendo com que ele seja excluído e suas entradas adicionadas ao conjunto $Q$. Assim, o *underflow* pode se propagar até a raiz. O *underflow* da raiz só ocorre se lhe resta apenas um filho. Nesse caso ela é excluída, e seu filho passa a ser a raiz da árvore. Uma vez atingido um nível para o qual o *underflow* não se propagou, é feita a verificação e, se necessário, o ajuste dos MBRs no caminho até a raiz.

A reinserção das entradas pertencentes ao conjunto $Q$ se inicia pelas entradas que foram adicionadas a ele por último, pois elas podem se referir a nós dos níveis superiores e, portanto, poderão servir para guiar a reinclusão de entradas dos nós de níveis inferiores. Cada entrada é reinserida no mesmo nível em que ela se encontrava anteriormente, o que garante que, ao final, todas as folhas continuarão no mesmo nível.

## 2.7.2 $R^+$-tree

A fim de evitar a intersecção apresentada na R-tree entre os retângulos das entradas dos nós não-folha, Sellis et al [SRF87] desenvolveram a $R^+$-tree, um método de acesso que usa a técnica da divisão do espaço nativo sem sobreposição[11]. A estratégia utilizada é permitir que o MBR de um objeto seja decomposto em uma coleção de sub-retângulos disjuntos cuja união forme o MBR original sempre que isso seja necessário para que os retângulos das entradas que o contêm nos níveis superiores não se sobreponham.

A estrutura da $R^+$-tree é exatamente a mesma da R-tree, mas as regras para sua construção são bastante diferentes:

- Para cada entrada *(ponteiro, MBR)* de um nó não-folha, a sub-árvore apontada por *ponteiro* contém um retângulo $R$ se, e somente se, $R$ é totalmente sobreposto por $MBR$, a menos que $R$ seja um retângulo de uma entrada de um nó folha (ou seja, o retângulo de um objeto), caso em que a intersecção entre $R$ e $MBR$ é suficiente. Isso significa que os retângulos das entradas dos nós intermediários podem ser divididos

[11] Veja a seção 2.3.1.

se necessário, enquanto que os retângulos dos objetos são duplicados e armazenados em todas as folhas cujo espaço eles intersectam.

- Para quaisquer duas entradas *(ponteiro*$_1$*, MBR*$_1$*)* e *(ponteiro*$_2$*, MBR*$_2$*)* de um nó intermediário, a intersecção entre $MBR_1$ e $MBR_2$ é nula.

- A raiz tem pelo menos dois filhos, a não ser que ela seja uma folha. Não há uma ocupação mínima exigida para os outros nós.

- Todas as folhas estão no mesmo nível.

A figura 2.43 mostra um conjunto de retângulos e pontos indexados por uma $R^+$-tree cuja capacidade máxima de armazenamento é de quatro entradas por nó. O retângulo $r_3$ foi representado nas folhas correspondentes a $R_4$ e $R_7$, e $r_8$ nas folhas correspondentes a $R_6$, $R_9$ e $R_{10}$ para permitir que os retângulos dos níveis superiores fossem disjuntos.

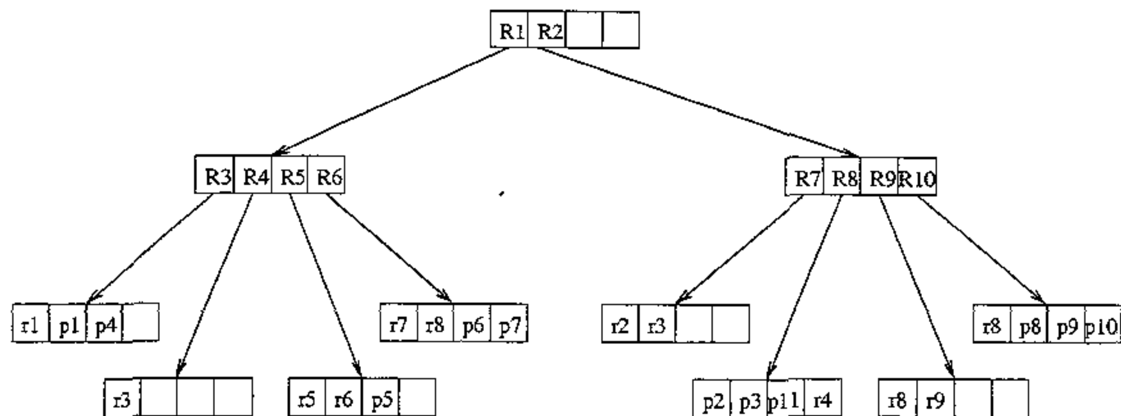
**Inserção**

Para se inserir um retângulo na $R^+$-tree, a estrutura é percorrida a partir da raiz e, para cada nó não-folha visitado, todas as entradas cujos MBRs intersectam o novo retângulo têm suas sub-árvores percorridas. O novo retângulo é, então, armazenado em todas as folhas atingidas. Existem três casos de inserção que necessitam de maior cuidado e, no entanto, não foram mencionadas em [SRF87]. O primeiro é quando um retângulo é inserido em um nó onde ele não intersecta o MBR de nenhuma das entradas (veja a figura 2.44a). O segundo é quando o MBR do novo objeto intersecta apenas parcialmente algumas das entradas (figura 2.44b). O terceiro acontece quando não há como expandir os MBRs das entradas do nó para conterem o novo retângulo sem que eles se intersectem (figura 2.44c).

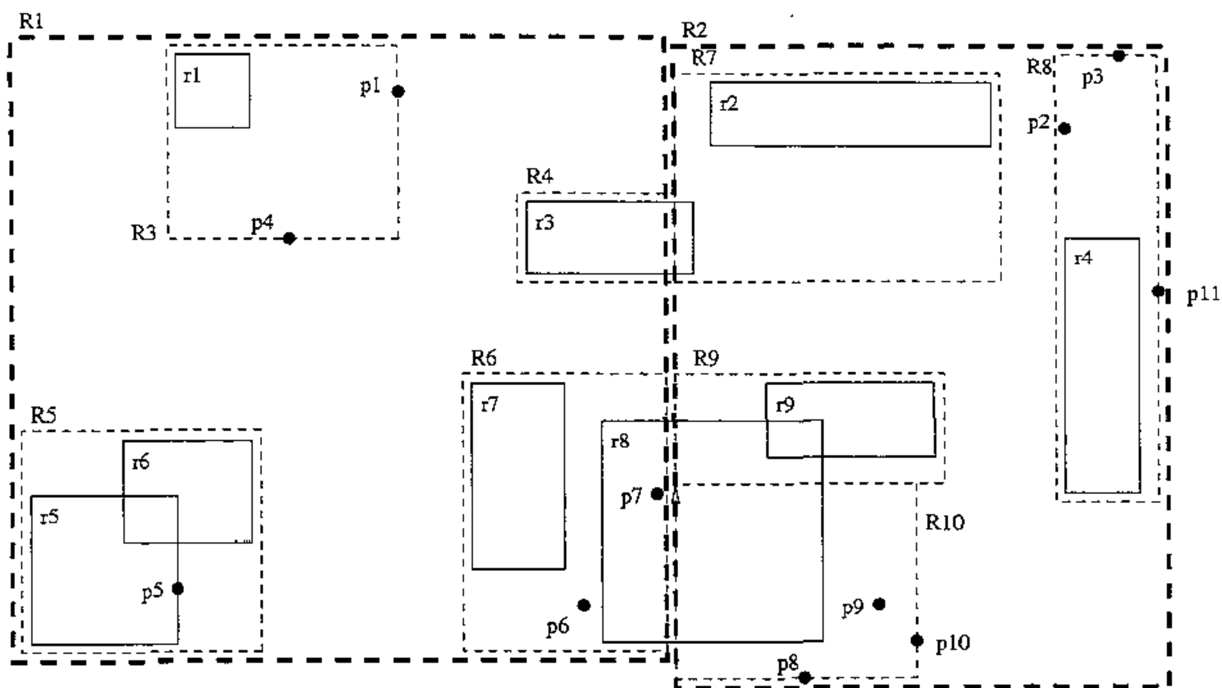
Para resolver estes problemas, em sua implementação Cox [Cox91] adotou uma solução que divide entre as entradas dos nós intermediários de cada nível o espaço total onde os dados estão dispostos. Assim, em qualquer nível não há espaços vagos, o que previne as situações citadas. Para isso um novo retângulo, chamado *MaxRect*, é acrescentado a cada entrada. Esse retângulo descreve o espaço sobre o qual cada entrada é responsável, e tem as seguintes propriedades:

- todos os *MaxRects* das entradas de um determinado nó intermediário dividem completamente e sem sobreposição o espaço representado pelo *MaxRect* da entrada correspondente a esse nó em seu nó pai;

- caso o nó seja a raiz, a união dos *MaxRects* de suas entradas representam o espaço total onde os dados estão dispostos.




(a)




(b)

Figura 2.43: Uma $R^+$-tree (a) e os retângulos e pontos indexados (b).

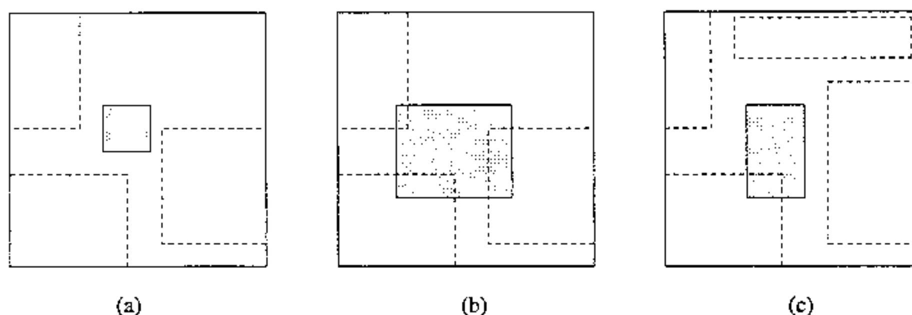



Figura 2.44: Três situações especiais na inserção de um retângulo. Os retângulos tracejados representam os MBRs das entradas de um nó.

Os *MaxRects* são utilizados para direcionar o percurso durante a inserção, e os MBRs das entradas para direcionar as consultas. Um problema desta abordagem é que cada nó só consegue armazenar cerca de metade das entradas que os nós de mesmo tamanho na R-tree comportam. A figura 2.45 mostra um conjunto de entradas e seus *MaxRects*.

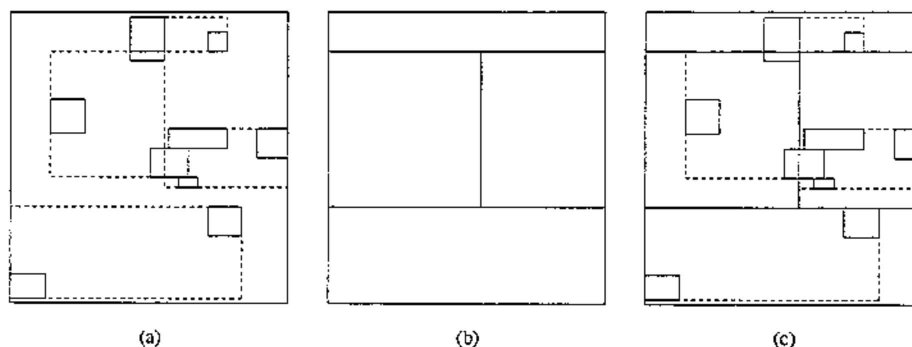



Figura 2.45: Um conjunto de entradas de um nó intermediário (retângulos tracejados) (a) e a divisão do espaço através de MaxRects (b) e (c).

Se a inserção de um retângulo causa um *overflow*, o espaço coberto pelo nó onde ele foi inserido é dividido por uma linha paralela a um dos eixos, e os retângulos que se situam de cada um dos lados dessa linha são associados a dois nós distintos. Se o nó não é uma folha, os retângulos que intersectam a linha são particionados, e cada sub-retângulo é associado ao nó apropriado. Se o nó que sofre o *split* é uma folha, os retângulos que intersectam a linha são representados nos dois nós resultantes, sem particionamento. Como na R-tree, o *split* pode se propagar para os níveis superiores, mas essa propagação pode ser necessária também para os inferiores. Isso ocorre se a linha divisória aplicada aos retângulos de um determinado nível intersectar algum retângulo nos níveis inferiores. A figura 2.46 exemplifica essa situação. O retângulo $A$ é pai de $B$, que por sua vez é pai de $C$. Se $A$ é particionado, dando origem a $A_1$ e $A_2$, $B$ e $C$ também têm que ser divididos. Caso contrário, $B$ e $C$ estariam representados em $A_1$ ou em $A_2$, o que viria a contrariar a

propriedade de que a sub-árvore de uma entrada só contêm um retângulo se seu MBR o sobrepõe completamente.

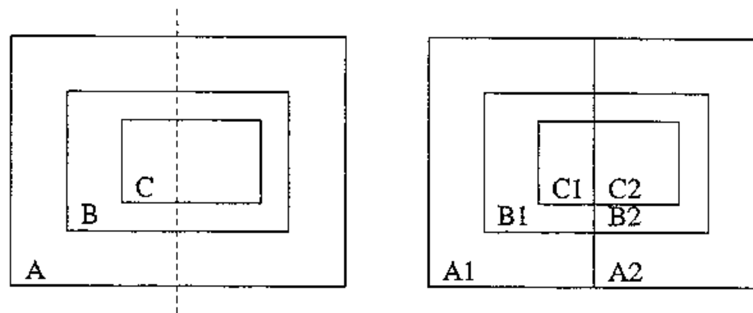



Figura 2.46: Propagação de um split para os níveis inferiores em um nó de uma $R^+$-tree.

Sellis et al propõem um algoritmo de *split* que classifica os retângulos das entradas uma vez em cada eixo e usa um parâmetro — *ff (fill factor)* — que indica o número de retângulos a serem armazenados no primeiro nó resultante. Os autores ainda sugerem que a escolha do eixo de partição seja baseada em um ou mais dos seguintes critérios:

- proximidade entre os retângulos de cada grupo;
- comprimento mínimo total nos eixos $x$ e $y$;
- área total mínima das duas sub-regiões;
- número mínimo de retângulos particionados.

Os três primeiros critérios procuram diminuir a área de *dead-space*, enquanto o último procura limitar a altura da árvore. Nem sempre os quatro podem ser satisfeitos ao mesmo tempo.

A $R^+$-tree sofre o mesmo problema que outros métodos de acesso que fazem a divisão do espaço nativo sem sobreposição: ela não suporta conjuntos de dados cujas densidades sejam maiores que a capacidade de armazenamento dos nós. Isto pode ser contornado usando um encadeamento de nós para conter os retângulos que se sobrepõem em uma área muito densa, mas a árvore ficaria desbalanceada.

### Consulta

O algoritmo de consulta para determinação de intersecção percorre a árvore a partir da raiz, seguindo pelas sub-árvores apontadas pelas entradas cujos retângulos intersectam a janela de consulta. Ao atingir as folhas, as entradas cujos retângulos intersectam a

janela são reportadas como parte da resposta (é necessário se eliminar os identificadores duplicados, já que um mesmo objeto pode estar representado em mais de uma folha).

O algoritmo para identificação dos retângulos que contêm a janela de consulta se comporta da mesma forma se o nó corrente é não-folha. Apenas nas folhas o predicado "contém" é verificado em lugar de "intersecta".

**Exclusão**

Para que um retângulo seja excluído da $R^+$-tree, ele deve ser removido de todas as folhas onde está armazenado. Essas folhas são identificadas pelo algoritmo de consulta descrito no tópico anterior. Se necessário, os MBRs dos níveis superiores que continham o retângulo excluído são ajustados.

Uma vez que não há exigência de ocupação mínima dos nós, o *underflow* só ocorre quando um nó fica vazio. Neste caso, ele se propagará para os níveis imediatamente superiores se os nós desses níveis tiverem somente um filho.

### 2.7.3 R*-tree

A R*-tree, proposta por Beckmann et al [BKSS90], é uma tentativa de melhorar o agrupamento de retângulos feito pela R-tree. A estrutura das duas árvores é exatamente a mesma, assim como são os mesmos os algoritmos de consulta e exclusão. O que as diferencia são os algoritmos de inserção, pois enquanto os da R-tree buscam reduzir apenas a área dos MBRs das entradas dos nós não-folha, os da R*-tree consideram, além deste, outros três fatores:

- a *sobreposição* entre os retângulos das entradas dos nós não-folha, cuja diminuição tende a melhorar o desempenho da estrutura, pois reduz o número de sub-árvores a serem percorridas em uma consulta;
- a *ocupação* dos nós da estrutura, cujo aumento tende a diminuir o custo das consultas devido à redução da altura da árvore;
- o *perímetro* dos retângulos. Ao se reduzir o valor desse parâmetro procura-se, tanto quanto possível, obter retângulos de formatos próximos a quadrados, pois o retângulo com menor perímetro para um valor fixo de área é um quadrado. Como quadrados podem ser agrupados mais facilmente, o agrupamento dos MBRs de um determinado nível resultariam em MBRs menores no nível superior.

Os procedimentos para melhoria da estrutura com base nos fatores citados estão concentrados em duas rotinas: a que escolhe a folha onde um retângulo deve ser armazenado e a que trata a ocorrência de *overflow* na inserção de uma entrada.

### Percurso na inserção

A folha onde se armazenará a entrada referente a um objeto é escolhida percorrendo-se a árvore a partir da raiz. Se as entradas do nó corrente apontam para nós não-folha, o percurso continuará pela sub-árvore correspondente à entrada cujo MBR necessitar de menor aumento de área para conter o novo objeto. Empates são resolvidos escolhendo-se a entrada cujo MBR tiver a menor área. Até esse ponto o algoritmo é exatamente igual ao da R-tree. Se as entradas do nó corrente apontam para folhas, a folha escolhida será aquela correspondente à entrada cujo MBR necessitar de menor aumento da somatória de suas áreas de sobreposição com os MBRs das outras entradas do nó. Resumindo, esse algoritmo procura minimizar a área dos MBRs das entradas que apontam para os nós não-folha e a sobreposição entre os MBRs das que apontam para os nós folha.

### Tratamento de overflow

Tanto a R-tree como a R*-tree são afetadas pela ordem de inserção dos dados, o que significa que uma árvore pode se comportar de formas distintas para o mesmo conjunto de dados se estes forem armazenados em seqüências diferentes. Os retângulos inseridos no início da construção podem gerar MBRs nos níveis superiores que levarão a uma queda de desempenho à medida que novos retângulos são armazenados. Para permitir a redistribuição das entradas antigas na árvore, Beckmann propôs que se utilizasse a reinserção de parte das entradas de um nó que sofresse um *overflow*, de forma semelhante à reinserção feita no algoritmo de exclusão para tratamento de nós que sofrem *underflow*.

O algoritmo de reinserção classifica as entradas do nó em ordem decrescente da distância entre o centróide do retângulo de cada uma e o centróide do retângulo que envolve todas as entradas. Após a classificação, as $p$ primeiras entradas são reinseridas, e as restantes continuam no nó que havia sofrido *overflow*. Segundo Beckmann, o melhor desempenho é obtido para $p$ igual a 30% de $M$. A reinserção dos $p$ primeiros retângulos pode ser feita de duas formas: do mais distante para o menos distante do centro (*far reinsert*), ou do menos distante para o mais distante (*close reinsert*). Nos experimentos de Beckmann et al o *close reinsert* resultou em melhor desempenho que o *far reinsert*, mas os resultados dos nossos testes (seção 4.4) apresentaram a R*-tree que utiliza o *far reinsert* como a melhor opção para a indexação dos dados utilizados, considerando as condições de operação da aplicação de onde eles foram extraídos (seção 3.2).

Como é possível que as entradas venham a ser reinseridas no mesmo nó em que estavam, este pode sofrer novo *overflow*. O mesmo pode acontecer com um ou mais nós do mesmo nível que venham a receber as entradas. Para evitar que a rotina caia num *loop*, a reinserção só é usada na primeira vez em que ocorre um *overflow* em um determinado nível. Da segunda vez é usada a rotina de *split*.

Cox [Cox91] usa em seus experimentos um algoritmo de reinserção diferente do de Beckmann no que se refere à classificação das entradas. Para cada dimensão, ele as ordena de acordo com a distância *naquela dimensão* entre o centro do MBR de cada uma e o centro do retângulo que envolve as entradas do nó. Depois ele escolhe a classificação cuja retirada dos $p$ retângulos mais distantes resultar em menor área.

O uso da rotina de reinserção no tratamento de *overflow* além de permitir uma reorganização da árvore, induz a uma melhor ocupação dos nós nos casos em que consegue evitar o *split*, pois além de garantir uma ocupação de 70% no nó que sofreu *overflow*, possibilita um aumento da ocupação de seus vizinhos.

O algoritmo de *split* da R*-tree classifica as entradas do nó duas vezes em cada dimensão: uma pela menor coordenada de cada MBR e outra pela maior. Para cada classificação, considera-se $M - 2m + 2$ distribuições das $M + 1$ entradas em dois grupos, onde na $k$-ésima distribuição ($k = 1, \ldots, (M - 2m + 2)$) o primeiro grupo contém as primeiras $m - 1 + k$ entradas e o outro contém as entradas restantes. Para cada distribuição são calculadas:

- a área da distribuição, que é a soma das áreas dos dois retângulos que envolvem os grupos resultantes;
- o perímetro da distribuição, que é a soma dos perímetros dos dois retângulos que envolvem os grupos;
- a área de intersecção da distribuição, que é igual à área de intersecção entre os dois retângulos.

O eixo onde será feita a partição será aquele que apresentar a menor somatória dos perímetros de todas as suas distribuições. Uma vez escolhido o eixo, o próximo passo é a determinação da distribuição entre os dois grupos. A escolhida será a distribuição, naquele eixo, que tiver a menor área de intersecção entre os grupos. Empates são resolvidos escolhendo-se a distribuição com menor área.

### 2.7.4 Hilbert R-tree

A Hilbert R-tree foi proposta por Kamel e Faloutsos em [KF94]. A idéia principal deste método de acesso é criar uma ordenação dos retângulos de dados [12] através de uma curva de Hilbert, de forma que cada nó da estrutura tenha um conjunto bem definido de vizinhos. Assim, numa vizinhança com $s$ nós, ao invés de se fazer o *split* de um nó tão logo sua capacidade seja ultrapassada, tenta-se repassar algumas de suas entradas para seus $s - 1$ vizinhos. Somente se todos eles já estiverem cheios executa-se o *split*, criando-se um novo

[12] Os retângulos que representam os objetos, e que são armazenados nas folhas.

nó e dividindo-se as entradas dos $s$ vizinhos entre os $s + 1$ nós resultantes. Ou seja, a Hilbert R-tree adia o *split* de um nó até que sua capacidade mais a de seus $s - 1$ vizinhos seja ultrapassada. Kamel e Faloutsos chamaram esse procedimento de "política de *split* $s$ para $s + 1$" (a política de *split* da R-tree, que divide um nó em dois, é chamada pelos autores de "política 1 para 2"). Ajustando-se convenientemente o valor de $s$, pode-se fazer com que a estrutura atinja uma ocupação tão próxima de 100% quanto se queira.

A estrutura da Hilbert R-tree é semelhante à da R-tree:

- cada nó folha contém no máximo $C_f$ entradas da forma $(identificador, MBR)$, onde $C_f$ é a capacidade de uma folha, $identificador$ é um ponteiro para a descrição completa de um objeto e $MBR$ é o seu retângulo envolvente mínimo;

- cada nó não-folha contém no máximo $C_n$ entradas da forma $(ponteiro, MBR, LHV)$, onde $C_n$ é a capacidade de um nó não folha, $ponteiro$ contém o endereço de um nó filho, $MBR$ é o retângulo envolvente mínimo dos retângulos das entradas desse nó filho e LHV *(largest Hilbert value)* é a maior coordenada linear na curva de Hilbert entre os retângulos *de dados* nas sub-árvores do nó (as coordenadas lineares dos retângulos das entradas dos nós não-folha nunca são calculadas, apenas as dos retângulos de dados). A coordenada linear de um retângulo é definida como sendo a coordenada linear de seu centro.

- apesar de Kamel e Faloutsos não determinarem um valor mínimo para a ocupação dos nós, fica clara no texto a sua existência.

A figura 2.47 mostra um conjunto de retângulos indexados por uma Hilbert R-tree. As coordenadas lineares dos retângulos de dados do nó $B$ estão em itálico, entre colchetes. Os LHVs dos nós estão também entre colchetes, em negrito. A figura 2.48 apresenta o conteúdo da raiz e de uma das folhas (nó $B$). Todos os retângulos no nó $A$ tem coordenada linear menor ou igual a 33; no nó $B$ as coordenadas lineares são maiores que 33 e menores ou iguais a 107, e assim por diante.

### Inserção

Para se inserir um novo retângulo em uma Hilbert R-tree, a coordenada linear de seu centro é usada como chave. O algoritmo percorre a árvore a partir da raiz e, a cada nó visitado, a entrada que indica o próximo nó do percurso é a que tiver o menor LHV que seja maior ou igual à coordenada linear do retângulo a ser inserido. O algoritmo apresentado em [KF94] não trata o caso em que a coordenada linear do novo retângulo é maior que o maior LHV da árvore, mas nesse caso, obviamente, o percurso deve seguir pela entrada com o maior LHV.

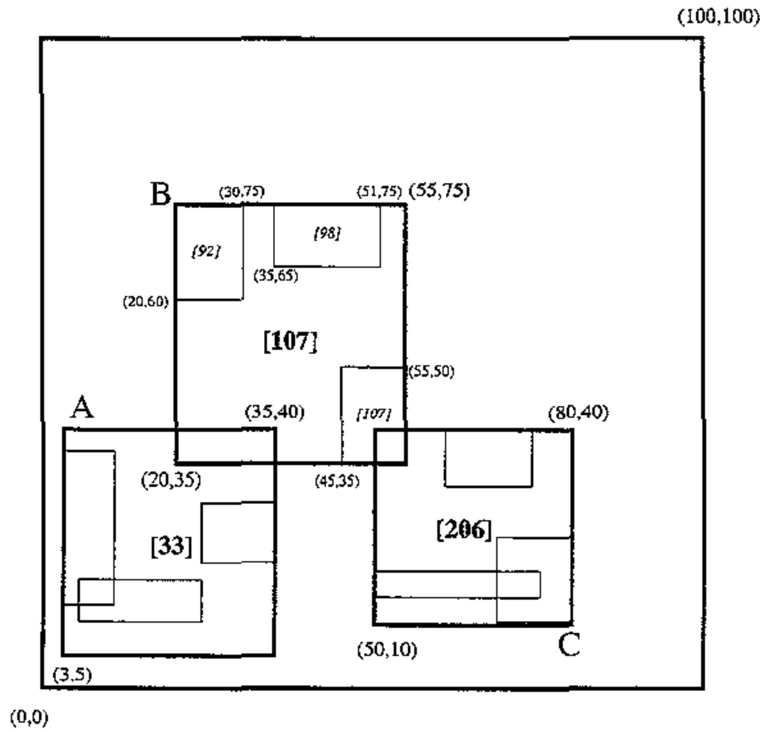



Figura 2.47: Um conjunto de retângulos indexados por uma Hilbert R-tree.

Se a folha atingida dispõe de espaço, o retângulo é armazenado na posição apropriada conforme sua coordenada linear, pois as entradas são ordenadas por esse valor. Se a folha já estiver cheia, o algoritmo verifica se há espaço em algum dos seus $s - 1$ vizinhos. Se afirmativo, as entradas dos $s$ vizinhos mais a nova entrada são distribuídas entre todos de acordo com suas coordenadas lineares. Caso contrário, um novo nó é criado, e as entradas são distribuídas entre os $s + 1$ nós. Como na R-tree, o *split* pode se propagar para os níveis superiores. Em qualquer caso, as entradas no caminho entre a folha e a raiz devem ser verificadas e, se necessário, seus MBRs e LHVs devem ser ajustados. Caso ocorra um *split* da raiz, um nova raiz é criada, e os dois nós resultantes do *split* passam a ser seus filhos.

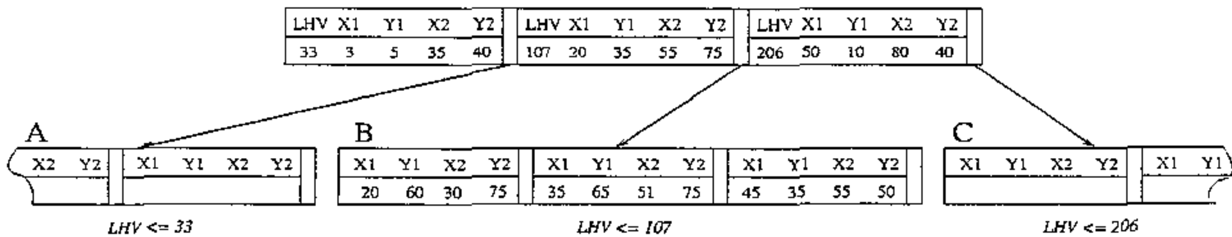



Figura 2.48: Estrutura correspondente aos retângulos da figura anterior.

Embora a ocupação dos nós da árvore cresça à medida que se utiliza valores maiores de $s$, o custo da inserção em termos do número de nós lidos também aumenta, pois a cada vez que se encontra um nó cheio é necessário se recuperar também seus $s - 1$ vizinhos para que a redistribuição das entradas seja feita. Kamel e Faloutsos apontam a política de *split* 2 para 3 como sendo a que apresenta o melhor compromisso entre o desempenho das consultas e o custo das inserções.

**Consulta**

As rotinas de consulta da Hilbert R-tree são exatamente iguais as da R-tree.

**Exclusão**

O algoritmo de exclusão da Hilbert R-tree, ao invés de reinserir as entradas de um nó que sofre *underflow*, procura repassar para ele algumas entradas de seus $s$ vizinhos (para a exclusão são necessários $s$ vizinhos, enquanto que na inserção são $s - 1$). Se todos eles já estiverem com a ocupação mínima, o nó em questão e seus $s$ vizinhos são condensados em $s$ nós. O *underflow* pode se propagar para os níveis superiores da estrutura. Se ele chegar até a raiz, ela é excluída, e seu filho restante passa a ser a nova raiz.

Como na inclusão, as entradas do caminho entre a folha onde ocorreu a remoção e a raiz são verificadas e, se necessário, seus MBRs e LHVs são atualizados.

### 2.7.5 R-trees compactadas

Quando o conjunto de dados a ser indexado é conhecido previamente, é possível se construir uma R-tree com ocupação de quase 100%, diminuindo o custo das consultas através da redução do número de nós a serem acessados. As árvores compactadas são úteis quando os dados são estáticos, isto é, quando não sofrem atualizações, ou ainda quando elas são poucas e não freqüentes.

O modo de operação de todos os algoritmos de compactação é o mesmo, e tem os seguintes passos:

- O arquivo de dados é pré-processado de forma que os $r$ retângulos nele contidos sejam ordenados segundo algum valor e depois divididos em $\lceil r/M \rceil$ grupos consecutivos de $M$ retângulos, onde $M$ é a capacidade máxima de armazenamento de um nó. O último grupo pode conter menos de $M$ retângulos, o que ocorre se $r/M$ não é um número inteiro.

- Cada grupo de $M$ retângulos é armazenado em uma folha, e para cada folha criada armazena-se em uma área temporária uma entrada *(número-do-nó, MBR)*. Essas entradas serão usadas na construção dos nós do nível imediatamente superior.

- Os MBRs das entradas armazenadas na área temporária são, da mesma forma, distribuídos em grupos de $M$ retângulos (o último pode ter menos que $M$), e cada grupo é armazenado em um nó. Para cada nó é inserida uma entrada na área temporária, e o algoritmo segue construindo recursivamente os níveis superiores, até que a raiz seja criada.

Três algoritmos de compactação foram propostos, e a diferença entre eles está na forma como os retângulos de cada nível são ordenados. O primeiro algoritmo, desenvolvido por Roussopoulos e Leifker [RL85], classifica os retângulos de acordo com suas coordenadas no eixo $x$. Em [KF93], Kamel e Faloutsos propõem o uso de uma curva de Hilbert para classificar os retângulos de acordo com as coordenadas lineares de seus pontos centrais. A vantagem desse algoritmo sobre o primeiro reside na sua capacidade de agrupar melhor os retângulos considerando a proximidade em todas as dimensões.

O algoritmo mais recente foi apresentado por Leutenegger, Lopez e Edgington [LLE97], o *Sort-Tile-Recursive* (STR). Considerando um conjunto de $r$ retângulos dispostos num espaço bidimensional, a idéia básica é dividi-lo em $\sqrt{r/M}$ fatias verticais de forma que cada fatia contenha retângulos suficientes para preencher aproximadamente $\sqrt{r/M}$ nós. Inicialmente o algoritmo calcula o número de páginas $P = \lceil r/M \rceil$ que a árvore terá em seu nível mais baixo, ou seja, seu número de folhas. Em seguida os retângulos são classificados pelas coordenadas de seus centros no eixo $x$ e divididos em $\sqrt{P}$ grupos, que são as fatias verticais. Cada fatia consiste de um conjunto de $\sqrt{P} \times M$ retângulos consecutivos da lista ordenada, podendo a última conter um número menor. Finalmente, os retângulos pertencentes a cada fatia são classificados pelas coordenadas de seus centros no eixo $y$, e cada subgrupo com $M$ retângulos consecutivos é armazenado em um nó. O algoritmo constrói recursivamente os nós dos níveis superiores ordenando e dividindo da mesma forma os retângulos que envolvem as entradas do nível imediatamente inferior.

Generalizando para espaços $k$-dimensionais, o algoritmo STR primeiro classifica os hiper-retângulos de acordo com a coordenada de seus centros na primeira dimensão. Depois ele os divide em $\lceil P^{\frac{1}{k}} \rceil$ fatias, onde cada fatia consiste de um conjunto de $M \times \lceil P^{\frac{k-1}{k}} \rceil$ hiper-retângulos consecutivos da lista ordenada. Cada fatia é, então, processada recursivamente usando as $k-1$ coordenadas restantes, ou seja, é tratada como se fosse um conjunto de dados $(k-1)$-dimensional.

Segundo os resultados dos experimentos apresentados em [LLE97], o algoritmo que classifica os retângulos pelo valor de suas coordenadas no eixo $x$ só consegue concorrer com os outros dois na execução de *point queries* sobre conjuntos de pontos. Além disso, a classificação através do STR obteve melhor desempenho que a ordenação pela curva de Hilbert para diferentes conjuntos de dados, exceto naqueles em que a distribuição era muito irregular. Nesses casos, o algoritmo de Kamel e Faloutsos atingiu melhores

resultados onde os dados eram retângulos, e a liderança se alternou para a consulta a pontos, dependendo da área da janela e do tamanho do *buffer*.

### 2.7.6 Outras pesquisas sobre as R-trees

Alguns pesquisadores tem tentado desenvolver algoritmos para aumentar o desempenho das R-trees. Em [TS93, TS94], Theodoridis e Sellis propõem novos algoritmos de percurso para inserção de um retângulo e de execução de *split*, onde se busca reduzir tanto a área como a sobreposição dos MBRs, além do *dead-space*, fator que que ainda não havia sido considerado explicitamente nos algoritmos anteriores. Eles também recomendam a classificação das entradas segundo algum valor antes da execução do *split*, o que poderia ser feito através de uma *space-filling curve* ou pelas coordenadas dos retângulos em uma das dimensões. Seus experimentos mostraram uma melhoria do desempenho da R-tree que a levaram a resultados semelhantes aos da R*-tree, que é normalmente apontada na literatura como melhor que a primeira.

Em [GLL97], García, López e Leutenegger apresentam um algoritmo de complexidade $O(n^2)$ para se encontrar a partição ótima de um nó que armazena $n$ retângulos em uma R-tree que indexa dados em um espaço bidimensional ($O(n^k)$ se o espaço for $k$-dimensional). A vantagem desse algoritmo reside no fato de que o número de formas de se dividir um conjunto de $n$ retângulos em dois subconjuntos é exponencial. Os autores mostram, no entanto, que se a função que calcula o custo de cada par de subconjuntos obedece a algumas condições, apenas $O(n^2)$ pares de MBRs delimitadores dos subconjuntos precisam ser verificados. Para tanto, eles introduzem, inicialmente, o conceito de *MBR âncora*. Dado um MBR $R_S$ que delimita um conjunto $S$ de retângulos que deve ser bipartido, um MBR $R_A$ de um subconjunto $A$ de $S$ é um MBR âncora se $R_A$ compartilha com $R_S$ *pelo menos* duas coordenadas, isto é, $x_{min}$, $y_{min}$, $x_{max}$ ou $y_{max}$. A partir daí, os autores mostram que:

- em cada par $(R_0, R_1)$ de MBRs que envolvem os subconjuntos de uma bipartição de $S$, pelo menos um é MBR âncora;
- só existem $O(n^2)$ MBRs âncora para um conjunto $S$ de retângulos.

O algoritmo apresentado em [GLL97] é válido se a função $f(R_0, R_1)$ que calcula o custo de uma bipartição de $S$ a partir dos MBRs $R_0$ e $R_1$ aumenta de valor sempre que há um aumento no comprimento do lado de qualquer um desses dois MBRs em qualquer das dimensões. Exemplos de funções que se enquadram nessas condições são a soma das áreas de $R_0$ e $R_1$, e a soma de seus perímetros. Os autores fazem notar que para funções desse tipo, dado um MBR $R_0$, o valor ótimo (ou seja, o menor) para $f(R_0, R_1)$ é obtido quando $R_1$ é o MBR que envolve somente os retângulos que não estão totalmente contidos

em $R_0$, pois qualquer outro $R_1$ que fosse escolhido teria um lado maior que aquele em alguma das dimensões e, conseqüentemente, o valor de $f(R_0, R_1)$ seria maior. Assumindo, sem perda de generalidade, que $R_0$ é um MBR âncora (já que pelo menos um dos dois tem que ser), e que existem somente $O(n^2)$ deles, apenas $O(n^2)$ pares de MBRs têm que ser avaliados.

Tais considerações deram origem ao primeiro algoritmo apresentado em [GLL97], que gera partições sem nenhuma restrição quanto ao número de elementos de cada subconjunto. A bipartição escolhida leva em conta apenas o valor ótimo para $f(R_0, R_1)$. Esse algoritmo foi, então, estendido pelos autores para atender a uma restrição de cardinalidade, considerando apenas as partições onde ela é respeitada. Infelizmente, essa extensão tem uma deficiência que o original não apresenta e que não é citada no trabalho. Se no conjunto de retângulos a serem repartidos um deles contém todos os outros, e entre estes nenhum toca a borda do primeiro, o algoritmo devolve uma partição vazia e a outra com todos os retângulos. Isso acontece porque o primeiro retângulo é também o único MBR âncora que se pode obter do conjunto, e como ele envolve todos os outros não há retângulos para serem associados ao segundo subconjunto. Como este resultado não pode ser aceito quando há uma restrição de cardinalidade, a operação de *split* falha.

O interesse em se investir na R-tree enquanto ela tem concorrentes mais eficientes em sua própria família (nos testes apresentados em [KF94], a Hilbert R-tree obteve resultados ainda melhores que os da R*-tree) provavelmente se deve à existência de pesquisas em outras áreas, como paralelismo [KF92] e controle de concorrência [NK93], com respeito às R-trees, e sua ausência com relação às outras.

# Capítulo 3

# Dados e Consultas

## 3.1 Introdução

Este capítulo descreve as características dos dados utilizados nos experimentos, o processo de sua obtenção e seus possíveis usos em outras pesquisas. Nele também são definidas as consultas executadas sobre esses dados.

## 3.2 Origem dos dados

O conjunto de dados sobre o qual foram realizados os testes foi extraído do SAGRE (Sistema Automatizado de Gerência de Rede Externa), uma aplicação que utiliza dados geográficos desenvolvida pelo Centro Pesquisa e Desenvolvimento da TELEBRÁS (CPqD). Sua função é automatizar os diversos processos relacionados ao cadastro, planejamento, projeto, implantação, operação, manutenção, expansão e gerência da rede externa das empresas operadoras do Sistema TELEBRÁS [CCH+96, Mag97]. O termo *rede externa* se refere a toda a rede de telecomunicações que se encontra fora das estações telefônicas, o que inclui a rede de canalização (conjunto de dutos enterrados conectados a caixas subterrâneas), a rede aérea (rede de cabos suspensos) e a rede subterrânea (rede de cabos que passam pela rede de canalização). O sistema mantém ainda informações sobre o mapa urbano básico[1], as estações telefônicas e os limites de gerência de rede (linhas que delimitam as áreas de abrangência de estações, distritos, etc.).

Os dados se referem à cidade de Valinhos, localizada a 7 km de Campinas e a 90 km da cidade de São Paulo, com uma população de 74.608 habitantes, segundo a contagem realizada pelo IBGE no ano de 1996 [IBG]. Ela foi escolhida pela TELEBRÁS para

[1] O mapa urbano básico (MUB) é formado por elementos como ruas, quadras, lotes, rios, lagos, acidentes geográficos e obras públicas de uma cidade, entre outros. Ele é a base para qualquer aplicação relativa a serviços utilitários (telefonia, eletricidade, saneamento, etc.).

os testes do sistema por ser uma cidade pequena, com uma rede externa relativamente estável, ao mesmo tempo em que apresenta grande parte dos elementos gerenciados.

## 3.3 Conversão dos dados

Uma das principais fontes de dados do SAGRE são plantas da rede em papel. A inclusão dessas informações no banco de dados requer um processo de conversão, que envolve sua digitalização e transcrição para um formato em que possam ser automaticamente consistidas e armazenadas. Essa conversão é feita por empresas especializadas, que normalmente fornecem os dados nos formatos dos SIGs que utilizam, que por serem definidos num nível muito primitivo dificultam o processo de consistência e aceitação. Além disso, como empresas diferentes podem usar formatos diferentes, haveria o trabalho adicional de tratar cada um deles.

Para facilitar a conversão e a aceitação dos dados, a TELEBRÁS definiu um formato próprio, a ser utilizado por qualquer empresa incumbida dessa tarefa [TEL95, MGS+94]. Para cada elemento de dados convertido, o formato descreve as seguintes informações (não necessariamente todas):

- seus dados convencionais (por exemplo, o número de pares de um cabo ou o nome de um logradouro);
- sua geometria, que pode ser definida pelas coordenadas de um único ponto ou de uma lista deles;
- coordenadas adicionais para a apresentação de sua representação gráfica ou identificação de elementos relacionados;
- coordenadas e ângulo de posicionamento de sua legenda, ou seja, o nome que o identifica no mapa.

O formato de conversão da TELEBRÁS foi utilizado neste trabalho no sentido inverso daquele para o qual foi definido, isto é, as informações armazenadas no banco de dados do SAGRE foram recolocadas nesse formato antes de serem usadas nos testes. Dois motivos levaram a esse procedimento:

- A criação de um conjunto de dados reais com atributos convencionais e espaciais que pudesse ser fornecido a outros pesquisadores. Tal conjunto poderia ser usado no teste de desempenho de sistemas espaciais ou convencionais, assim como no desenvolvimento de otimizadores de consultas que levassem em conta os dois tipos de atributos na escolha dos planos de execução.

- O desenvolvimento de programas que facilitassem à TELEBRÁS o transporte dos dados, o que também simplificaria a sua disponibilização para outras pesquisas, indo ao encontro do primeiro objetivo.

É bem verdade que esses mesmos dados já existiam no formato TELEBRÁS, uma vez que eles foram recebidos dessa forma das empresas que fizeram a conversão. O que ocorre é que após a conversão eles passaram por um processo de consistência e correção, de forma que a atitude correta é a utilização dos dados armazenados no sistema.

Uma vez que parte dos dados precisava ficar em sigilo, alguns atributos convencionais foram criptografados durante a extração. Para que sua representatividade não fosse perdida, manteve-se, após a codificação, a distribuição de valores de cada atributo semelhante à original.

Foram convertidos os elementos do mapeamento urbano básico (MUB), rede aérea (RA), canalização subterrânea (CS), limites de gerência de rede (LGR) e parte dos elementos das estações telefônicas (ET). A tabela 3.1 mostra o número de programas de conversão por grupo de elementos. Esses programas foram codificados em GML, uma linguagem semelhante ao SQL integrada ao VISION*, o sistema de informação geográfica que dá suporte ao SAGRE.

| Grupo | Número de programas |
|---|---:|
| Mapa urbano básico | 25 |
| Rede aérea | 26 |
| Canalização subterrânea | 8 |
| Estações telefônicas | 2 |
| Limites de gerência de rede | 5 |
| Total | 66 |

Tabela 3.1: Número de programas de conversão por grupo de elementos.

Os dados de Valinhos contêm ocorrências de 43 dos 66 elementos pertencentes a esses grupos, listados na tabela 3.2. A tabela 3.3 relaciona os elementos para os quais foram feitos programas de conversão apesar de não existirem nos dados de teste.

Alguns elementos apresentam certas particularidades em sua representação:

- Cada logradouro (rua, avenida, etc.) é composto por uma ou mais linhas centrais, que descrevem sua geometria. A linha central é um conjunto de segmentos de reta que passa ao longo do eixo de um logradouro, e é delimitada por pontos de intersecção com outros logradouros, como mostra a figura 3.1.

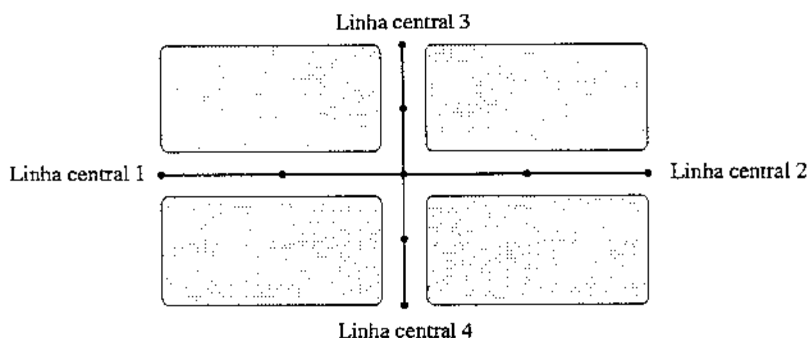



Figura 3.1: Dois logradouros, cada um composto por duas linhas centrais.

- Cada lote é representado como um ponto, assim como as divisas entre lotes.

Como se pode ver nas tabelas 3.2 e 3.3, nenhum elemento é representado como polígono, nem mesmo aqueles que delimitam uma área, os quais são representados como linhas poligonais fechadas. Acontece que o VISION* armazena atributos como área e perímetro para elementos cujas geometrias são representadas por polígonos, e como essa informação não era necessária para o SAGRE, seus projetistas optaram pelo uso de linhas poligonais.

## 3.4 Conjunto de MBRs utilizado nos testes

Após a conversão dos dados para o formato TELEBRÁS, foram criados os MBRs sobre os quais os testes foram realizados. A área total ocupada é de 361.398.257,25792 $m^2$, sendo as coordenadas mínimas e máximas as seguintes:

- $x_{min}$ = 286.586,042
- $x_{max}$ = 307.716,81
- $y_{min}$ = 7.448.789,53
- $y_{max}$ = 7.465.892,47

O conjunto é composto por 66.837 MBRs, dos quais os pontos representam 82,2%, contra 17,8% de retângulos correspondentes aos elementos representados por linhas poligonais. As tabelas 3.4 e 3.5 trazem, respectivamente, o número de pontos e retângulos gerados. A tabela 3.5 apresenta também as áreas mínima ($A_{min}$), máxima ($A_{max}$) e média ($A_{med}$) dos MBRs de cada elemento e do conjunto completo de retângulos. Finalmente, a tabela 3.6 mostra a distribuição dos tamanhos desses retângulos com relação à área total considerada.

Pode-se notar claramente que há uma grande variação nas áreas dos MBRs, mesmo se considerarmos apenas as ocorrências de um mesmo elemento. Essa característica dificulta a indexação desse tipo de conjunto de dados por métodos de acesso que se baseiam no comprimento do maior lado de MBR em cada dimensão para construir as janelas de consulta, como a 2dMAP21 e combinações de métodos de acesso a pontos com a técnica da abstração, apresentada na seção 2.3.1.

Além do conjunto completo de retângulos e pontos, mostrado na figura 3.2, foram utilizados dois sub-conjuntos para verificar o comportamento dos métodos de acesso na indexação de MBRs de ocorrências de um mesmo elemento: os pontos que representam os postes (figura 3.3) e os MBRs das quadras (figura 3.4).

Figura 3.2: MBRs dos dados de Valinhos.

Figura 3.3: Localização dos postes de Valinhos.

## 3.5 Consultas

A montagem de um conjunto de consultas reais é mais difícil que a obtenção dos dados. No caso do SAGRE isso não foi possível, pois o sistema não estava em operação, o que impediu a avaliação de fatores como distribuição das janelas de consulta, seus tamanhos e a freqüência com que cada um desses tamanhos aparece na carga de trabalho. Por outro lado, o uso de um conjunto de consultas reais poderia tornar os resultados dos experimentos excessivamente específicos.

Assim optou-se por utilizar conjuntos de janelas de diferentes tamanhos para que se pudesse verificar o comportamento dos métodos de acesso de acordo com a variação desse fator. Foram criados seis arquivos contendo, cada um, 1.000 retângulos de área

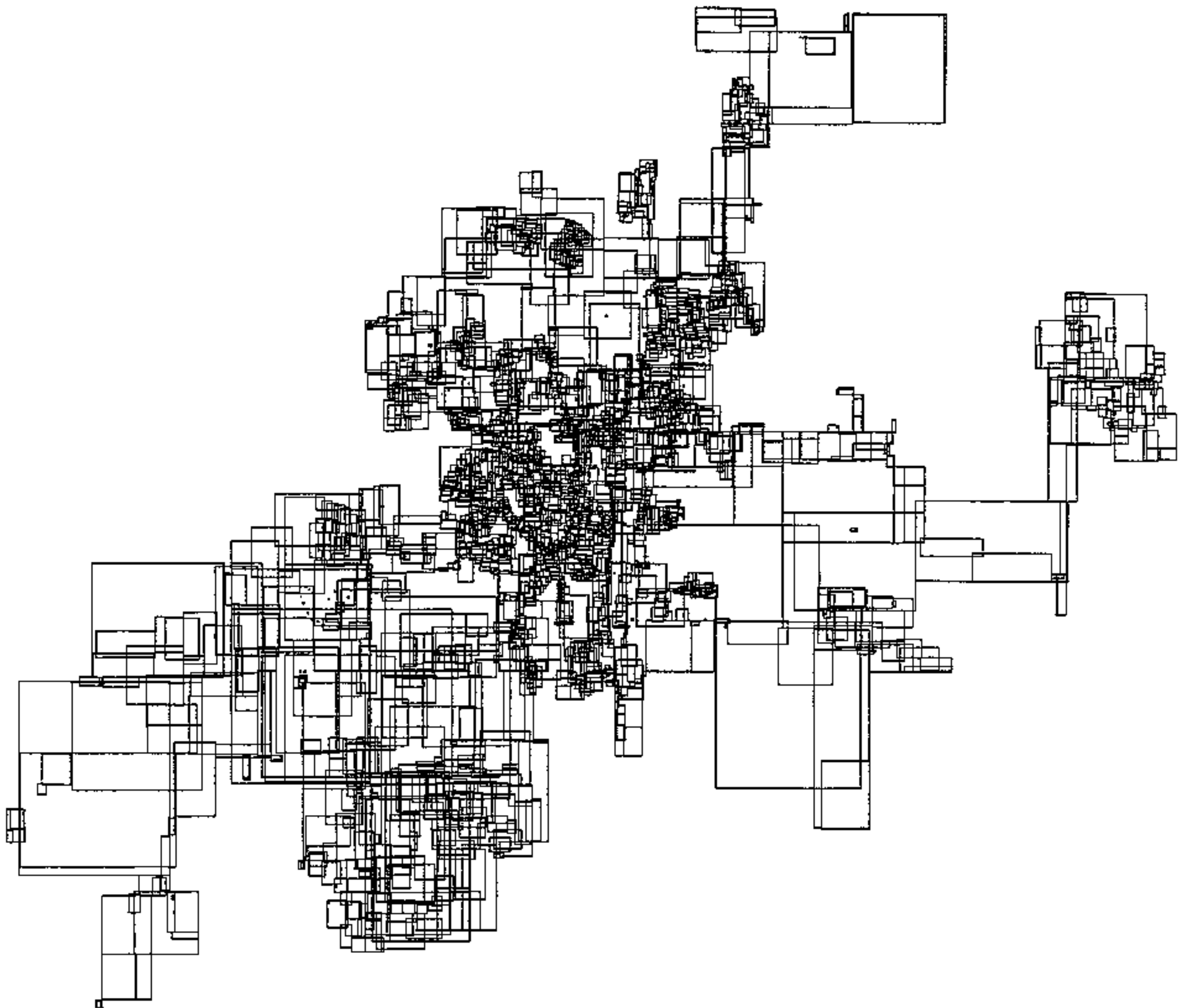


Figura 3.4: MBRs das quadras de Valinhos.

constante. As áreas foram fixadas em 0,0001%, 0,001%, 0,01%, 0,1%, 1% e 10% da área total considerada. Esses tamanhos foram escolhidos pelo fato de existirem objetos com áreas semelhantes a essas no conjunto de dados. Em cada arquivo foram gerados retângulos de cinco formatos: quadrados, retângulos com a extensão no eixo $x$ duas e três vezes maior que a extensão no eixo $y$, e retângulos com extensão no eixo $x$ duas e três vezes menor que a extensão no eixo $y$. Além desses, foi criado um arquivo formado por pontos para ser utilizado na execução de *point queries*.

Os centros de objetos escolhidos aleatoriamente do conjunto de dados serviram como centros das janelas geradas, o que deu às consultas uma distribuição semelhante à dos dados. Esse procedimento já havia sido utilizado em [ND97], e é coerente com o modo de operação de aplicações semelhantes ao SAGRE. De fato, em sistemas como esse, a maior

parte das transações são efetuadas sobre as regiões onde a densidade dos dados é maior, o que normalmente corresponde ao centro da cidade e bairros vizinhos. As figuras 3.5 a 3.11 mostram a disposição das janelas de consulta em cada arquivo.

| Elemento | Grupo | Geometria |
|---|---|---|
| Quadra | MUB | linha poligonal |
| Divisa de lote | MUB | ponto |
| Linha central | MUB | linha poligonal |
| Hidrografia | MUB | linha poligonal |
| Obra de arte | MUB | linha poligonal |
| Limite de faixa de domínio | MUB | linha poligonal |
| Edificação de destaque | MUB | linha poligonal |
| Indicação de lote | MUB | ponto |
| Rodovia | MUB | linha poligonal |
| Ferrovia | MUB | linha poligonal |
| Linha de transmissão | MUB | linha poligonal |
| Localidade | MUB | ponto |
| Logradouro | MUB | não tem |
| Edificação de destaque textual | MUB | ponto |
| Bloqueio | RA | não tem |
| Caixa terminal | RA | ponto |
| Lance de cabo | RA | linha poligonal |
| Subida de lateral | RA | linha poligonal |
| Caixa predial | RA | ponto |
| Emenda sem bloco | RA | ponto |
| Armário de distribuição | RA | ponto |
| Aterramento | RA | ponto |
| Equipamento de pupinização | RA | ponto |
| Poste | RA | ponto |
| Cordoalha | RA | linha poligonal |
| Âncora | RA | ponto |
| Pé de galinha | RA | ponto |
| Percurso | RA | não tem |
| Indicação de seção de serviço | RA | ponto |
| Indicação de caixa subterrânea | RA | ponto |
| Lance de cabo com múltiplas transições | RA | linha poligonal |
| Caixa subterrânea | CS | ponto |
| Pedestal de armário de distribuição | CS | ponto |
| Lance de dutos | CS | linha poligonal |
| Duto lateral | CS | linha poligonal |
| Ponto de acesso predial | CS | ponto |
| Ponto de interesse | CS | ponto |
| Estação telefônica | ET | ponto |
| Projeção predial da estação | ET | linha poligonal |
| Limite de área de estação | LGR | linha poligonal |
| Limite de seção de serviço | LGR | linha poligonal |
| Limite de área de tarifa básica | LGR | linha poligonal |
| Limite de distrito | LGR | linha poligonal |

Tabela 3.2: Elementos convertidos dos dados de Valinhos. Grupos: MUB – mapeamento urbano básico; RA – rede aérea; CS – canalização subterrânea; ET – estações telefônicas; LGR – limites de gerência de rede.

| Elemento | Grupo | Geometria |
|---|---|---|
| Quadra planejada | MUB | linha poligonal |
| Acidente geográfico | MUB | linha poligonal |
| Limite de localidade | MUB | linha poligonal |
| Limite de estado | MUB | linha poligonal |
| Meio-fio | MUB | linha poligonal |
| Edificação de destaque projetada | MUB | linha poligonal |
| Calçada | MUB | linha poligonal |
| Marco geográfico | MUB | ponto |
| Rosa dos ventos | MUB | ponto |
| Meio-fio indefinido | MUB | linha poligonal |
| Edificação de destaque textual projetada | MUB | ponto |
| Vinculação | RA | ponto |
| Armário de equipamentos | RA | ponto |
| Capacitor | RA | ponto |
| Reserva de pares | RA | ponto |
| Carrier | RA | ponto |
| Repetidor carrier | RA | ponto |
| Indicação de rota | RA | ponto |
| Lance de cabo enterrado | RA | linha poligonal |
| Fachada | RA | ponto |
| Derivação de dutos | RA | ponto |
| Limite de rota | RA | Linha poligonal |

Tabela 3.3: Elementos não existentes nos dados de Valinhos para os quais foram codificados programas de conversão. Grupos: MUB – mapeamento urbano básico; RA – rede aérea.

| Elemento | Ocorrências |
|---|---|
| Divisa de lote | 18.200 |
| Indicação de lote | 14.539 |
| Caixa terminal | 2.336 |
| Caixa predial | 124 |
| Emenda sem bloco | 2.910 |
| Armário de distribuição | 32 |
| Aterramento | 799 |
| Equipamento de pupinização | 22 |
| Poste | 13.813 |
| Âncora | 47 |
| Estação telefônica | 2 |
| Caixa subterrânea | 383 |
| Pedestal de armário de distribuição | 46 |
| Localidade | 2 |
| Edificação de destaque textual | 977 |
| Pé de galinha | 427 |
| Indicação de seção de serviço | 76 |
| Indicação de caixa subterrânea | 76 |
| Ponto de acesso predial | 135 |
| Ponto de interesse | 3 |
| Total | 54.949 |

Tabela 3.4: Número de ocorrências dos elementos representados como pontos.

| Elemento | Ocorrências | $A_{min}$ $(m^2)$ | $A_{max}$ $(m^2)$ | $A_{med}$ $(m^2)$ |
|---|---|---|---|---|
| Quadra | 2.473 | 0,00006200 | 6.717.282,90688190 | 89.276,74664189 |
| Linha central | 3.350 | 0.00000000 | 2.108.596,16732032 | 16.776,92856100 |
| Hidrografia | 476 | 0.00000000 | 2.689.065,66774404 | 57.788,01956186 |
| Obra de arte | 296 | 0,00720000 | 68.620,73914521 | 1.597,02142492 |
| Limite de faixa de domínio | 1 | 26.697,34880004 | 26.697,34880004 | 26.697,34880004 |
| Edificação de destaque | 6 | 1.276,88180001 | 11.927,73209998 | 6.070,24735466 |
| Rodovia | 86 | 0,07040000 | 8.003.921,33206586 | 395.244,68888080 |
| Ferrovia | 53 | 0,12114600 | 1.685.405,05062396 | 66.474,97414048 |
| Linha de transmissão | 2 | 649,73885600 | 170.821,22994426 | 85.735,484400136 |
| Limite de área de estação | 1 | 13.616.220,74062492 | 13.616.220,74062492 | 13.616.220,74062492 |
| Limite de seção de serviço | 253 | 18,13713600 | 5.620.612,23226038 | 178.991,24850259 |
| Limite de área de tarifa básica | 18 | 37,17203300 | 15.297.370,49430146 | 4.288.203,08702874 |
| Limite de distrito | 8 | 1.850,91141794 | 154.995.243,20999992 | 62.161.482,22331629 |
| Lance de cabo | 3.431 | 0,51996200 | 1.443.877,80444018 | 5.965,72186906 |
| Subida de lateral | 123 | 1,36090100 | 5.638,91934601 | 455,93831466 |
| Cordoalha | 642 | 0.00000000 | 2.477,51123500 | 155,04705760 |
| Lance de dutos | 410 | 0.00000000 | 28.399,26297901 | 4.336,89958628 |
| Duto lateral | 246 | 0.00000000 | 2.653,35648996 | 137,35504047 |
| Lance de cabo com múltiplas transições | 8 | 151,70806000 | 2.865,71558400 | 943,48361838 |
| Projeção predial da estação | 5 | 0,00072800 | 2.048,49884402 | 867,19100340 |
| Total | 11.888 | 0.00000000 | 154.995.243,20999992 | 83.995,71515549 |

Tabela 3.5: Número de ocorrências dos elementos representados como linhas poligonais. As áreas se referem aos seus MBRs.

| $Área_{mbr}/Área_{total}$ | | | $Área_{mbr}(km^2)$ | | | Ocorrências | Num. acumulado | % do total | % acumulada |
|---|---|---|---|---|---|---|---|---|---|
| | | $< 0,0001\%$ | | | $< 0,00036$ | 2.924 | 2.924 | 24,596231 | 24,596231 |
| $\geq 0,0001\%$ | $\wedge$ | $< 0,001\%$ | $\geq 0,00036$ | $\wedge$ | $< 0,0036$ | 4.282 | 7.206 | 36,019515 | 60,615747 |
| $\geq 0,001\%$ | $\wedge$ | $< 0,01\%$ | $\geq 0,0036$ | $\wedge$ | $< 0,036$ | 3.570 | 10.776 | 30,030283 | 90,646030 |
| $\geq 0,01\%$ | $\wedge$ | $< 0,1\%$ | $\geq 0,036$ | $\wedge$ | $< 0,36$ | 888 | 11.664 | 7,469717 | 98,115747 |
| $\geq 0,1\%$ | $\wedge$ | $< 1\%$ | $\geq 0,36$ | $\wedge$ | $< 3,6$ | 196 | 11.860 | 1,648721 | 99,764468 |
| $\geq 1\%$ | $\wedge$ | $< 10\%$ | $\geq 3,6$ | $\wedge$ | $< 36$ | 24 | 11.884 | 0,201884 | 99,966353 |
| $\geq 10\%$ | | | $\geq 36$ | | | 4 | 11.888 | 0,033647 | 100,000000 |

Tabela 3.6: Distribuição dos tamanhos das áreas dos MBRs em relação à área total (elementos representados como linhas poligonais).

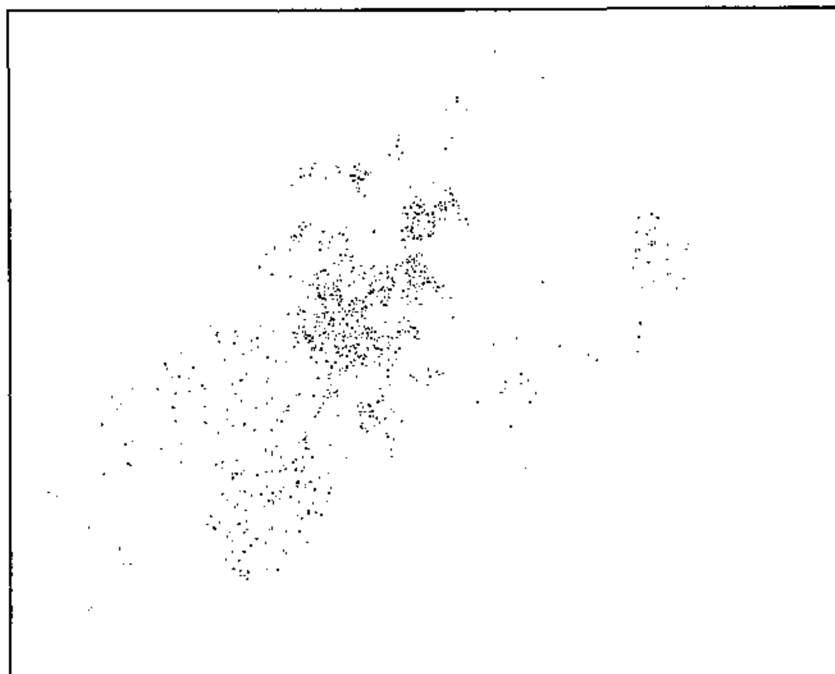


Figura 3.5: Janelas de consulta com 0,0001% da área total.

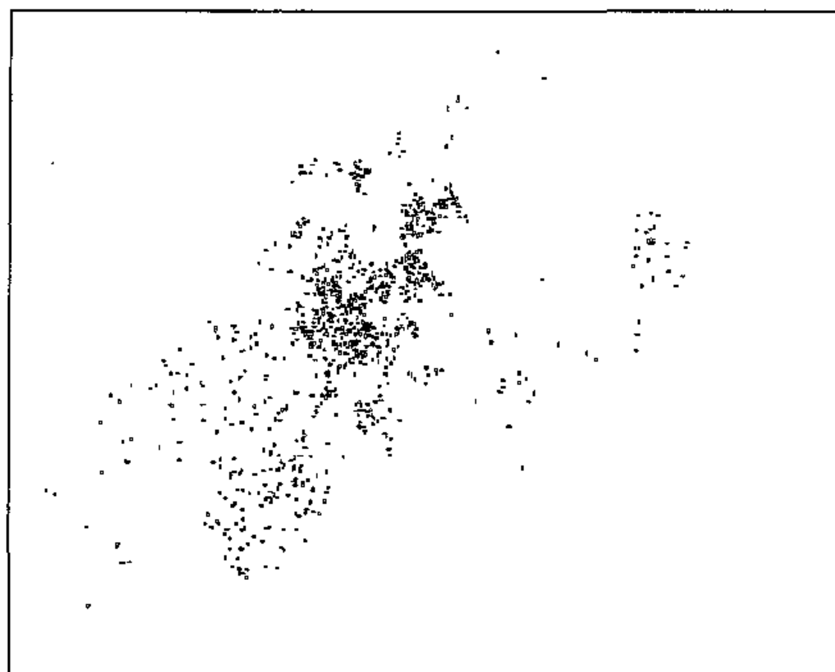


Figura 3.6: Janelas de consulta com 0,001% da área total.

Figura 3.7: Janelas de consulta com 0,01% da área total.

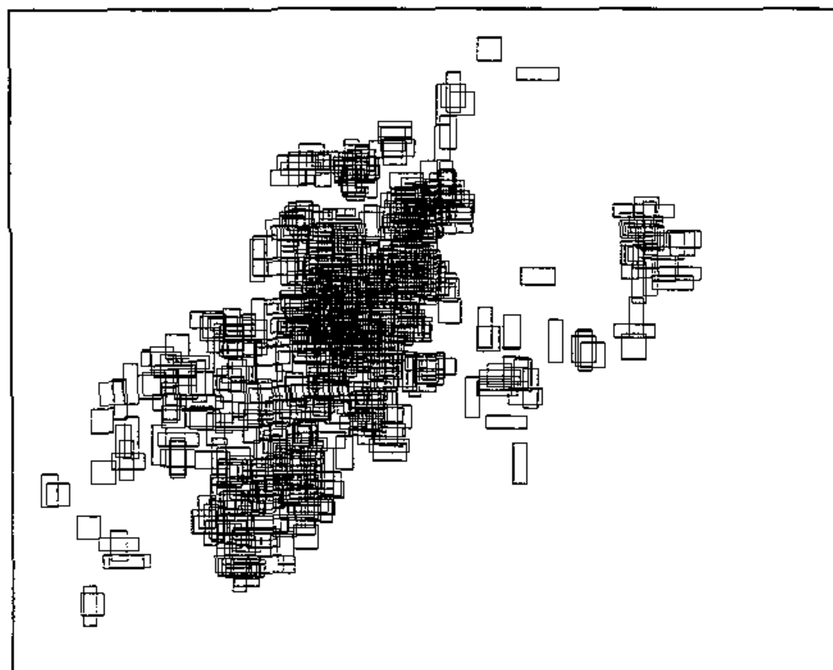


Figura 3.8: Janelas de consulta com 0,1% da área total.

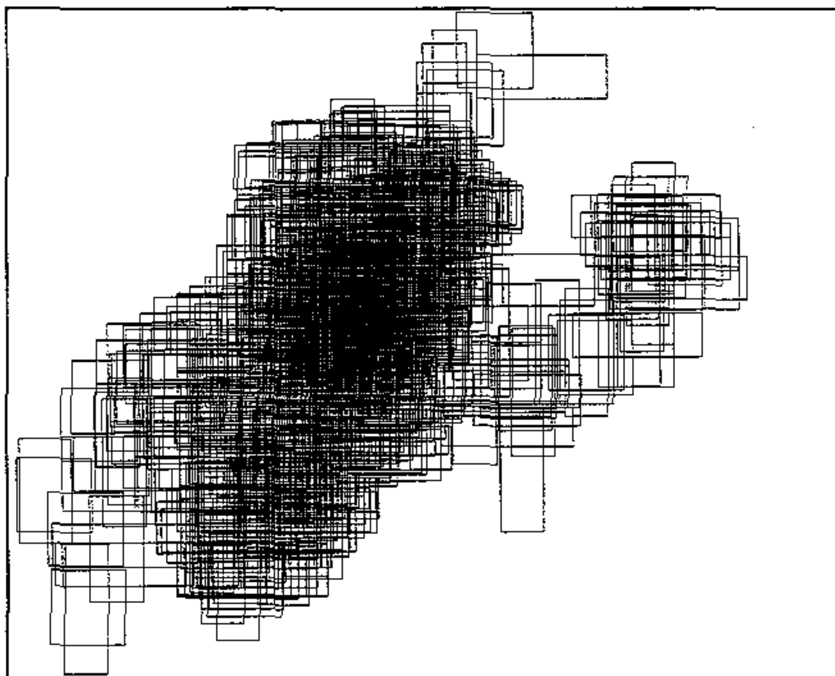


Figura 3.9: Janelas de consulta com 1% da área total.

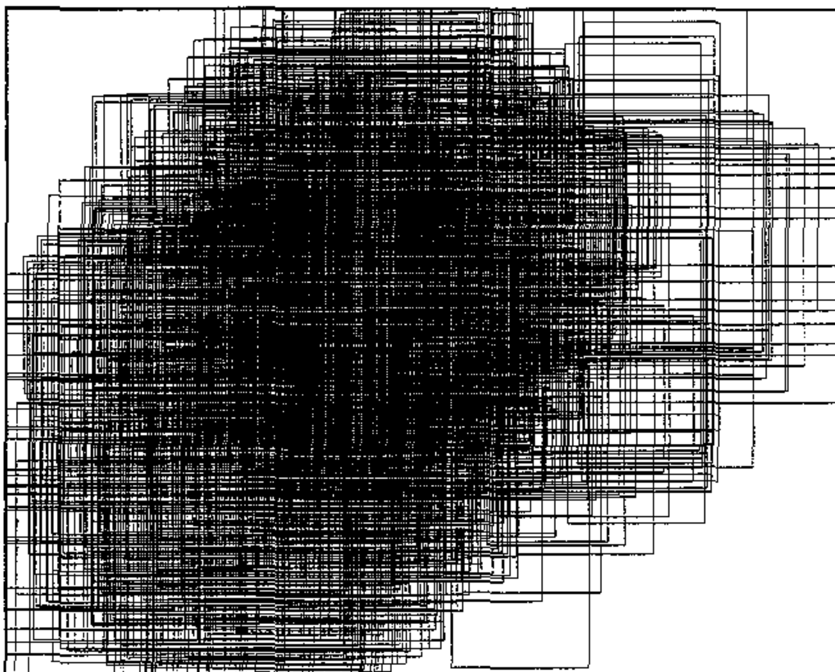


Figura 3.10: Janelas de consulta com 10% da área total.

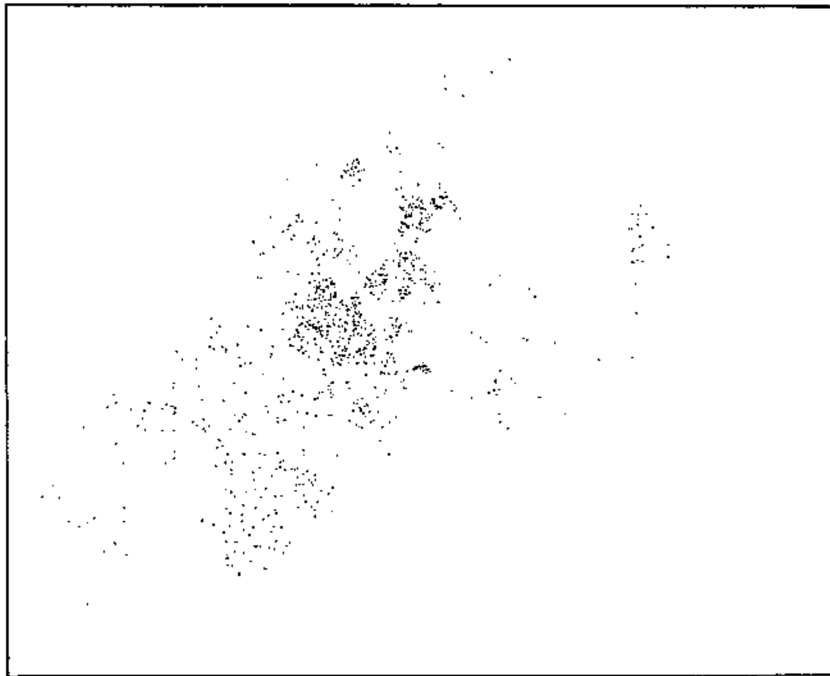


Figura 3.11: Localização dos pontos utilizados nas *point queries*.

# Capítulo 4

# Comparação de desempenho

## 4.1 Introdução

Este capítulo detalha a execução e os resultados dos testes de desempenho, onde foram avaliadas duas variantes da R-tree (Guttman e Greene), cinco da R*-tree, e a $R^+$-tree. A exposição começa com a definição dos procedimentos, parâmetros e critérios utilizados nos experimentos, na seção 4.2. A seção 4.3 trata das alterações feitas no código original de Cox devido a problemas identificados durante os testes. A seção 4.4 apresenta e avalia os resultados obtidos. Finalmente, a seção 4.5 faz algumas comparações entre os resultados deste trabalho e os de Cox [Cox91].

## 4.2 Descrição dos testes

### 4.2.1 Configuração dos métodos de acesso

A intenção de se comparar os resultados desta pesquisa com os de Cox sugeria que os valores de parâmetros utilizados em seu trabalho para configurar os métodos de acesso fossem mantidos aqui. Entretanto, muitos desses valores foram mudados com o objetivo de se obter um ambiente mais próximo de uma situação real para a realização dos testes. Em seus experimentos, Cox utilizou os seguintes parâmetros:

- Coordenadas dos retângulos: representadas por variáveis do tipo *float* (4 bytes)[1].
- Tamanho das páginas de disco: 1.024 bytes, que é apontado em [BKSS90] com sendo o limite inferior para tamanhos reais de páginas em disco.

[1]Os métodos de acesso foram implementados na linguagem C.

- Capacidade de armazenamento do *cache*: 50 páginas, o que representava em torno de 16% do número total de páginas armazenadas na R-tree e na R*-tree, e aproximadamente 7% das páginas armazenadas na $R^+$-tree.
- Número máximo de entradas por nó *(M)*: 50 para a R-tree e a R*-tree, e 28 para a $R^+$-tree. Esses valores foram calculados automaticamente a partir tamanho da página e do tamanho das entradas, que por sua vez depende do tipo de variável usado para representar as coordenadas. O número máximo de entradas é menor na $R^+$-tree porque suas entradas armazenam dois retângulos: o MBR e o MaxRect (veja a seção 2.7.2).
- Número mínimo de entradas por nó ($m$): para a R-tree e a R*-tree, Cox fixou o valor de $m$ igual a 40% de $M$, conforme sugerido em [BKSS90], o que resultou em $m = 20$. Para a $R^+$-tree o valor de $m$ é 1.
- Número de entradas a serem reinseridas no tratamento de *overflow* da R*-tree ($p$): 15 (isto é, 30% de $M$, valor também sugerido em [BKSS90]).

Os seguintes parâmetros de configuração foram usados neste trabalho:

- Coordenadas dos retângulos: representadas por variáveis do tipo *double* (8 bytes) para evitar a perda de precisão dos dados.
- Tamanho das páginas de disco: 4.096 bytes. Apesar desse valor não ser utilizado normalmente em testes de desempenho de métodos de acesso espaciais, tamanhos de páginas maiores que 1.024 bytes são freqüentes em aplicações reais. Em [Gre89], Greene afirma que seus experimentos indicaram que o tamanho ótimo de página para R-trees está entre 100 e 200 entradas por nó, o que naquele trabalho correspondia a 2.414 e 4.814 bytes respectivamente. Utilizando os parâmetros aqui fixados este intervalo estaria entre 4.096 e 8.192 bytes, levando-se em conta apenas os múltiplos de 1KB. Segundo uma análise apresentada em outro artigo [GG97], o valor ótimo para páginas de índices armazenados em disco está entre 8 e 32KB, considerando as tecnologias atuais.
- Capacidade de armazenamento do *cache*: esse parâmetro foi variado de acordo com o conjunto de dados indexado. Para o conjunto total dos dados de Valinhos foi utilizado um *cache* de 50 páginas; para o conjunto de postes, 10 páginas; para as quadras, 2 páginas. Em todos os casos o número de páginas do *cache* representa em torno de 5% das páginas armazenadas nos métodos de acesso.
- Número máximo de entradas por nó *(M)*: 102 para a R-tree e a R*-tree, e 56 para a $R^+$-tree, calculados a partir tamanho da página e do tamanho das entradas.

- Número mínimo de entradas por nó *(m)*: 40 para a R-tree e a R*-tree (40% de $M$), e 1 para a $R^+$-tree.
- Número de entradas a serem reinseridas no tratamento de *overflow* da R*-tree ($p$): 30 (ou seja, 30% de $M$).

### 4.2.2 Critérios de comparação

Neste trabalho são adotados os mesmos critérios de comparação de [Cox91]:

- número de páginas acessadas em disco nas consultas e atualizações, cujo uso em lugar do tempo de resposta permite que os testes sejam feitos em uma estação de trabalho não dedicada;
- espaço de armazenamento em disco, medido em termos do número de páginas ocupadas em cada método de acesso;
- ocupação média dos nós das estruturas.

### 4.2.3 Tipos de consulta

A avaliação dos métodos de acesso com relação à recuperação dos dados foi feita a partir do número de páginas de disco acessadas durante a execução de três tipos de consulta: *point queries*, *range queries* para determinação de intersecção e *range queries* para determinação dos objetos que contêm a janela de consulta. Não foram executadas *range queries* para determinação dos objetos contidos na janela, pois os resultados seriam iguais aos das *range queries* para determinação de intersecção, já que a diferença entre os algoritmos que realizam esses dois tipos de consulta está apenas no tratamento das entradas dos nós folha, onde não são causados novos acessos a páginas do índice. Também não foram executadas *range queries* para determinação dos retângulos que contêm a janela de consulta para o arquivo de postes, porque isso não faria sentido.

Os retângulos contidos nos seis arquivos descritos na seção 3.5 foram usados como janelas de consulta nas *range queries*, e o arquivo de pontos descrito na mesma seção, nas *point queries*.

### 4.2.4 Atualizações

Uma das características da aplicação de onde os dados utilizados neste trabalho foram extraídos é a baixa ocorrência de exclusões de dados espaciais, já que a tendência de uma cidade, assim como a de uma rede de telefonia é crescer. Ainda assim existem alguns casos em que remoções são executadas, como por exemplo:

- na substituição de um equipamento, em que a exclusão do objeto antigo é seguida pela inserção de um novo com coordenadas semelhantes, se não iguais;
- na elaboração de projetos de expansão da rede, onde vários elementos podem ser incluídos e excluídos enquanto os projetistas analisam as alternativas.

Para verificar como as atualizações afetam o desempenho dos métodos de acesso, os testes foram divididos em três fases, como nos experimentos de Cox [Cox91]:

- Fase 1: inserção dos MBRs dos dados e realização das consultas descritas na seção 4.2.3. Os testes desta fase simularam a manipulação de dados estáticos.
- Fase 2: remoção aleatória[2] de metade dos objetos indexados e posterior reexecução das consultas.
- Fase 3: intercalação de 10.000 inserções com 10.000 remoções aleatórias e posterior reexecução das consultas. O ponto de partida para as atualizações desta fase foram os índices resultantes das remoções da fase 2. Cada objeto a ser incluído em um dado momento foi escolhido aleatoriamente entre os excluídos na fase 2 mais os excluídos até aquele momento na fase 3. As inclusões foram intercaladas com as exclusões. Nesta fase e na anterior foram simulados ambientes onde os dados são dinâmicos.

A primeira fase foi executada para os três conjuntos de dados utilizados — o conjunto completo dos dados de Valinhos (66.837 objetos), o conjunto de postes (13.813 objetos) e conjunto de quadras (2.473 objetos) — enquanto as duas últimas foram realizadas apenas para o conjunto completo.

O número de acessos a disco também foi medido durante as atualizações. Em cada fase, o *cache* foi esvaziado após a execução das atualizações, assim como após o processamento de cada tipo de consulta para cada tamanho de janela.

## 4.3 Alterações na implementação

Em seu trabalho, Cox implementou e avaliou o desempenho de cinco métodos de acesso: a R-tree como proposta por Guttman, a R-tree de Greene, a R*-tree sem reinserção de entradas no tratamento de *overflow* (que ele chamou simplesmente de R*-tree), a R*-tree com reinserção (chamada por ele de R*-treeR) e a R$^+$-tree. De início, essas implementações foram utilizadas também nos testes deste trabalho praticamente sem alterações, salvo

[2] Apesar das atualizações serem aleatórias, os mesmos objetos foram envolvidos, na mesma ordem, para todos os métodos de acesso.

aquelas relacionadas à configuração dos índices, descritas na seção 4.2.1. No entanto, as duas variantes da R*-tree apresentaram um comportamento geral bastante diferente do esperado, assim como a R+-tree em um caso particular. A busca dos motivos dessas diferenças levaram à detecção de falhas na implementação desses métodos de acesso, assim como de alguns detalhes no algoritmo de reinserção da R*-tree que o tornava diferente da proposta original [BKSS90].

A seguir são descritas as falhas encontradas e as alterações feitas para saná-las. Também são apontadas as diferenças no algoritmo citado.

### 4.3.1 R*-tree

Um dos problemas encontrados na R*-tree foi o tempo gasto para a carga dos dados. Só para se ter uma idéia, a carga dos 66.837 retângulos na R-tree durava por volta de 3,5 minutos de CPU, enquanto que na R*-tree sem reinserção o tempo de carga era de aproximadamente 50 minutos, e na R*-tree com reinserção, aproximadamente 120 minutos[3].

O motivo desse comportamento estava no procedimento que escolhe, em cada nível, a sub-árvore onde o novo retângulo deve ser inserido. No nível imediatamente superior às folhas, isso era feito procurando-se a entrada que ao receber o novo retângulo teria o menor aumento de *overlap* com as outras entradas do nó, e resolvia-se os empates tomando-se a entrada cujo retângulo sofresse menor aumento de área. Acontece que esse algoritmo tem complexidade $O(n^2)$, já que ele calcula a sobreposição de cada entrada com todas as outras do nó. A solução foi verificar primeiro o aumento de área que cada entrada sofreria caso recebesse o novo retângulo, o que tem uma complexidade $O(n)$, e somente se todas as entradas sofressem aumento de área o teste de aumento de *overlap* seria executado. É claro que se houvesse uma entrada que não sofresse aumento de área, seu aumento de *overlap* também seria zero, e ela seria a escolhida. Com essas alterações, o tempo de carga da R*-tree sem reinserção caiu para um valor em torno de 4,5 minutos de CPU, e o da R*-tree com reinserção para 11,5 minutos, aproximadamente.

Outro dado que chamou a atenção foi o maior número de páginas alocadas e, conseqüentemente, a menor ocupação média dos nós nas R*-trees que na R-tree. Para armazenar todos os retângulos de Valinhos foram alocadas 981 páginas para a R-tree, e a ocupação média ficou em 67,8%, enquanto que a R*-rtree sem reinserção precisou de 1.256 páginas, com uma ocupação média de 53,2%, e a R*-tree com reinserção, 1.115 páginas, com ocupação média de 59,7%. Esse comportamento se refletiu no desempenho

[3]Esses tempos são apenas para ilustrar a diferença na carga entre a R-tree e as duas R*-trees. Como Cox não se preocupou com a otimização do código, já que o principal critério de comparação era o número de acessos a disco e não o tempo de resposta, não se deve tomá-los como padrões para esses métodos de acesso. Nota: tempos obtidos da coluna TIME da saída do comando *ps* do Unix.

das consultas, onde a R-tree venceu em todos os casos, contrariando praticamente toda a literatura a respeito.

Inicialmente se pensou que isso se devesse a alguma particularidade nos dados, e tentou-se encontrar alguma explicação nesse sentido, pois um melhor desempenho da R-tree com relação às outras duas já havia sido reportado por Cox para os casos em que os dados eram formados por uma combinação de pontos, retângulos pequenos e retângulos grandes. Entretanto, a resposta foi encontrada, mais uma vez, no procedimento de escolha da sub-árvore para inserção do novo retângulo. O que ocorria é que quando havia um empate na escolha da entrada pelo critério do aumento de *ovelap* e também pelo critério de aumento de área, a entrada escolhida era a primeira encontrada entre as que haviam empatado, e não a que tivesse a menor área entre elas, como proposto em [BKSS90]. Note que isso ocorria sempre que a inserção de um retângulo não causava nenhum aumento de área nos MBRs de mais de uma entrada do nível imediatamente superior às folhas.

A correção desse procedimento fez com que o número de páginas alocadas à R*-tree sem reinserção caísse para 974, com uma ocupação média de 68,3%, e o número de páginas da R*-tree com reinserção baixasse para 908, com ocupação média de 73,1%.

Duas diferenças encontradas no algoritmo de reinserção da R*-tree com relação ao proposto em [BKSS90] também ocasionaram alterações no código:

- Apenas a rotina para realizar o *far reinsert* estava presente, isto é, a reinserção das $p$ entradas cujos MBRs tinham os centros mais distantes do centro do nó eram reinseridas começando por aquela cujo MBR tinha o centro mais distante. Em [BKSS90], os autores indicam o *close reinsert* (a inserção das $p$ entradas com centros mais distantes do centro do MBR do nó a partir daquela que tem o centro menos distante) como o que proporciona o melhor desempenho nas consultas. Assim, acrescentou-se o *close reinsert* a esse algoritmo, para que se pudesse comparar as duas formas de reinserção.

- Na escolha das entradas a serem reinseridas são feitas duas classificações, e não uma, como proposto em [BKSS90]. Para cada dimensão, o procedimento de Cox ordena as entradas de acordo com a distância naquela dimensão entre os centros de seus MBRs e o centro do MBR do nó. Depois é usada apenas a classificação cuja retirada dos $p$ retângulos mais distantes resultar em menor área. Embora em [BKSS90] não haja qualquer referência sobre a forma de se obter a distância entre o centro do MBR de uma entrada e o centro do MBR do nó, o usual é calculá-la como a raiz quadrada da soma dos quadrados das distâncias em cada dimensão. Depois as entradas podem ser classificadas uma única vez, segundo os valores obtidos. Para efeito de comparação, criou-se uma versão da rotina de reinserção que trabalha dessa forma.

Os testes foram executados, então, sobre cinco variantes da R*-tree, uma sem reinserção e quatro com reinserção, a saber:

- com a rotina de Cox para escolha das $p$ entradas a serem reinseridas e *close reinsert*;
- com a rotina de Cox para escolha das $p$ entradas a serem reinseridas e *far reinsert*;
- com a rotina original para escolha das $p$ entradas a serem reinseridas (como proposta em [BKSS90]) e *close reinsert*;
- com a rotina original para escolha das $p$ entradas a serem reinseridas (como proposta em [BKSS90]) e *far reinsert*.

### 4.3.2 $R^+$-tree

A implementação da $R^+$-tree com dois retângulos em cada entrada e a conseqüente diminuição do número máximo de entradas por nó fez com que o desempenho desse método de acesso ficasse muito abaixo dos demais. No entanto, quando os testes tratavam da execução de *range queries* para determinação dos objetos que continham a janela de consulta, a $R^+$-tree ganhava sempre, e sua vantagem se tornava maior à medida que o tamanho das janelas crescia.

Esse comportamento não era lógico, pois na R-tree e na R*-tree os algoritmos para a avaliação desse tipo de consulta consideram, em cada nó não-folha, apenas as entradas cujos MBRs contêm a janela, enquanto que na $R^+$-tree todas as entradas de nós não-folha cujos MBRs a intersectam devem ser levadas em conta, já que um retângulo que inclui a janela pode ser seccionado e armazenado em várias sub-árvores. Logo, o desempenho da $R^+$-tree deveria ser inferior ao das outras estruturas também nessas consultas. Além disso, a diferença deveria crescer com o aumento do tamanho da janela, pois as chances de sub-árvores serem eliminadas nos níveis superiores da R-tree e da R*-tree se tornam maiores, enquanto que a consulta na $R^+$-tree deve continuar descendo na estrutura até atingir todas as folhas cujos MBRs intersectam a janela, que provavelmente serão em maior número a cada aumento de tamanho.

O motivo para o bom desempenho inesperado da $R^+$-tree era que seu algoritmo considerava, nos nós não-folha, apenas as entradas que continham a janela de consulta, como na R-tree e na R*-tree, e não todas as que a intersectavam. Isso dava a ela uma grande vantagem, pois os retângulos das entradas de seus níveis superiores são menores que os das outras duas estruturas. No entanto, o algoritmo devolvia respostas erradas. Após a correção, o comportamento da $R^+$-tree passou a ser o esperado, ou seja, as respostas passaram a ser corretas e o desempenho, ruim.

Para diminuir a desvantagem da $R^+$-tree nesses testes, desenvolveu-se neste trabalho outro algoritmo para determinação dos retângulos que incluem uma janela de consulta:

1. faça da raiz o nó corrente;
2. enquanto o nó corrente é não-folha faça:
   (a) encontre a primeira entrada cujo MBR intersecte a janela de consulta;
   (b) faça do nó apontado por essa entrada o nó corrente;
3. devolva como resposta todas as entradas dessa folha cujos MBRs incluam a janela.

Como se pode ver, o algoritmo visita apenas um nó em cada nível da estrutura. Não sabemos se ele é original, pois é muito simples. Mas como existe a dúvida, é necessário provar que ele está correto.

Em primeiro lugar, se um retângulo de dados não inclui a janela de consulta ele não fará parte da resposta, o que é garantido pelo passo 3. Então resta provar que se existem retângulos que incluem a janela, todos eles serão reportados, ou seja, alguma folha é atingida e todos eles estão representados nessa folha.

Partindo do princípio de que o algoritmo da $R^+$-tree para determinação dos retângulos que *intersectam* uma janela de consulta está correto, podemos deduzir que sempre que há um retângulo nessas condições armazenado na árvore existe um caminho entre a folha em que ele se encontra e a raiz, em que todos os nós têm pelo menos uma entrada cujo MBR intersecta a janela de consulta. Sendo assim, e considerando que a inclusão é um caso particular de intersecção, conclui-se que o novo algoritmo sempre atinge uma folha se há algum retângulo que inclui a janela de consulta armazenado na $R^+$-tree.

Sejam: $J$ a janela de consulta, $f$ a folha atingida pelo algoritmo, e $R_f$ o MBR associado a ela. Seja $R$ um retângulo de dados que contém $J$. Suponha que $R$ não está representado em $f$. Seja $n$ o número de folhas onde $R$ está representado, $n \geq 1$. Chamemos de $f_i$ cada uma dessas folhas, e de $R_{fi}$ o MBR associado a $f_i, 1 \leq i \leq n$. Então $R \subseteq (\bigcup R_{fi}), 1 \leq i \leq n$. Mas como $R_f \cap J \neq \emptyset$, temos que $R \cap R_f \neq \emptyset$. Logo, $R_f \cap (\bigcup R_{fi}) \neq \emptyset$, e $\exists\ R_{fi}, 1 \leq i \leq n$, tal que $R_{fi} \cap R_f \neq \emptyset$, o que é um absurdo, pois contraria as propriedades da $R^+$-tree. Então $R$ está representado em $R_f$ $\square$

A figura 4.1 compara o desempenho deste algoritmo com o original. As consultas foram executadas sobre os MBRs dos dados completos de Valinhos.

Devido à melhora do desempenho, o novo algoritmo foi utilizado nos testes deste trabalho.

## 4.4 Apresentação e avaliação dos resultados

A seguir são apresentados os gráficos correspondentes aos valores obtidos nos testes. Durante a exposição serão adotadas as seguintes abreviações:

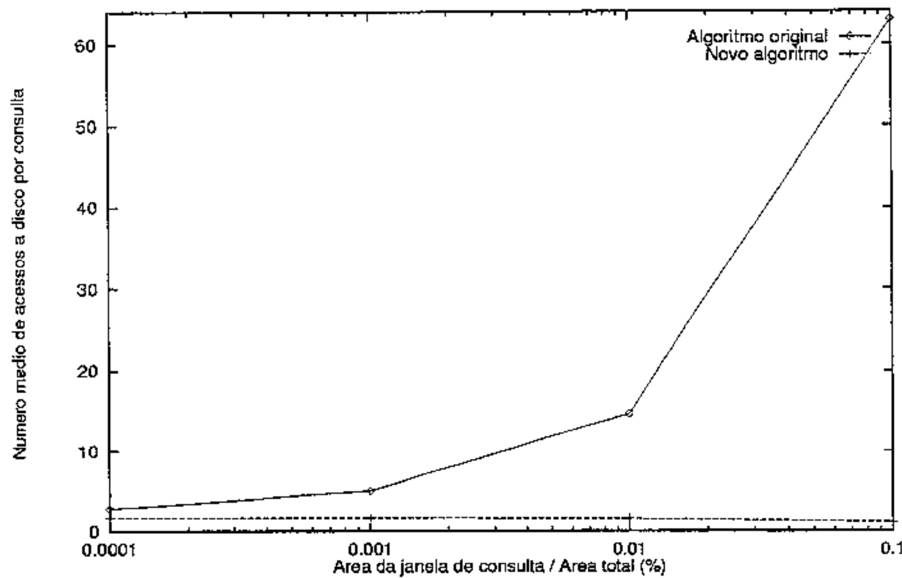



Figura 4.1: Comparação do desempenho dos dois algoritmos de determinação, na R$^+$-tree, dos retângulos que incluem uma janela de consulta.

- R-tree — R-tree de Guttman com algoritmo quadrático de *split*;
- R-treeG — R-tree de Greene;
- R*-treeSR — R*-tree sem a rotina de reinserção;
- R*-treeCC — R*-tree com a rotina de Cox para escolha das $p$ entradas a serem reinseridas e *close reinsert*;
- R*-treeCF — R*-tree com a rotina de Cox para escolha das $p$ entradas a serem reinseridas e *far reinsert*;
- R*-treeOC — R*-tree com a rotina original para escolha das $p$ entradas a serem reinseridas e *close reinsert*;
- R*-treeOF — R*-tree com a rotina original para escolha das $p$ entradas a serem reinseridas e *far reinsert*.

Os gráficos referentes às consultas apresentam o desempenho de cada método de acesso normalizado em função do desempenho da R*-treeOC. Esse método de acesso foi escolhido como referência por dois motivos:

- a R*-tree é considerada na literatura como o índice de melhor desempenho entre os que estão sendo testados;

- em [BKSS90] afirma-se que a rotina de *close reinsert* é mais eficiente que a de *far reinsert*.

Os valores dos outros gráficos não estão normalizados.

Para facilitar a avaliação dos resultados, cada tipo de operação será discutido separadamente, e ao final serão feitas algumas considerações de cunho geral. Como a R$^+$-tree apresentou um desempenho bem pior que os demais índices na maioria das situações, ela só será citada nos casos em que houver uma exceção.

## 4.4.1 Inserções e remoções

Na inserção dos dados completos de Valinhos (figura 4.2) a R-tree, a R-treeG e a R*-treeSR suplantaram, como era de se esperar, as variantes da R*-tree que usam a rotina de reinserção no tratamento de *overflow*. Entre estas, a que teve o pior desempenho foi a R*-treeOC, cujo número de acessos a disco foi 9,3% superior ao do método de acesso vencedor, a R-treeG. Em compensação, as variantes da R*-tree com reinserção resultaram em estruturas menores (figura 4.3) e, portanto, com melhor ocupação dos nós (figura 4.4). A melhor ocupação foi a da R*-treeCF, com 73,1% e 908 blocos, enquanto a pior foi a da R-tree, com 67,8% e 981 blocos.

Na inclusão do conjunto de postes (figura 4.11) a desvantagem dos métodos de acesso que fazem a reinserção diminuiu bastante, a ponto da R*-treeOC, desta vez a melhor entre elas, praticamente se equiparar às duas melhores, a R*-treeSR e a R-tree. Além disso, a R-treeG apresentou os piores resultados. Isso mostra que na inclusão de pontos a sobrecarga causada pela reinserção de entradas é menor que durante a inclusão de retângulos. A R*-treeCF foi, novamente, a estrutura com menor número de blocos (figura 4.12) e melhor ocupação dos nós (figura 4.13), e os piores números foram os da R*-treeSR.

A inserção do conjunto de quadras (figura 4.14) mostrou uma situação inversa à anterior, ou seja, houve um aumento bastante acentuado nos acessos a disco para as R*-trees que fazem a reinserção com relação às demais. Mais uma vez, a R*-treeCF apresentou o menor número de blocos (figura 4.15) e a maior ocupação (figura 4.16), enquanto a R-treeG obteve os piores resultados.

A remoção dos dados executada na fase 2 fez com que a ocupação dos nós caísse muito em todos os índices (figura 4.7), ficando entre 55,5% (R-treeG) e 58,6% (R*-treeCC). A intercalação de inclusões e exclusões na fase 3 recuperou em parte essa ocupação (figura 4.10), que ficou entre 60,8% (R-tree) e 62,7% (R*-treeCC). Em ambas as fases, o número de acessos a disco durante as atualizações foi menor para as R*-trees com reinserção (figuras 4.5 e 4.8).

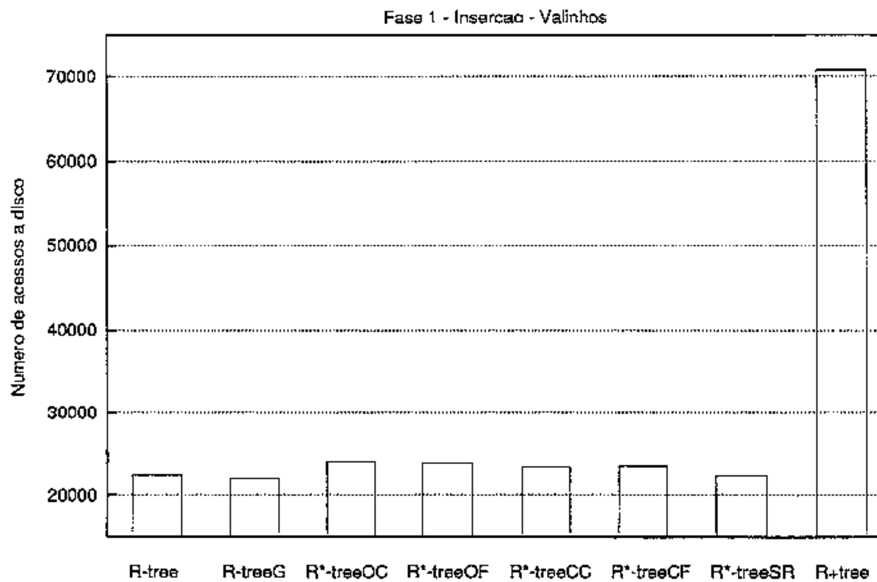



Figura 4.2: Número de páginas acessadas na inserção dos dados completos de Valinhos (66.837 objetos).

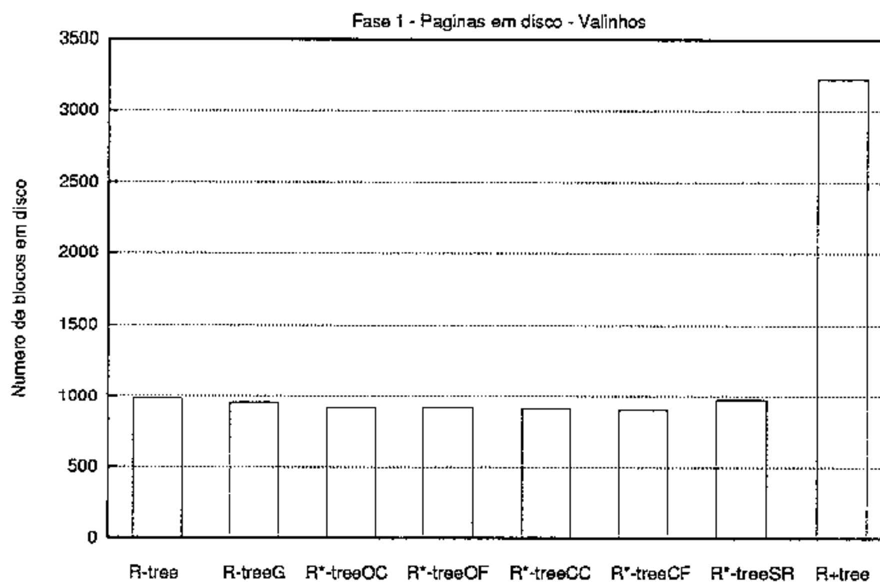



Figura 4.3: Número de páginas armazenadas em disco — fase 1 — dados completos de Valinhos.

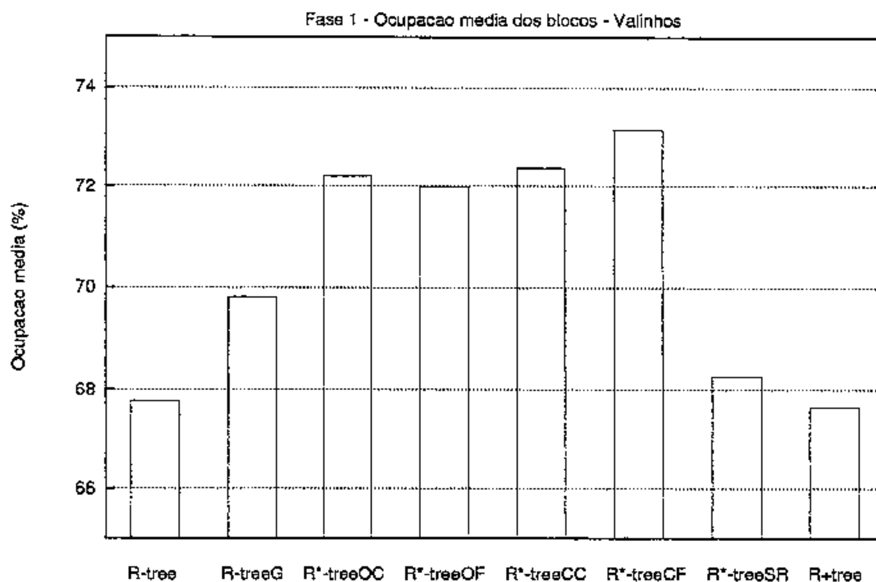



Figura 4.4: Ocupação média dos nós — fase 1 — dados completos de Valinhos.

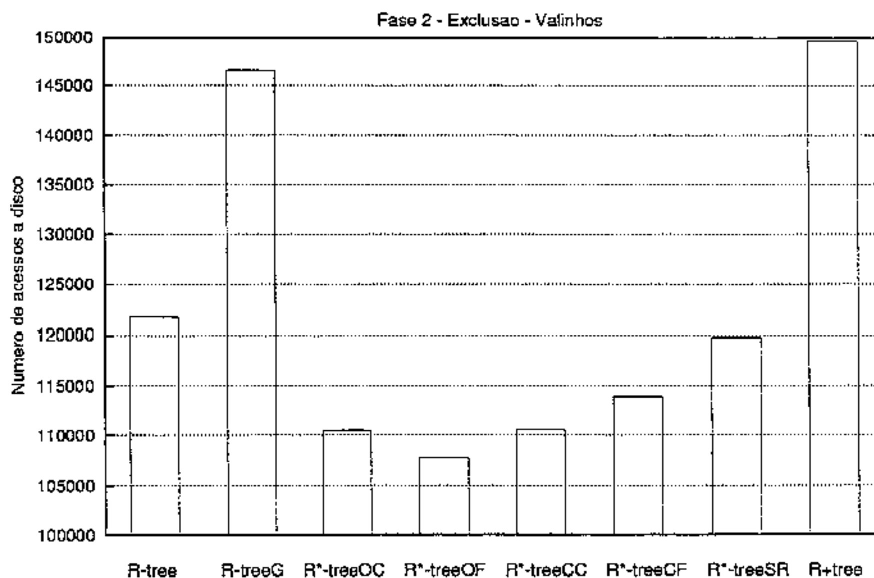



Figura 4.5: Número de páginas acessadas em disco na exclusão de metade dos objetos (33.418) — dados completos de Valinhos.

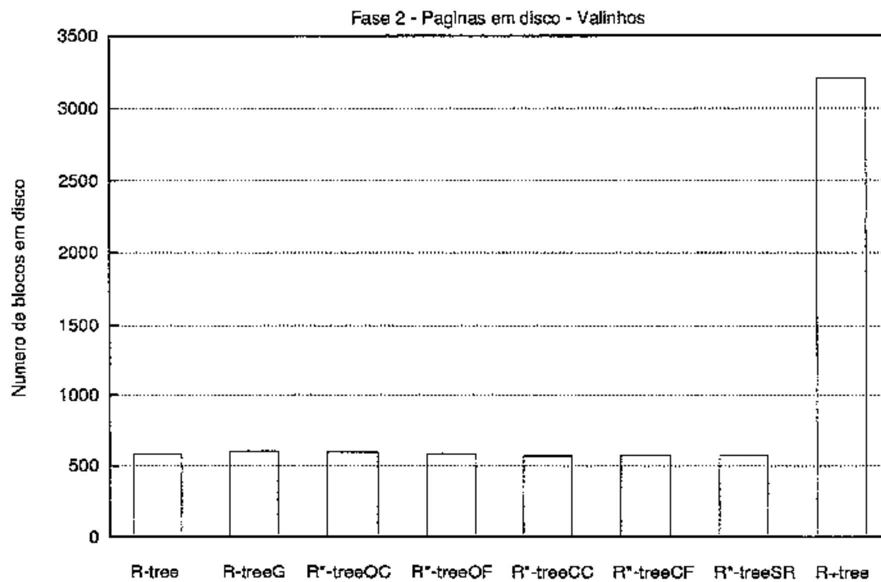



Figura 4.6: Número de páginas armazenadas em disco — fase 2 — dados completos de Valinhos.

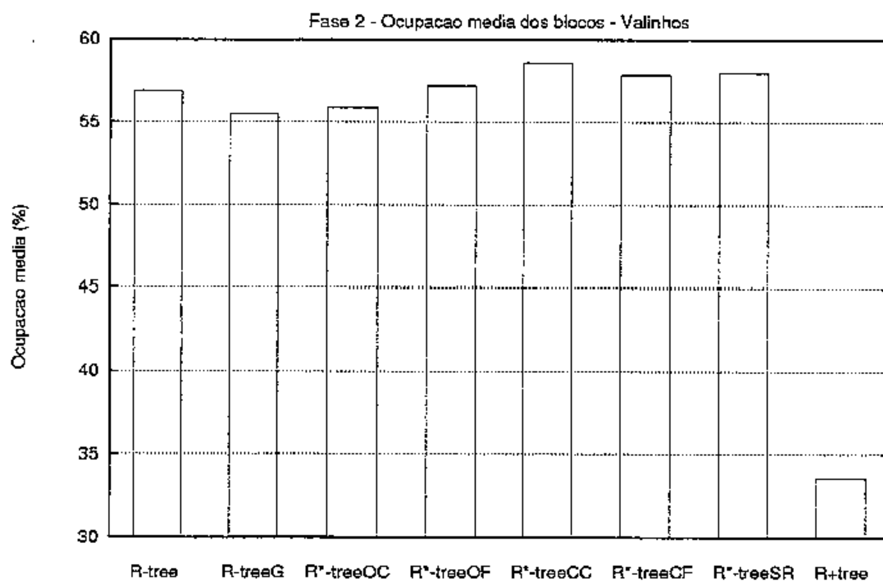



Figura 4.7: Ocupação média dos nós — fase 2 — dados completos de Valinhos.

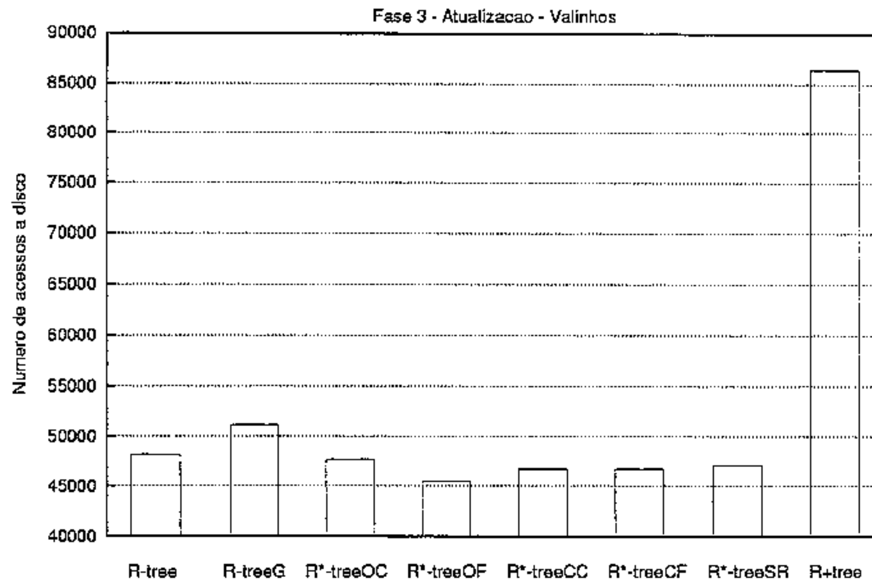



Figura 4.8: Número de páginas acessadas em disco na inclusão de 10.000 objetos e exclusão de outros 10.000 — dados completos de Valinhos.

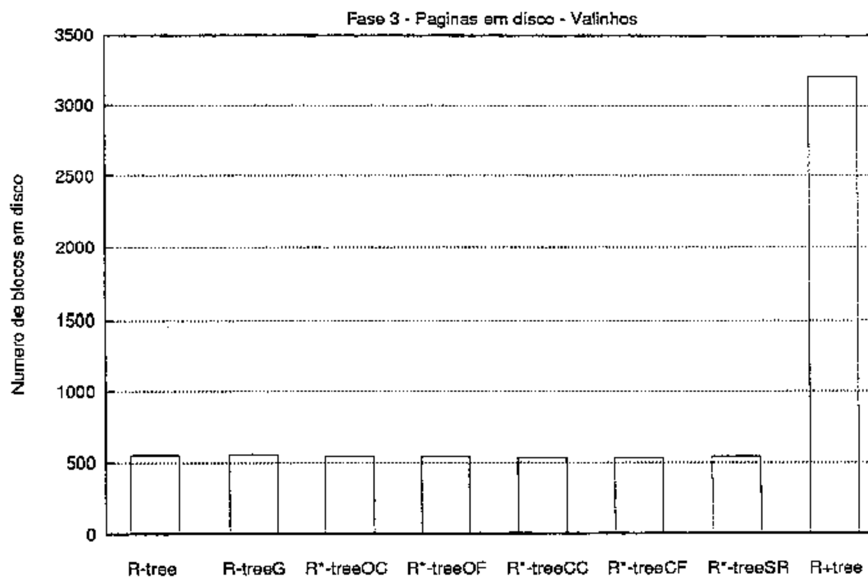



Figura 4.9: Número de páginas armazenadas em disco — fase 3 — dados completos de Valinhos.

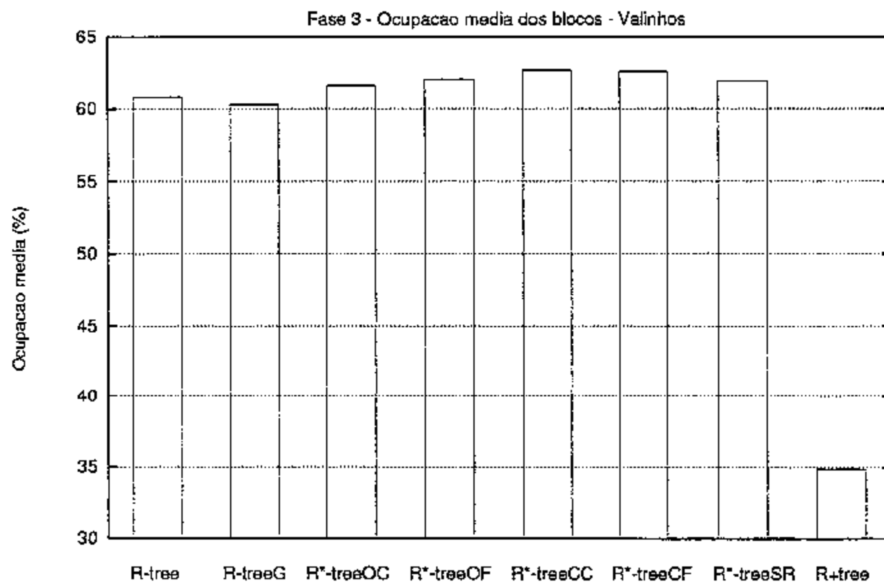



Figura 4.10: Ocupação média dos nós — fase 3 — dados completos de Valinhos.

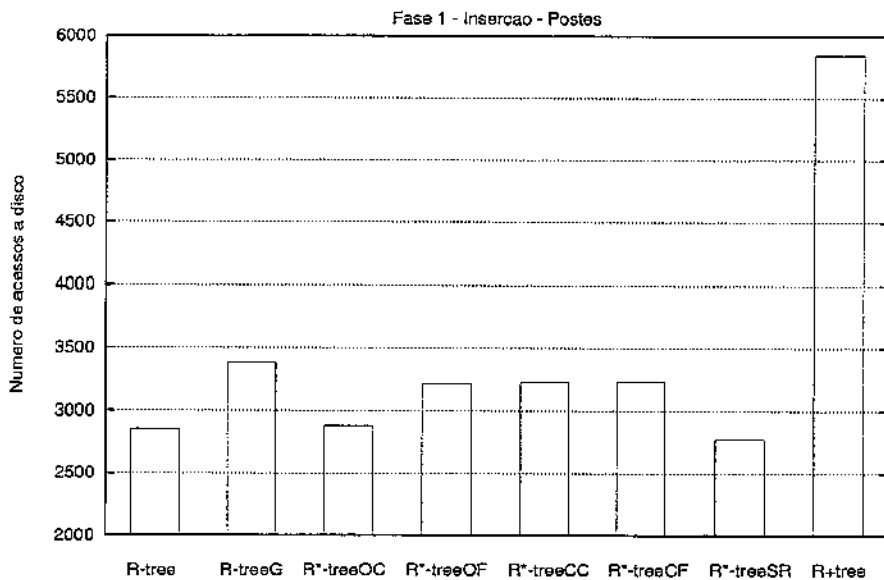



Figura 4.11: Número de páginas acessadas na inserção dos postes (13.813 objetos).

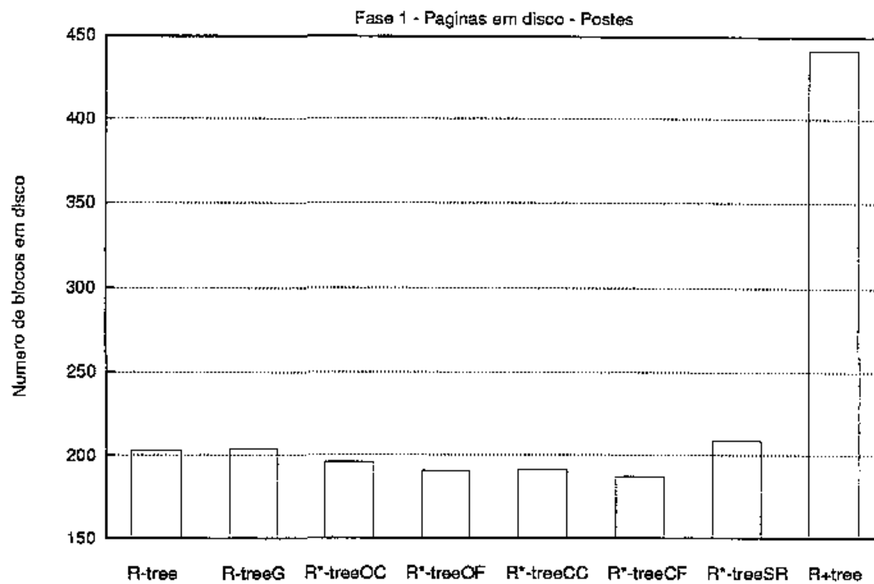



Figura 4.12: Número de páginas armazenadas em disco — fase 1 — postes.

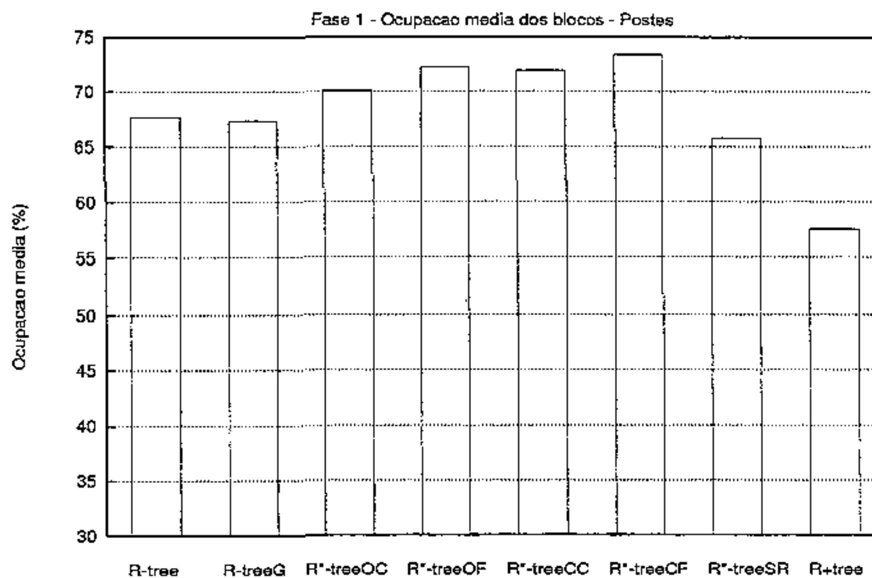



Figura 4.13: Ocupação média dos nós — fase 1 — postes.

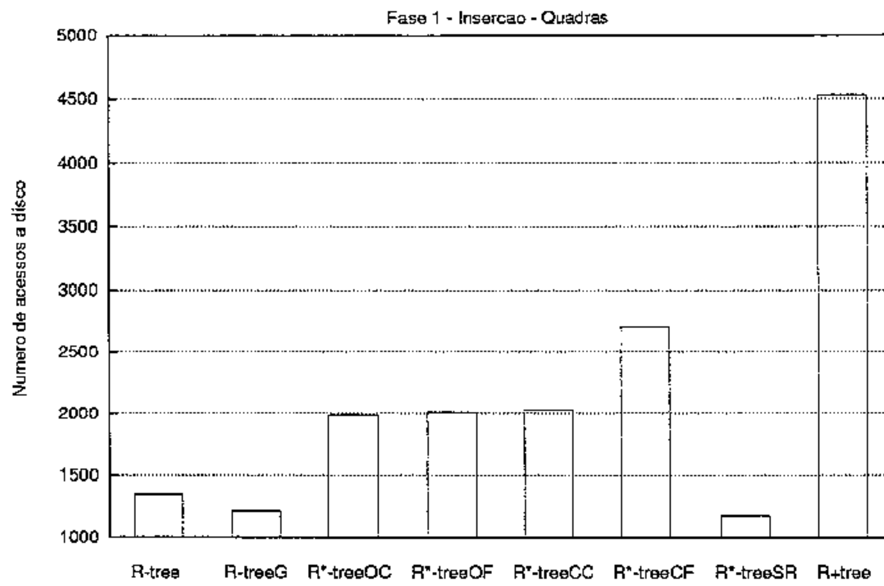



Figura 4.14: Número de páginas acessadas na inserção das quadras (2.473 objetos).

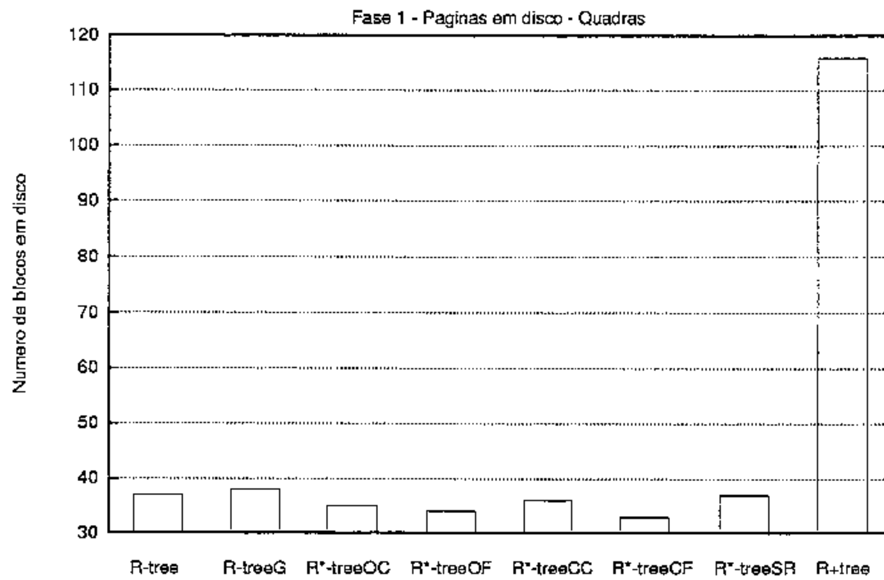



Figura 4.15: Número de páginas armazenadas em disco — fase 1 — quadras.

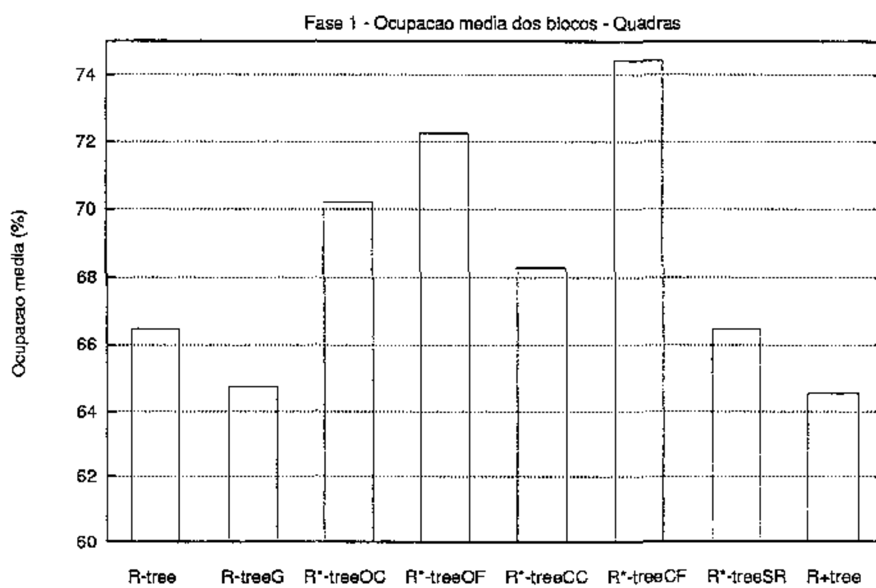



Figura 4.16: Ocupação média dos nós — fase 1 — quadras.

### 4.4.2 *Point queries*

A $R^+$-tree foi o método de acesso que apresentou o melhor desempenho nas *point queries* executadas sobre o conjunto completo dos dados de Valinhos nas três fases (figuras 4.17 a 4.19). Ela acessou de 27,5 a 53,4% menos páginas em disco que sua concorrente mais próxima, que nos três casos foi a $R^*$-treeOC. O que garantiu esse bom desempenho à $R^+$-tree foi o fato de que seu algoritmo só precisa acessar uma página por nível, já que cada ponto de consulta só pode estar incluso no MBR de uma entrada em cada nível. Ao contrário, os algoritmos das outras estruturas freqüentemente precisam acessar mais de um nó por nível, devido às sobreposições nos MBRs das entradas. No entanto, essa eficiência da $R^+$-tree não se repetiu com os conjuntos de postes (figura 4.20) e quadras (figura 4.21), onde ela só venceu a R-tree e a R-treeG. Nas *point queries* executadas sobre o conjunto de postes venceu a $R^*$-treeOF, e a $R^*$-treeCF se comportou melhor com o conjunto de quadras.

Dois fatores contribuíram para o fraco desempenho da $R^+$-tree na indexação do conjunto de postes:

- a baixa sobreposição entre os MBRs das entradas dos nós nas variantes da $R^*$-tree, já que o conjunto de dados era formado exclusivamente por pontos. Isso fez com que o número de páginas requisitadas por consulta em cada variante ficasse também próximo de uma por nível.

- o melhor agrupamento dos dados nas variantes da $R^*$-tree, o que lhes permitiu um melhor uso do cache.

A queda de desempenho da $R^+$-tree nas *point queries* executadas sobre o conjunto de quadras foi ocasionada pelo fato de que nas variantes da $R^*$-tree parte das consultas foi resolvida na raiz, enquanto que na $R^+$-tree as consultas prosseguiram até o nível das folhas.

Em todos os casos, as variantes da $R^*$-tree venceram a R-tree, que por sua vez foi mais eficiente que a R-treeG em todas as situações. O desempenho da $R^*$-treeSR ficou abaixo da maioria das variantes da $R^*$-tree que usam reinserção, embora em alguns casos ela tenha sido melhor que uma ou duas delas. Nas fases 2 e 3 dos testes sobre os dados completos de Valinhos (figuras 4.18 e 4.19, respectivamente) isso ocorreu porque a ocupação dos nós da $R^*$-treeSR sofreu uma queda menor que as das outras variantes da $R^*$-tree.

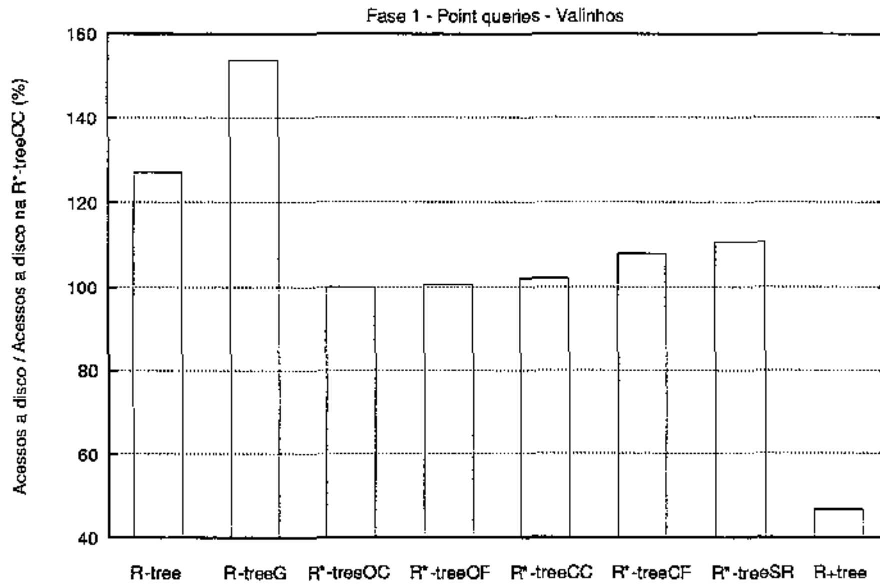



Figura 4.17: *Point queries* — fase 1 — dados completos de Valinhos.

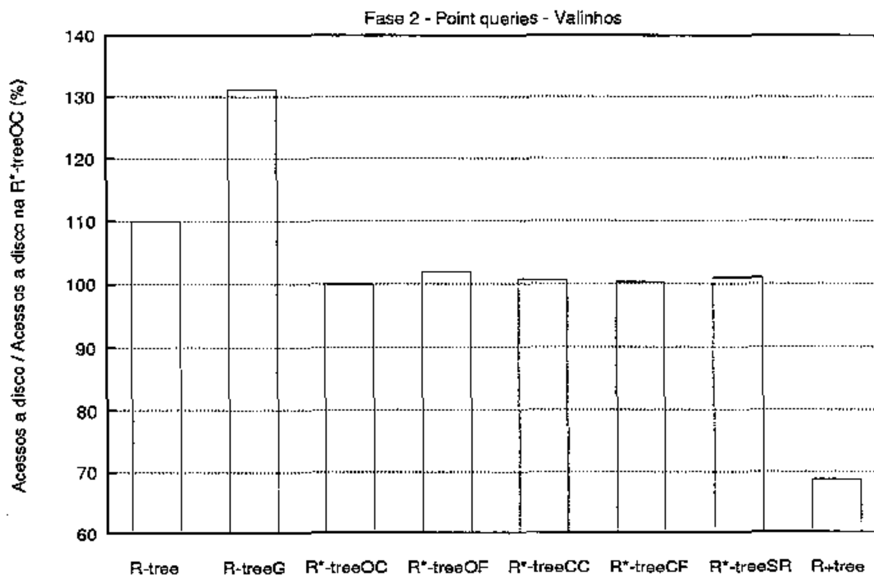



Figura 4.18: *Point queries* — fase 2 — dados completos de Valinhos.

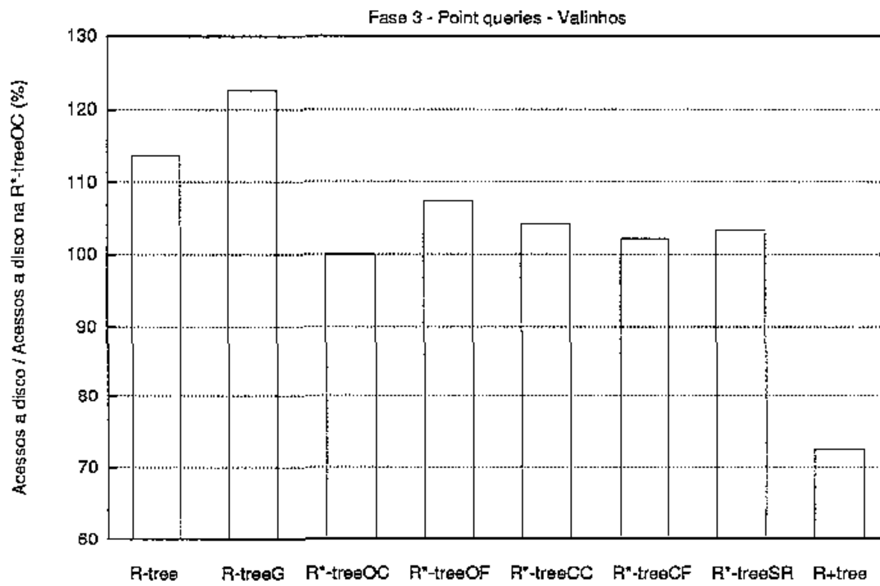



Figura 4.19: *Point queries* — fase 3 — dados completos de Valinhos.

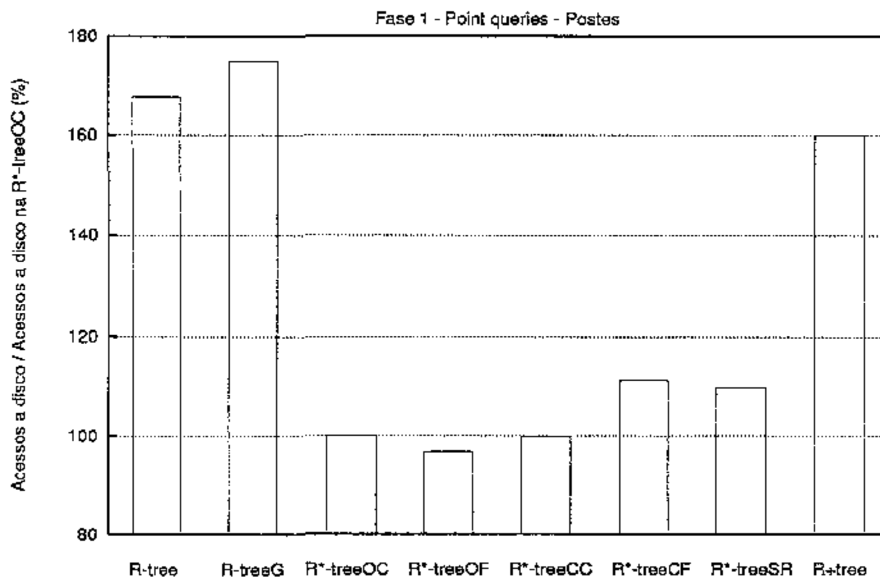



Figura 4.20: *Point queries* — fase 1 — postes.

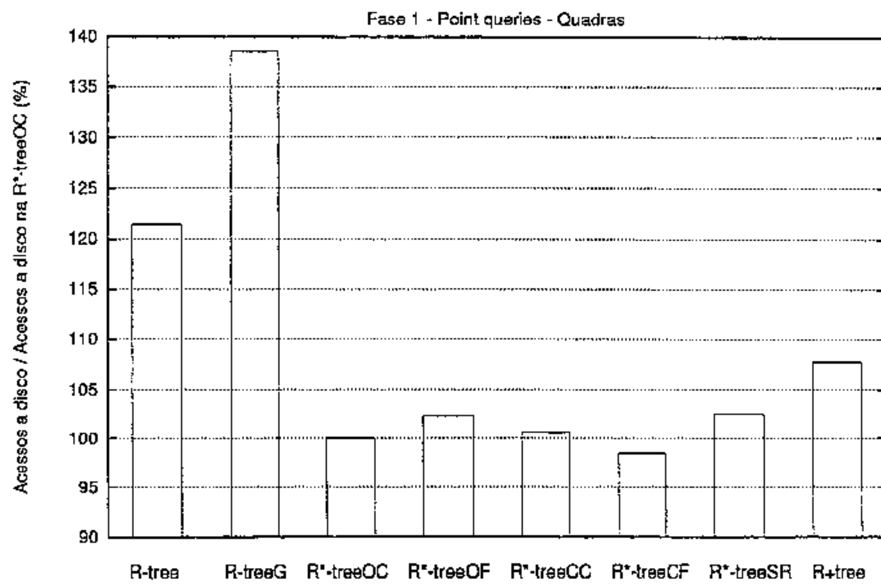



Figura 4.21: *Point queries* — fase 1 — quadras.

### 4.4.3 *Range queries* para determinação de intersecção

Na fase 1, a R$^+$-tree obteve o melhor desempenho em consultas sobre os dados completos de Valinhos (figuras 4.22 a 4.27) para janelas com área menor ou igual a 0.001% da área total, mas para janelas maiores seu número de acessos a disco foi muito maior que o das outras estruturas. Ela também não se comportou bem nas consultas sobre os conjuntos de postes (figuras 4.28 e 4.29) e quadras (figuras 4.30 e 4.31). O melhor desempenho geral nessa fase foi o da R*-treeOF, exceto para o conjunto de quadras, onde ela teve o pior desempenho entre as variantes da R*-tree, e a R*-treeCF venceu.

A exclusão de dados executada na fase 2 afetou de forma mais negativa o desempenho da R*-treeOC que os das outras variantes da R*-tree e o das duas variantes da R-tree, a tal ponto que ela chegou a ser suplantada pela R-tree nas consultas com janelas de 1 e 10% da área total (figura 4.24). O método de acesso com o melhor desempenho nessa fase foi, novamente, a R*-treeOF.

Na fase 3, a R*-treeCF obteve o melhor desempenho geral (figuras 4.26 e 4.27).

Outras observações que podem ser feitas acerca dos gráficos das *range queries* são:

- A R-tree tem um desempenho melhor que a R-treeG. A única exceção se verifica para janelas de 10% da área total em consultas sobre os dados completos de Valinhos na fase 1 (figura 4.22), onde a R-treeG venceu em razão de ter um número menor de nós.

- A R*-treeSR foi mais eficiente que a R-tree em todos os casos.

- As variantes da R*-tree que usam a rotina de reinserção foram melhores que a R*-treeSR na fase 1, exceto para as quadras (figura 4.31), onde esta venceu a R*-treeOF e teve desempenho semelhante à R*-treeCC. Nas fases 2 e 3, onde havia um grande número de exclusões, a R*-treeSR teve um desempenho próximo ao das outras variantes da R*-tree (figuras 4.25 e 4.27).

- As diferenças de desempenho entre as variantes da R*-tree e as variantes da R-tree tendem a cair com o aumento do tamanho da janela de consulta, pois o número de objetos inclusos na janela aumenta em relação ao número daqueles recuperados em função de *false hits*. Isso diminui a importância de um melhor agrupamento dos objetos, como acontece nas variantes da R*-tree, e aumenta a de um agrupamento em um menor número de páginas. No caso extremo, onde a janela de consulta envolve todo o espaço de dados, todas as páginas devem ser recuperadas para qualquer um dos métodos de acesso. Assim, como a diferença do número de páginas armazenadas entre as variantes da R-tree e as da R*-tree não foi tão grande, o aumento das janelas leva a uma diminuição da diferença do número de páginas acessadas.

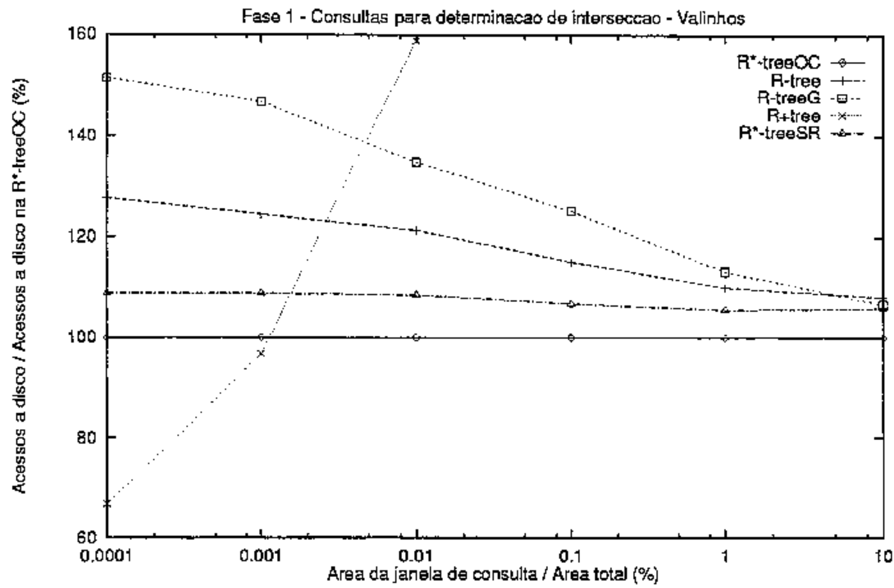



Figura 4.22: *Range queries* para determinação de intersecção — fase 1 — dados completos de Valinhos.

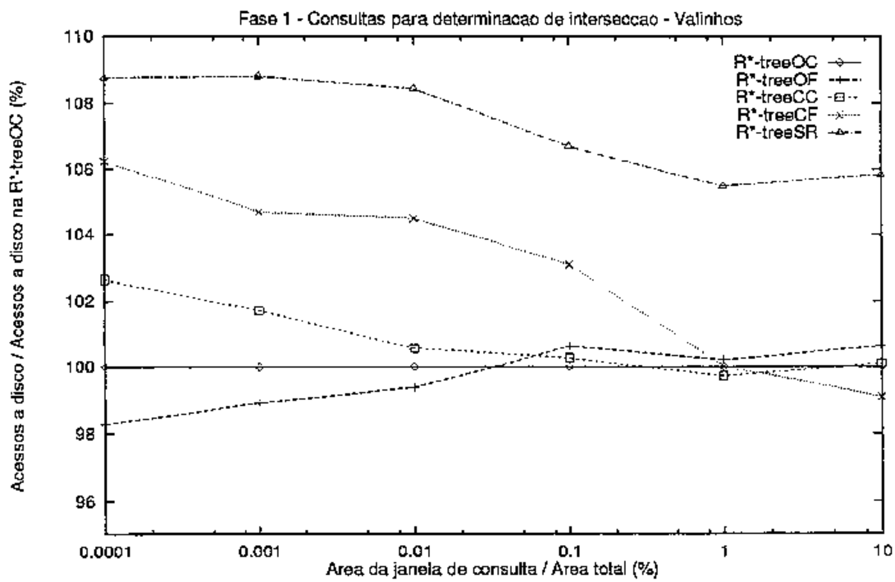



Figura 4.23: *Range queries* para determinação de intersecção (apenas as variantes da R*-tree) — fase 1 — dados completos de Valinhos.

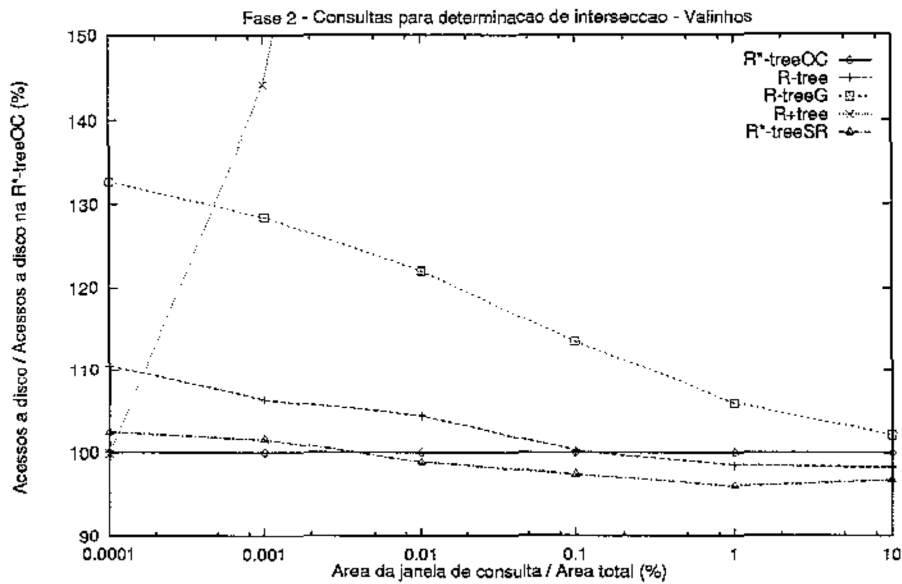



Figura 4.24: *Range queries* para determinação de intersecção — fase 2 — dados completos de Valinhos.

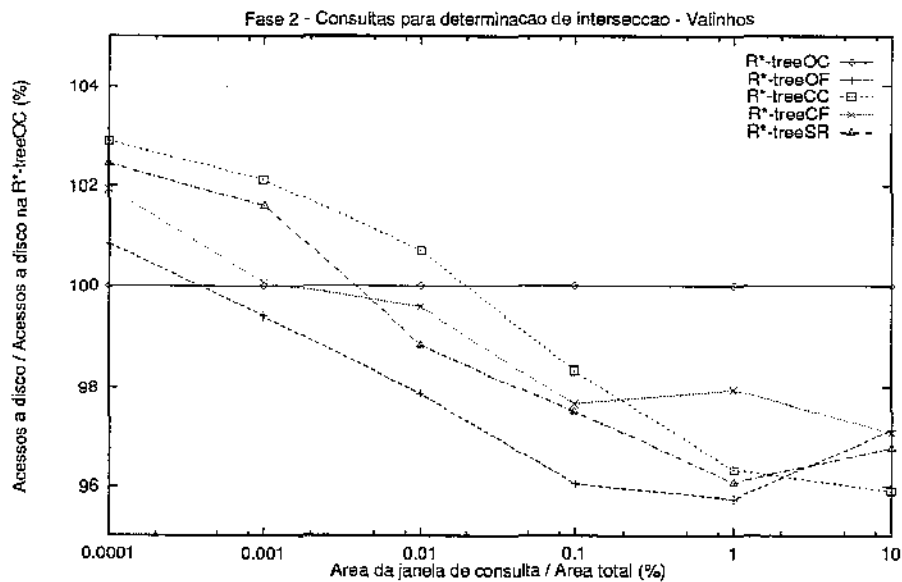



Figura 4.25: *Range queries* para determinação de intersecção (apenas as variantes da R*-tree) — fase 2 — dados completos de Valinhos.

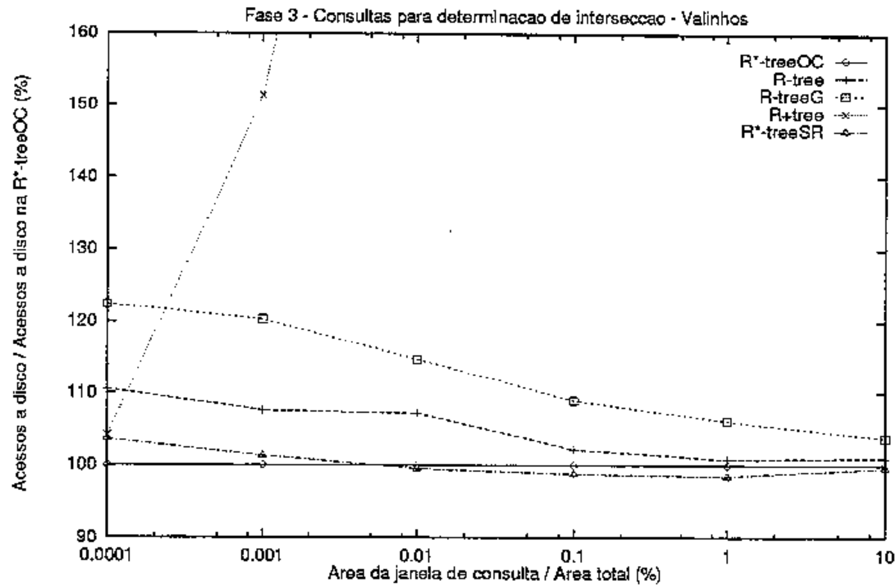



Figura 4.26: *Range queries* para determinação de intersecção — fase 3 — dados completos de Valinhos.

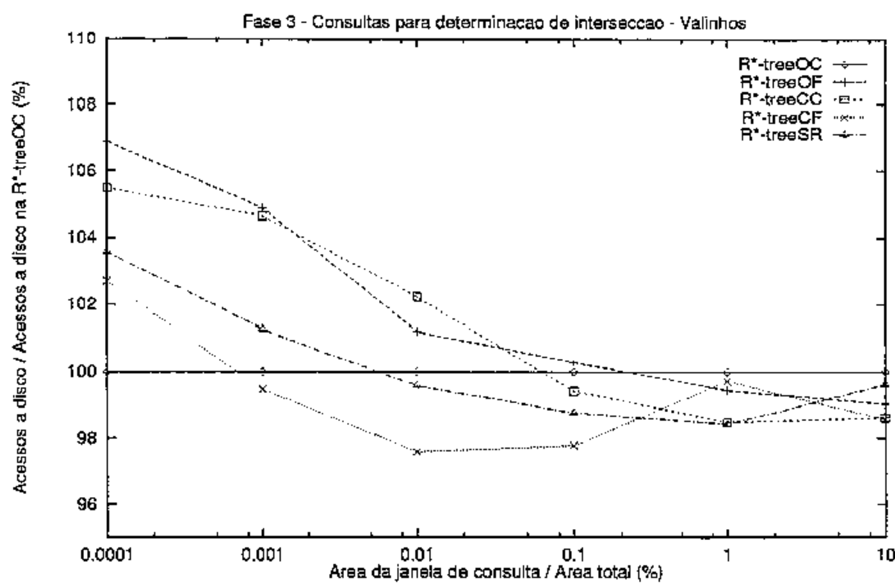



Figura 4.27: *Range queries* para determinação de intersecção (apenas as variantes da R*-tree) — fase 3 — dados completos de Valinhos.

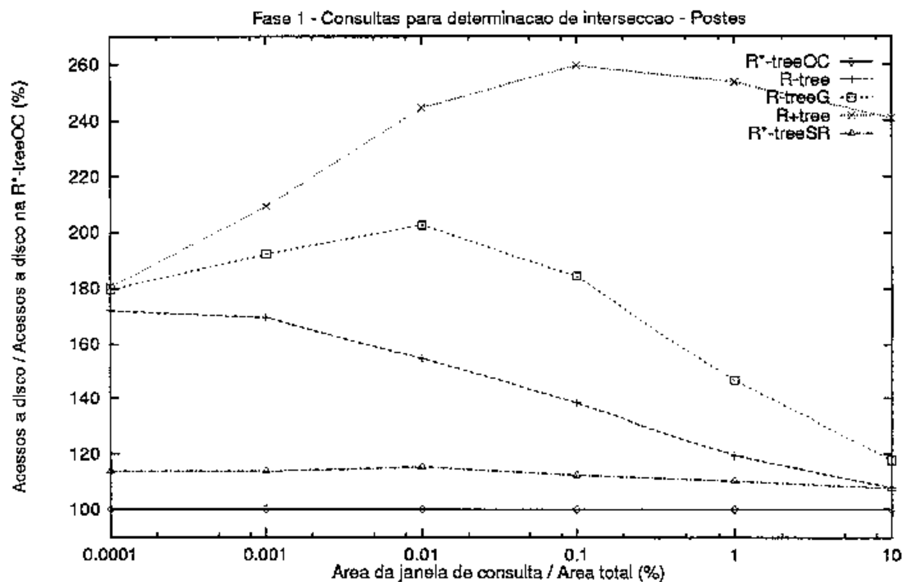



Figura 4.28: *Range queries* para determinação de intersecção — fase 1 — postes.

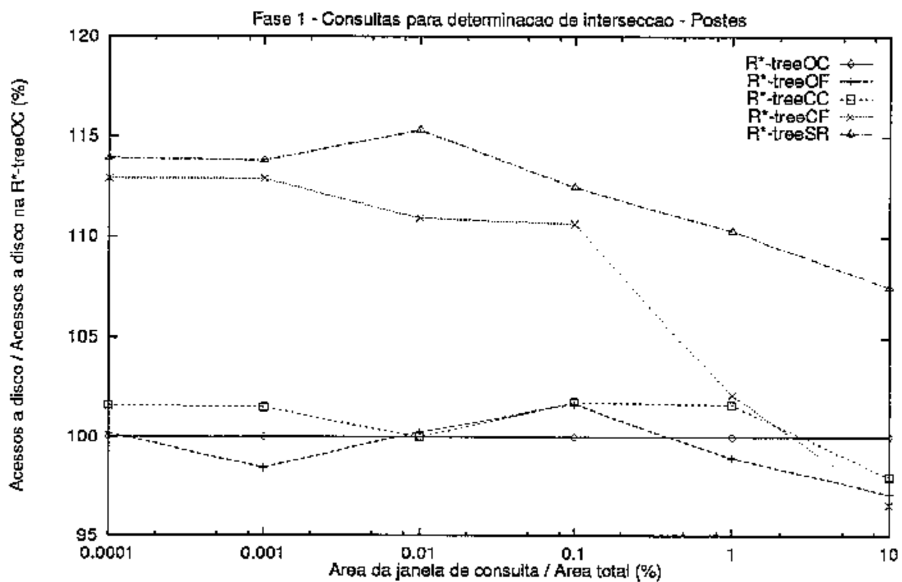



Figura 4.29: *Range queries* para determinação de intersecção (apenas as variantes da R*-tree) — fase 1 — postes.

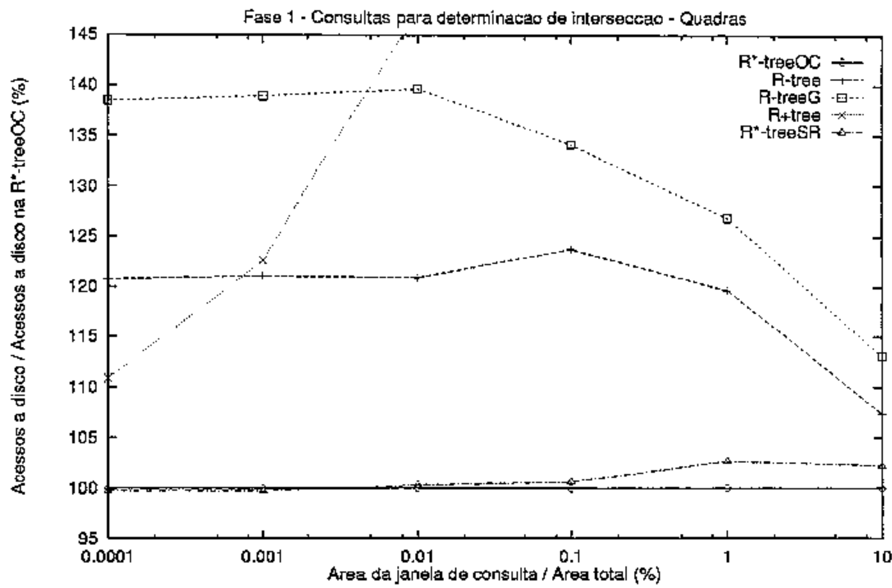



Figura 4.30: *Range queries* para determinação de intersecção — fase 1 — quadras.

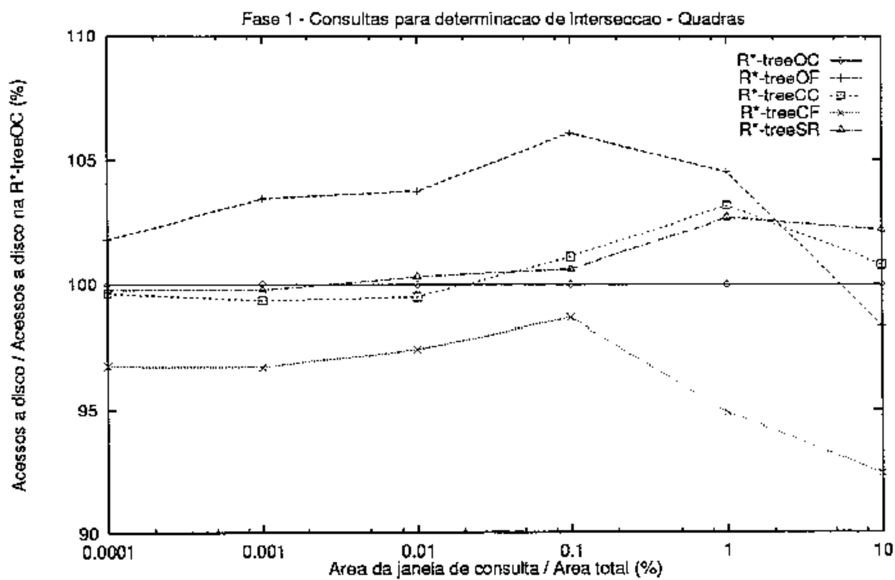



Figura 4.31: *Range queries* para determinação de intersecção (apenas as variantes da R*-tree) — fase 1 — quadras.

### 4.4.4 *Range queries* para determinação dos retângulos que incluem a janela de consulta

Nesse tipo de consulta, a $R^+$-tree obteve o melhor desempenho com janelas de 0,0001% da área total aplicadas sobre os dados completos de Valinhos nas três fases (figuras 4.32 a 4.37), e também com janelas de 0,001% da área total aplicadas sobre os mesmos dados na fase 1 (figuras 4.32 e 4.33). Ela apresentou, ainda, resultados razoáveis com janelas de 0,0001% nas consultas aplicadas sobre o conjunto de quadras (figuras 4.38), e com janelas de 0,001% em consultas sobre os dados completos de Valinhos nas fases 2 e 3 (figuras 4.34 e 4.36). Nos demais casos seu rendimento foi muito ruim, pois à medida que as janelas de consulta aumentam de tamanho, um maior número de consultas é resolvido nos níveis superiores das outras estruturas testadas, enquanto o algoritmo da $R^+$-tree exige que cada consulta prossiga até o nível das folhas.

De modo geral, a R-tree foi melhor que a R-treeG, e a R-treeSR, melhor que a R-tree. As variantes da $R^*$-tree venceram a R-tree e a R-treeG em praticamente todas as situações, mas não há como dizer qual dessas variantes se adaptou melhor a esse tipo de consulta, pois elas se alternaram na liderança dependendo do conjunto de dados, da fase e do tamanho das janelas. No entanto, é interessante destacar o desempenho da $R^*$-treeOF nas consultas executadas sobre os dados completos de Valinhos na fase 1 (figura 4.33).

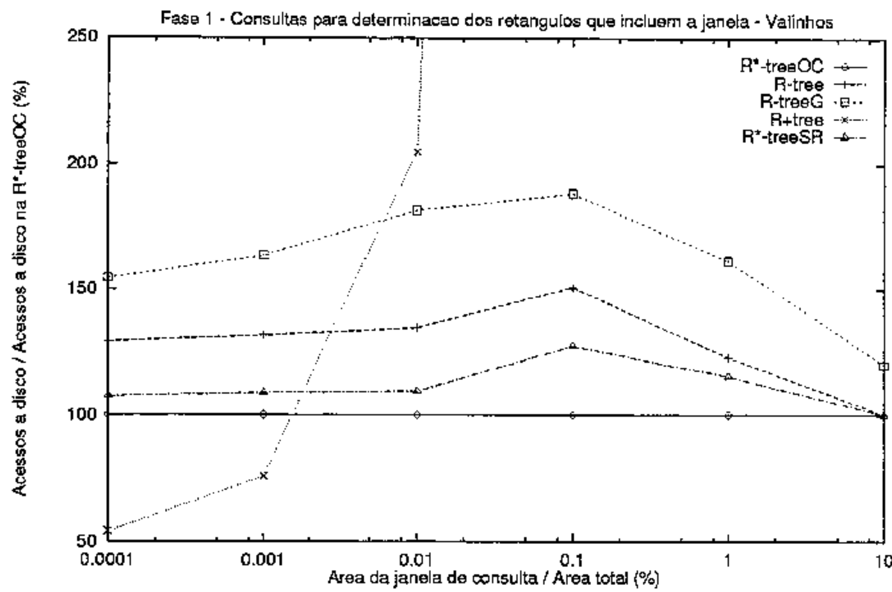



Figura 4.32: *Range queries* para determinação dos retângulos que incluem a janela de consulta — fase 1 — dados completos de Valinhos.

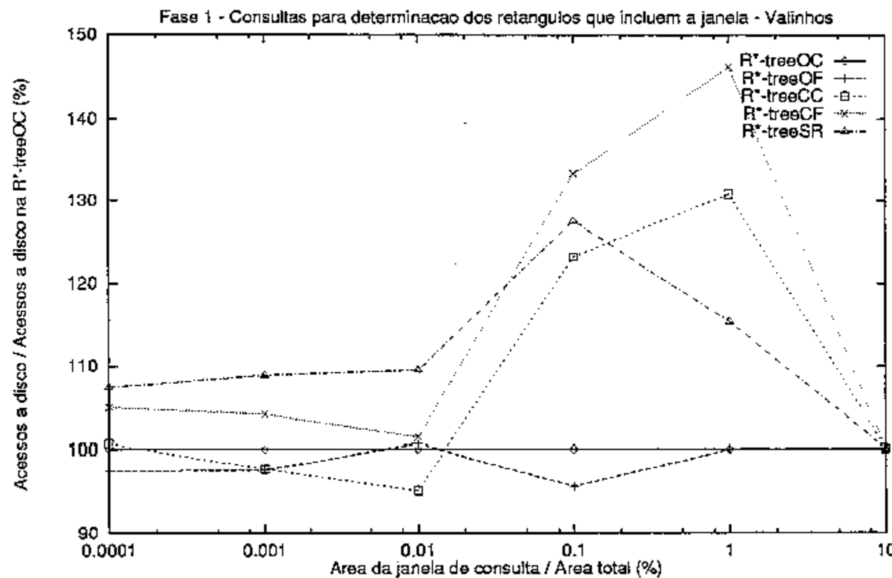



Figura 4.33: *Range queries* para determinação dos retângulos que incluem a janela de consulta (apenas as variantes da R*-tree) — fase 1 — dados completos de Valinhos.

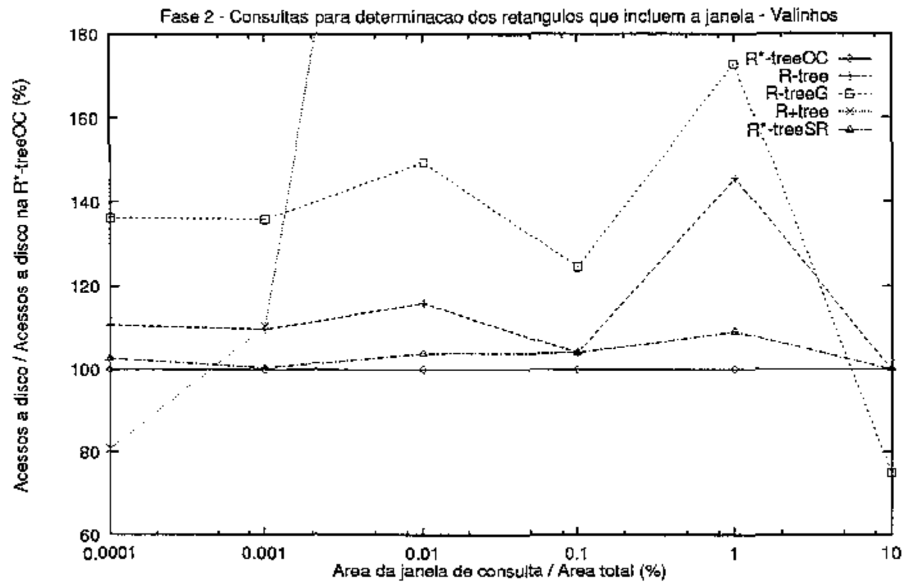



Figura 4.34: *Range queries* para determinação dos retângulos que incluem a janela de consulta — fase 2 — dados completos de Valinhos.

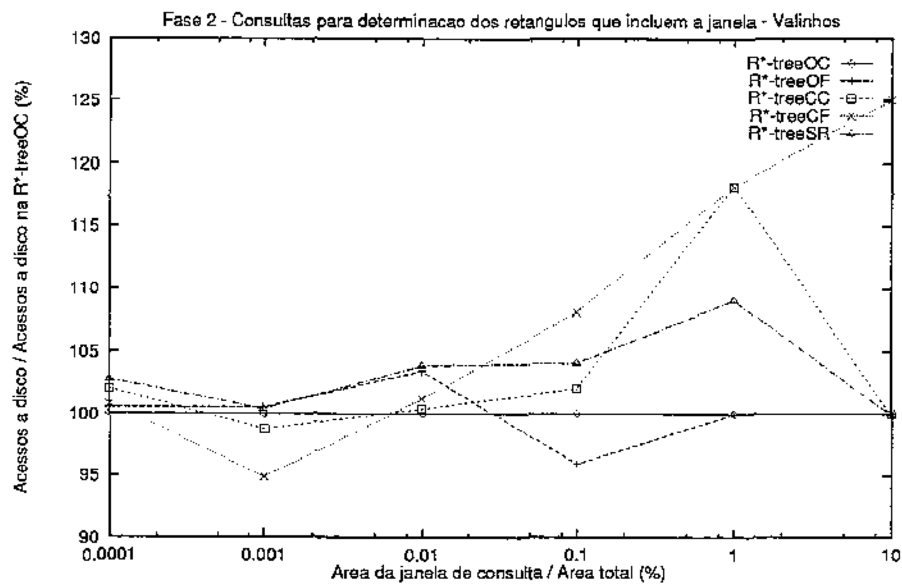



Figura 4.35: *Range queries* para determinação dos retângulos que incluem a janela de consulta (apenas as variantes da R*-tree) — fase 2 — dados completos de Valinhos.

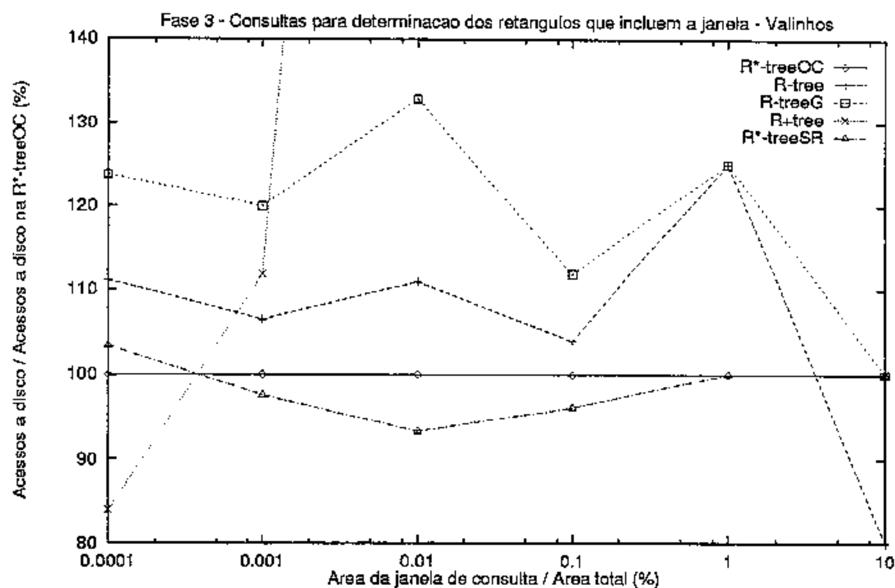



Figura 4.36: *Range queries* para determinação dos retângulos que incluem a janela de consulta — fase 3 — dados completos de Valinhos.

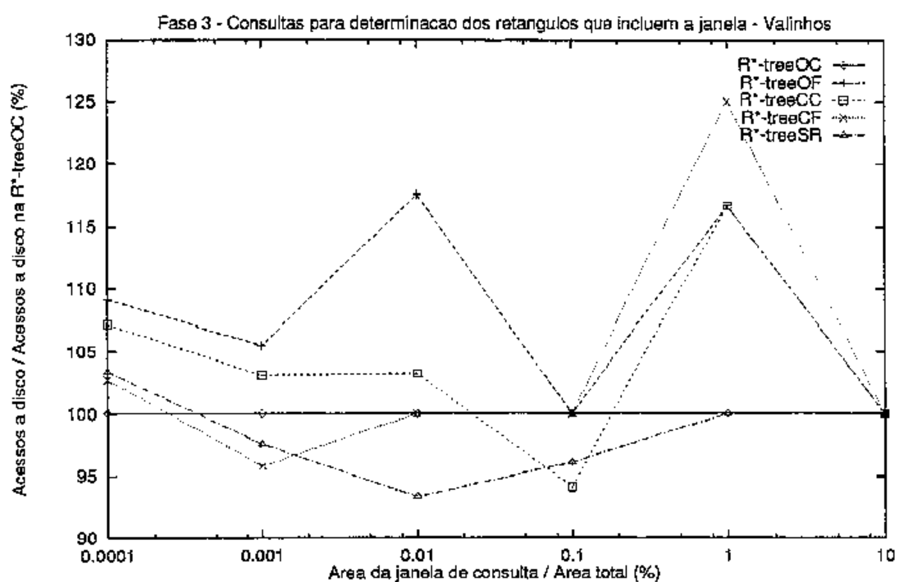



Figura 4.37: *Range queries* para determinação dos retângulos que incluem a janela de consulta (apenas as variantes da R*-tree) — fase 3 — dados completos de Valinhos.

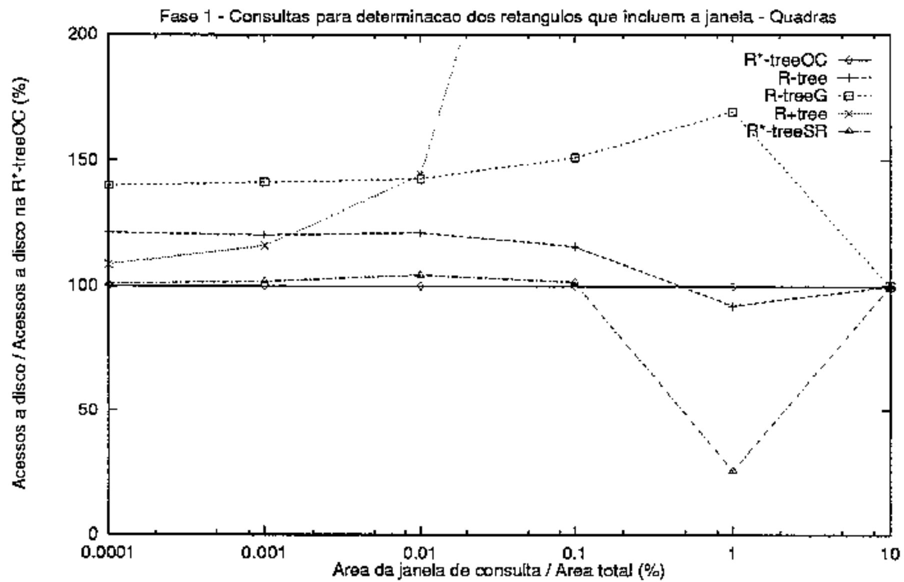



Figura 4.38: *Range queries* para determinação dos retângulos que incluem a janela de consulta — fase 1 — quadras.

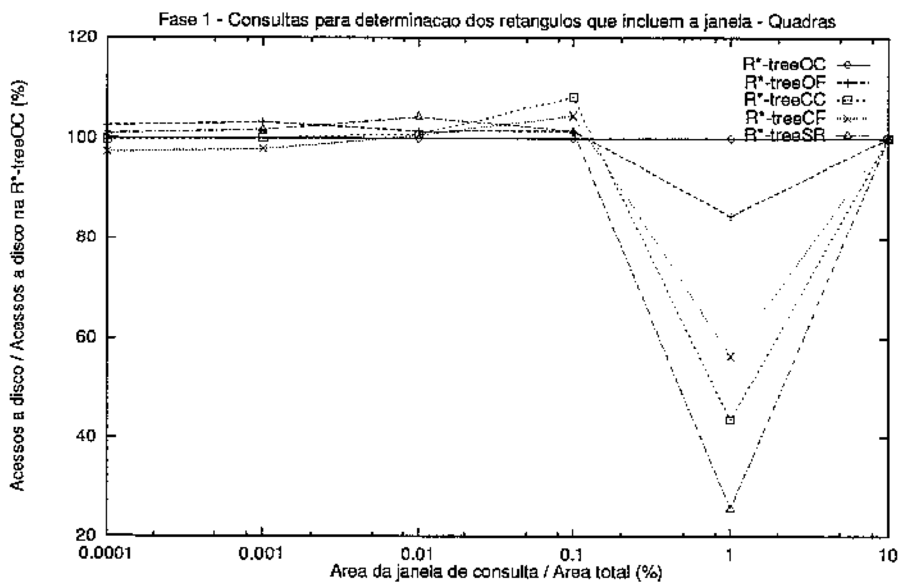



Figura 4.39: *Range queries* para determinação dos retângulos que incluem a janela de consulta (apenas as variantes da R*-tree) — fase 1 — quadras.

### 4.4.5 Conclusões

A $R^*$-treeOF obteve o melhor desempenho para o conjunto completo dos dados de Valinhos nas fases 1 e 2, além de se sobressair também com o conjunto de postes. Ela parece ser a melhor opção entre os métodos de acesso testados para aplicações que trabalhem com conjuntos de dados heterogêneos (quanto aos tipos de objetos representados) que tendam a se expandir, como é o caso da aplicação de onde os dados usados neste trabalho foram extraídos. Já na fase 3, com a presença de um grande número de inclusões e remoções, não houve um vencedor claro. Para se chegar a uma conclusão nesse caso, o ideal seria conhecer a freqüência com que cada tipo de consulta e cada faixa de tamanhos de janela ocorre na carga de trabalho da aplicação (ou tipo de aplicação) em estudo.

A R-tree de Guttman foi mais eficiente que a de Greene em praticamente todas as situações. Da mesma forma, a $R^*$-treeSR venceu a R-tree em quase todos os casos, o que mostra a importância das rotinas para redução do *overlap*. Esse resultado se torna mais interessante porque algoritmos desenvolvidos para a R-tree, como os de controle de concorrência e os de suporte a implementações paralelas, podem ser usados com a $R^*$-treeSR. O mesmo não se pode afirmar para as outras variantes da $R^*$-tree sem antes se verificar qual a influência que a reinserção de entradas no tratamento de *overflow* teria no desempenho desses algoritmos.

## 4.5 Comparação com os resultados de Cox

A comparação dos resultados deste trabalho com os de [Cox91] ficou de certa forma prejudicada devido aos problemas encontrados na implementação das variantes da $R^*$-tree e da $R^+$-tree. Ainda assim, pode-se fazer duas observações:

- Apesar de ter tido um desempenho bastante fraco, a $R^+$-tree conseguiu indexar os dados utilizados. No trabalho de Cox, ela não teve como lidar com 10 dos 14 arquivos que ele usou, devido ao grande número de sobreposições.

- Com relação à R-tree e à R-treeG, que não sofreram alterações, apenas uma divergência foi encontrada[4]: durante a fase 2, na execução de *point queries* sobre um conjunto de pontos, retângulos pequenos e retângulos grandes com distribuição não uniforme (características dos dados de Valinhos), Cox obteve um desempenho da R-treeG ligeiramente melhor que o da R-tree, enquanto que nos experimentos aqui reportados a R-treeG fez 19% mais acessos a disco que a R-tree em *point queries* sobre os dados de Valinhos na fase 2 (figura 4.18). Essa divergência foi causada pelas

[4]Excetuando-se, é claro, as diferenças de valores. Em alguns casos, a discrepância de desempenho entre a R-tree e a R-treeG foram bem maiores nos resultados de Cox que nos apresentados aqui.

diferenças entre os conjuntos de dados utilizados, especialmente na sua distribuição, e demonstra a possibilidade de resultados obtidos a partir de dados sintéticos não se aplicarem a situações que envolvam dados reais.

# Capítulo 5

# Conclusões e extensões

## 5.1 Conclusões

Quando esse trabalho foi iniciado, havia a intenção de se comparar os resultados nele obtidos com os de [Cox91], a fim de se tentar explicar as divergências de suas conclusões com as de outros autores. Caso se pudesse mostrar que o motivo dessas diferenças estava nos tipos de dados usados, se teria uma forte indicação da necessidade do uso dos dados reais para se chegar a resultados mais confiáveis. No entanto, verificou-se que as causas eram, na verdade, falhas na implementação dos métodos de acesso, e isso impossibilitou uma comparação mais efetiva.

Ainda assim, o fato da R*-tree original que utiliza o *far reinsert* em sua rotina de reinserção ter se sobressaído nos testes em detrimento da que utiliza o *close reinsert* (portanto contrariando o que se afirma em [BKSS90]) mostra como testes que utilizam dados reais e testes que utilizam dados sintéticos podem chegar a conclusões discordantes.

Como contribuições deste trabalho, pode-se citar:

- a extensão da classificação dos métodos de acesso espaciais proposta em [Cox91] para contemplar também os métodos que utilizam índices convencionais (capítulo 2);

- o levantamento de métodos de acesso espaciais e técnicas de representação da geometria dos objetos no processo de indexação propostos na literatura recente (capítulo 2);

- a identificação de uma situação para a qual o algoritmo de *split* de García, López e Leutenegger [GLL97] não funciona (capítulo 2);

- a obtenção de um conjunto de dados reais com atributos convencionais e espaciais, e o desenvolvimento de um conjunto de programas que possibilitou sua transcrição

para um formato no qual ele pode ser facilmente disponibilizado para outros pesquisadores (capítulo 3). Esses programas serão usados também pela TELEBRÁS para facilitar o transporte dos dados e a montagem de bancos de dados de teste para o SAGRE.

- a comparação do desempenho de um grupo de métodos de acesso na execução de atualizações e consultas sobre esse conjunto de dados (capítulo 4);
- a identificação e correção de falhas na implementação de alguns desses métodos de acesso, elucidando as causas de resultados obtidos em [Cox91] incompatíveis com os tradicionalmente citados na literatura (capítulo 4);
- a apresentação de um algoritmo para a execução de *range queries* para determinação dos retângulos que incluem uma janela de consulta em uma $R^+$-tree (capítulo 4).

## 5.2 Extensões

Algumas das possíveis extensões dessa pesquisa são:

- a inclusão de outros métodos de acesso da família da R-tree nos testes, como a Hilbert R-tree e as R-trees compactadas. Outra implementação da $R^+$-tree também deveria ser testada, para se verificar até que ponto a implementação usada neste trabalho foi responsável pelo fraco desempenho desse índice nos experimentos.
- a avaliação de métodos de acesso pertencentes a outras classes. Uma comparação entre métodos que usam índices convencionais, como a transformação através de *space filling curves*, a 2dMAP21 e a Filter Tree, seria particularmente interessante, pois os resultados poderiam ser usados mais rapidamente em sistemas gerenciadores de bancos de dados comerciais.
- a verificação do comportamento dos métodos de acesso com a utilização das técnicas de abstração propostas mais recentemente, apresentadas no capítulo 2;
- a inclusão de outros fatores na comparação do desempenho dos métodos de acesso, como tamanho do *cache* e tamanho da página de disco;
- o estudo de políticas de desalocação de páginas de *cache*;
- a avaliação do desempenho dos índices na execução de junções espaciais;

- a caracterização da carga de trabalho de aplicações de gerenciamento de serviços de utilidade pública, isto é, determinar com que freqüência as atualizações, tipos de consulta e tamanhos de janela de consulta aparecem nessa carga de trabalho. Isso não é fácil, pois iria requerer o acompanhamento de mais de uma dessas aplicações já em operação, mas possibilitaria a atribuição de pesos para cada tipo de atualização e consulta na avaliação de testes comparativos de desempenho.

- a obtenção de dados de outras cidades, de preferência maiores e, se possível, provenientes de aplicações diferentes, para a formação de uma base de testes mais abrangente.

# Bibliografia


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

[BK94] T. Brinkhoff and H. P. Kriegel. Approximations for a Multi-Step Processing of Spatial Joins. In J. Nievergelt, T. Roos, H. J. Schek, and P. Widmayer, editors, *IGIS '94*, volume 884 of *Lecture Notes in Computer Science*, pages 25–34. Springer-Verlag Berlin, 1994.

[BKSS90] N. Beckmannm, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, June 1990.

[BKSS94] T. Brinkhoff, H. P. Kriegel, R. Schneider, and B. Seeger. Multi-Step Processing of Spatial Joins. In R. T. Snodgrass and M. Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, volume 23, pages 197–208, May 1994.

[Câm95] G. Câmara. *Modelos, Linguagens e Arquiteturas para Bancos de Dados Geográficos*. PhD thesis, Instituto Nacional de Pesquisas Espaciais, December 1995.

[CCH+96] G. Câmara, M. A. Casanova, A. S. Hemerly, G. C. Magalhães, and C. M. B. Medeiros. *Anatomia de Sistemas de Informação Geográfica*. Escola de Computação. Instituto de Computação, UNICAMP, 1996.

[Cif95] R. R. Ciferri. Um Benchmark Voltado à Análise de Desempenho de Sistemas de Informações Geográficas. Master's thesis, Universidade Estadual de Campinas - UNICAMP, June 1995.

[Cox91] F. S. Cox. Analise de Métodos de Acesso a Dados Espaciais Aplicados a Sistemas Gerenciadores de Banco de Dados. Master's thesis, Universidade Estadual de Campinas, 1991.

[EN94] R. Elmasri and S. Navathe. *Fundamentals of Database Systems.* Addison-Wesley, 2nd edition, 1994.

[Fal86] C. Faloutsos. Multiattribute Hashing Using Gray Codes. In Carlo Zaniolo, editor, *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, pages 227–238, May 1986.

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

[FK94] C. Faloutsos and I. Kamel. Beyond Uniformity and Independence: Analisis of R-trees Using the Concept of Fractal Dimension. Technical report, University of Maryland, May 1994.

[FR89] C. Faloutsos and S. Roseman. Fractals for Secondary Key Retrieval. Technical Report CS-TR-2242, Department of Computer Science, University of Maryland, May 1989.

[FR91] C. Faloutsos and Y. Rong. DOT: A Spatial Access Method Using Fractals. In *Proceedings of the Seventh International Conference on Data Engineering*, pages 152–159, April 1991.

[Fra91] A. U. Frank. Properties of Geographic Data: Requirements for Spatial Access Methods. In O. Günter and H. J. Schek, editors, *Advances in Spatial Databases*, volume 525 of *Lecture Notes in Computer Science*, pages 225–234. Springer-Verlag Berlin, August 1991.

[GG97] Jim Gray and Goetz Graefe. The Five-Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb. *SIGMOD Record*, 26(4):63–68, December 1997.

[GLL97] Y. J. García, M. A. López, and S. Leutenegger. On Optimal Node Splitting for R-trees. Technical Report COMP.97.03, University of Denver, 1997.

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

[Gut84] A. Guttman. R-trees: A Dynamic Index Structure For Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD Conference on Management of Data*, pages 47–57, June 1984.

[IBG] Contagem da População: 1996. Instituto Brasileiro de Geografia e Estatística - IBGE. Available at http://www.ibge.org/geocientifica/geo.htm.

[Jag90] H. V. Jagadish. Linear Clustering of Objects with Multiple Attributes. In H. G. Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 332–342, May 1990.

[KF92] I. Kamel and C. Faloutsos. Parallel R-trees. In Michael Stonebraker, editor, *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, volume 21, pages 195–204, San Diego, California, June 1992.

[KF93] I. Kamel and C. Faloutsos. On Packing R-trees. In B. K. Bhargava, T. W. Finin, and Y. Yesha, editors, *Proceedings of the 2nd International Conference on Information and Knowledge Management (CIKM-93)*, pages 490–499, November 1993.

[KF94] I. Kamel and C. Faloutsos. Hilbert R-tree: An Improved R-tree Using Fractals. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proceedings of the 20th VLDB Conference*, pages 500–509, September 1994.

[KS97] N. Koudas and K. Sevcik. Size Separation Spatial Join. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 324–335, May 1997.

[KSSS89] H. P. Kriegel, M. Schiwietz, R. Schneider, and B. Seeger. Performance Comparison of Point and Spatial Access Methods. In A. P. Buchmann, O. Günter, T. R. Smith, and Y.-F. Wang, editors, *First Symposium on Large Spatial Databases*, volume 409 of *Lecture Notes in Computer Science*, pages 89–114. Springer-Verlag Berlin, July 1989.

[Kum94] Akhil Kumar. G-tree: A New Data Structure for Organizing Multidimensional Data. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):341–347, April 1994.

[LLE97] S. T. Leutenegger, M. A. Lopez, and J. Edgington. STR: A Simple and Efficient Algorithm for R-tree Packing. In *Proceedings of the Thirteenth International Conference on Data Engineering*, pages 497–506, April 1997.

[LLRC96] Y. J. Lee, D. M. Lee, S. J. Ryu, and C. W. Chung. Controlled Decomposition Strategy for Complex Spatial Objects. In R. R. Wagner and H. Thoma, editors, *DEXA '96*, volume 1134 of *Lecture Notes in Computer Science*, pages 207–223. Springer-Verlag Berlin, 1996.

[Mag97] G. C. Magalhães. Telecommunications outside plant management throughout brazil. In *Proceedings of the Twentieth International Conference on Automated Mapping/Facility Management*, pages 385–392, March 1997.

[MCD94] M. R. Mediano, M. A. Casanova, and M. Dreux. V-trees - A Storage Method for Long Vector Data. In *Proceedings of the 20th VLDB Conference*, pages 321–330, 1994.

[MCG96] M. R. Mediano, M. A. Casanova, and M. Gattass. Map-tree: Um Método de Acesso para Mapas Longos. In M. T. P. Vieira and A. J. M. Traina, editors, *Anais do Décimo Primeiro Simpósio Brasileiro de Banco de Dados*, pages 172–186, October 1996.

[Med95] M. R. Mediano. V-trees: Um Método de Armazenamento para Dados Vetoriais Longos. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, 1995.

[MGS+94] G. C. Magalhães, A. Giglioni, C. Santos, D. Teijero, and E. Argondizio. Especificação Técnica de Conversão de Dados Proposta da Telebrás - Projeto Sagre. In Sagres Editora, editor, *Anais GIS Brasil*, pages 43–52, Curitiba, October 1994.

[ND96] M. A. Nascimento and M. H. Dunham. Using Parallel $B^+$trees as a Practical Alternative to the Classical R-tree. In M. T. P. Vieira and A. J. M. Traina, editors, *Anais do Décimo Primeiro Simpósio Brasileiro de Banco de Dados*, pages 187–200, October 1996.

[ND97] M. Nascimento and M. H. Dunham. Using $B^+$trees to Efficiently Process Inclusion Spatial Queries. In *ACM GIS '97 Workshop Proceedings*, pages 3–8, November 1997.

[NDK96] M. A. Nascimento, M. H. Dunham, and V. Kouramajian. A Multiple Tree Mapping-Based Approach for Range Indexing. *Journal of the Brazilian Computer Society*, 2(3), April 1996.

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

[NK93] V. Ng and T. Kameda. Concurrent Acesses to R-trees. In D. J. Abel and B. C. Ooi, editors, *Advances in Spatial Databases, Third International Symposium, SSD'93*, volume 692 of *Lecture Notes in Computer Science*, pages 142–161, Singapore, June 1993. Springer-Verlag Berlin.

[OM84] J. A. Orenstein and T. H. Merret. A Class of Data Structures for Associative Searching. In *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 181–190, April 1984.

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

[OSH93] B. C. Ooi, R. Sacks-Davis, and J. Han. Indexing in Spatial Databases. Unpublished paper, available at http://www.iscs.nus.sg/~ ooibc, 1993.

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

[Sam80] H. Samet. Deletion in Two-Dimensional Quad Trees. *Communications of the ACM*, 23(12):703–710, December 1980.

[Sam95] H. Samet. Spatial Data Structures. In W. Kim, editor, *Modern Database Systems*, pages 361–385. ACM Press, 1995.

[SK91] R. Schneider and H. P. Kriegel. The TR*-tree: A New Representation of Polygonal Objects Supporting Spatial Queries and Operations. In H. Bieri and H. Noltemeier, editors, *Proceedings of the 7th Workshop on Computational Geometry*, volume 553 of *Lecture Notes in Computer Science*, pages 249–263. Springer-Verlag Berlin, March 1991.

[SK96] K. Sevcik and N. Koudas. Filter Trees for Managing Spatial Data Over a Range of Size Granularities. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 16–27, September 1996.

[SLS95] E. Stefanakis, Y. C. Lee, and T. Sellis. The Abstraction Technique for Spatial Access Methods. Technical Report KDBSLAB-TR-95-01, National Technical University of Athens, Greece, February 1995.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The $R^+$-tree: A Dynamic Index For Multi-dimensional Objects. In *Proceedings of the Thirteenth International Conference on Very Large Data Bases*, pages 507–518, September 1987.

[TEL95] Telecomunicações Brasileiras S/A - TELEBRÁS, Centro de Pesquisa e Desenvolvimento - CPqD, Departamento de Sistemas de Operação. *Especificação Técnica da Conversão de Dados*, September 1995. Versão 2.3.1.

[TS93] Y. Theodoridis and T. Sellis. Optimization Issues in R-tree Construction. Technical Report KDBSLAB-TR-93-08, National Technical University of Athens, Athens, Greece, October 1993.

[TS94] Y. Theodoridis and T. Sellis. Optimization Issues in R-tree Construction (extended abstract). In J. Nievergelt, T. Roos, H. J. Schek, and P. Widmayer, editors, *IGIS '94*, volume 884 of *Lecture Notes in Computer Science*, pages 270–273. Springer-Verlag Berlin, 1994.