Operação de carga-rápida (*bulk-loading*) em métodos de acesso métricos[1]

Thiago Galbiatti Vespa
thiago@icmc.usp.br

Orientador:
Prof. Dr. Caetano Traina Júnior
caetano@icmc.usp.br

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC, USP, como parte dos requisitos para a obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Novembro de 2007

[1]Este trabalho tem apoio financeiro do **CNPq** (Conselho Nacional de Desenvolvimento Científico e Tecnológico)

Operação de carga-rápida (*bulk-loading*) em métodos de acesso métricos

Thiago Galbiatti Vespa

"Feliz é o homem que acha sabedoria,
e o homem que adquire entendimento"

Provérbios 3:13 - Bíblia

Dedico esse trabalho primeiramente a Deus, pois sem Ele, nada seria possível. Aos meus pais Jesuino e Izabel que ajudaram na formação do meu caráter e me deram a vida com muito trabalho e amor. À minha irmã e ao meu cunhado pelo apoio. Em especial, à minha esposa e filho, que forneceram condições para que eu concluísse mais uma etapa desta vida.

Thiago Galbiatti Vespa

Agradecimentos

Agradeço ao meu orientador, Prof. Dr. Caetano Traina Júnior, pelos conselhos indispensáveis para minha vida acadêmica, pela contribuição para o desenvolvimento deste trabalho e, principalmente, pela dedicação, empenho, paciência e apoio dedicados à mim.

Aos meus pais pela educação dada a mim e apoio em mais essa etapa da minha vida.

À minha irmã Priscila Galbiatti Vespa Imamura e meu cunhado Wesley Akio Imamura que forneceram recursos e incentivo para a realização deste trabalho.

Em especial, à minha esposa Renata Mayumi Momo pela dedicação e amor despendidos à mim e ao meu filho Daniel Haruyuki Momo Vespa pela compreensão em minha ausência para realização desse trabalho.

À Prof. Dra. Agma Juci Machado Traina pelo apoio e conhecimento oferecido durante esses anos.

Ao William Eidi Imamura e amigos de estadia em São Carlos pela amizade e interesse demostrado em minhas atividades.

Aos meus amigos do GBDI que forneceram conselhos e sugestões para o meu trabalho, em especial, Marcos, Enzo, Ives e Humberto por idéias e auxílio no desenvolvimento.

À todos aqueles que, direta ou indiretamente, colaboraram para que este trabalho conseguisse atingir os objetivos propostos.

Ao CNPq, pelo apoio financeiro à realização desse trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviaturas

AESA	*Approximating Eliminating Search Algorithm.*
CBIR	Recuperação de Imagens Baseada em Conteúdo (*Content-Based Image Retrieval*).
DBM-tree	*Density-Based Metric tree.*
DF-tree	*Distance Fields tree.*
DNA	Ácido Desoxirribonucléico (*Deoxyribonucleic acid*).
kNNQ	Consulta aos k-Vizinhos Mais Próximos (k-*Nearest Neighbors Query*).
LAESA	*Linear Approximating Eliminating Search Algorithm.*
MA	Método de Acesso.
MAMd	Método de Acesso Multidimensional.
MAMdP	Método de Acesso Multidimensional Pontual.
MAMdNP	Método de Acesso Multidimensional Não Pontual.
MAM	Método de Acesso Métrico.
MASI	Método de Acesso Seqüencial Indexado.
MBR	Retângulo Limitante Mínimo (*Minimum Bounding Rectangle*).
MST	Árvore Geradora Mínima (MST - *Minimum Spanning Tree*).
MVP-tree	*Multiple Vantage Point Tree.*
OId	Identificador do objeto (*Object Identity*).
ROT	Relação de Ordem Total.
RQ	Consulta por Abrangência (*Range Query*).
SGBD	Sistema Gerenciador de Base de Dados.
SIG	Sistema de Informação Geográfica.
SQL	Linguagem Estruturada de Consultas (*Structured Query Language*).
VP-tree	*Vantage Point Tree.*

Lista de Símbolos

$\mathbb{N}$	Conjunto dos números naturais.
$\mathbb{V}$	Espaço vetorial.
$\mathbb{F}$	Corpo algébrico.
$\mathbb{R}$	Conjunto dos números reais.
$\mathbb{R}^+$	Conjunto dos números reais não negativos.
a, b, u, v, w	Elementos de um conjunto.
O	Elemento nulo.
1	Elemento unitário.
$\mathbb{S}$	Domínio ou universo de objetos possíveis ou válidos.
$d()$	Função de distância. $d(s_1,\ s_2)$, nos quais $s_1, s_2 \in \mathbb{S}$, representa a distância (similaridade) entre s_1 e s_2.
S	Conjunto (ou base, ou coleção) de elementos.
s_i	Elemento de S. Quando indicado, s_i é usado de forma restrita, representando um objeto de um subconjunto $s_i \in S \subseteq \mathbb{S}$.
$\mathbb{M}$	Espaço métrico definido pelo par $\langle \mathbb{S}, d() \rangle$.
L_p	Métricas de *Minkowski*.
L_{Edit}	Métrica de *Levenshtein*.
$(x_1, \cdots, x_E)$	Vetor.
E	Número de elementos no vetor.
r	Raio.
N_i	Nó pertencente a uma árvore ou sub-árvore
r_q	Raio de busca em uma consulta por abrangência (*Range Query*).
$RQ(s_q,\ r_q)$	Representa uma consulta por abrangência (*Range Query*) cujo o centro é dado por s_q e o raio por r_q.

k	Quantidade de objetos que se deseja recuperar em uma consulta de k-vizinhos mais próximos (*k-Nearest Neighbors Query*).
$kNNQ(o_q, k)$	Representa uma consulta aos k-vizinhos mais próximos (*k-Nearest Neighbors Query*), cujo o centro é dado por o_q e a quantidade de vizinhos que se deseja recuperar é dado por k.
A	Conjunto resposta.
$c_1, \cdots, c_n$	Chaves em uma função de espalhamento.
P_c	Número total de acesso aos nós.
H	Altura da árvore.
N	Número de elementos indexados na árvore.
M	Número de nós.
f_i	*Omni-foci*.
F	Conjunto de amostras
C_M	Capacidade máxima do nó.
C_m	Capacidade mínima do nó.
l	Nível atual da árvore.

VESPA, T. G. *Operação de carga rápida (bulk-loading) em métodos de acesso métrico.* São Carlos, 2007. Dissertação de Mestrado - Instituto de Ciências Matemáticas e de Computação - ICMC, USP.

Resumo

O grau de similaridade entre elementos de dados é o fator primordial para a recuperação de informações em Sistemas Gerenciadores de Bases de Dados que manipulam dados complexos, como seqüências genéticas, séries temporais e dados multimídia (imagens, áudios, vídeos, textos longos). Para responder a essas consultas em um tempo reduzido, faz-se necessário utilizar métodos que usam métricas para avaliar a similaridade entre os elementos. Esses métodos são conhecidos como Métodos de Acesso Métricos. Dentre os mais conhecidos na literatura estão a *M-tree* e a *Slim-tree*. Existem duas maneiras de executar as operações de construção de índices em qualquer método de acesso: inserindo elemento a elemento ou usando a operação de carga-rápida (*bulk-loading*). O primeiro tipo de construção é comum e necessário para todo tipo de método de indexação dinâmico. Já as operações de carga-rápida são utilizadas para conjuntos de dados maiores, como por exemplo, na recuperação de *backups* em bases de dados ou na criação posterior de índices. Nessas situações, a inserção individual tende a ser mais demorada. Realizar uma carga-rápida possibilita a construção de índices com melhor eficiência e em menor tempo, pois há a disponibilidade de todos os dados no instante da criação da estrutura de índices, possibilitando explorar as propriedades do conjunto como um todo. Os Sistemas Gerenciadores de Base de Dados oferecem operações de carga-rápida dos dados nos métodos tradicionais, as quais devem ser supridas também nos Métodos de Acesso Métricos. Neste trabalho, são apresentadas três abordagens, uma técnica para carga-rápida dos dados em Métodos de Acesso Métricos e foi desenvolvido um algoritmo baseado nessa técnica para construir uma *Slim-tree*. Este é o primeiro algoritmo de carga-rápida baseada em amostragem que sempre produz uma *Slim-tree* válida, portanto é o primeiro descrito na literatura que pode ser incluído em um Sistema Gerenciador de Base de Dados. Os experimentos descritos neste trabalho mostram que o algoritmo proposto mantém bom agrupamento dos dados e supera o desempenho dos métodos de inserção seqüencial levando em conta tanto o desempenho de construção quanto à eficiência para realizar consultas.

Abstract

The similarity degree between data elements is the primordial factor for information retrieval in databases that handle complex data, such as genetic sequences, time series and multimedia objects (long images, audio, videos, texts). To answer these queries in a reduced time, it is necessary methods that use metrics to evaluate the similarity between elements. These methods are known as Metric Access Methods. The most known Metric Access Methods in the literature are the *M-tree* and the *Slim-tree*. There are two ways to build index in any access method: inserting element one by one or using the bulk-load operation. The first build type is very common and required for all kinds of dynamic access methods. The bulk-load operations are used for bigger datasets, as for example, in the recovery of backups and re-creation of database indexes. In these situations, the individual insertion takes much time. The bulk-load operation makes it possible to construct indexes more efficiently and faster, because it has the availability of the whole data when the index structure are created, and thus, it is possible to explore the properties of the whole set. Database Management Systems offer bulk-load operations for the traditional methods, so it is important that they can be also supplied for Metric Access Methods. This work presents three bulk-loading approaches and it proposes a technique to bulk-load data into Metric Access Methods. An algorithm based on this technique was developed to construct a *Slim-tree*. This is the first bulk-load algorithm based on sampling that always produces a valid *Slim-tree*, therefore is the first one described in literature that can be enclosed in a Database Management System. The experiments show that this algorithm keeps good clustering of data and in such a way that it surpasses the performance of sequential insertion, taking into account the construction performance and the efficiency to perform queries.

Capítulo

1

Introdução

1.1 Considerações Iniciais

Na tarefa de armazenar e recuperar dados de maneira eficiente, um Sistema Gerenciador de Base de Dados (SGBD) emprega os métodos de indexação como um recurso importante para a localização rápida dos dados armazenados em disco. Inicialmente, esses métodos foram desenvolvidos para atender a dados de tipos numéricos e seqüências de caracteres. Sobre esses dados, existem dois tipos de operadores de comparação: por igualdade ($=$ e $\neq$) e relacionais ($<$, $\leq$, $>$ e $\geq$). Esses seis operadores (dois de igualdade e quatro relacionais) são conhecidos como "*the big six*" da linguagem SQL [Melton et al., 2002]. Os operadores de igualdade podem ser aplicados universalmente a qualquer dado, pois sempre é possível verificar se um dado é igual ou diferente do outro. Já os operadores relacionais são aplicáveis apenas em domínios de dados em que a comparação entre pares de elementos sempre permite decidir qual precede ou sucede o outro. Essa propriedade é conhecida como "Relação de Ordem Total" - ROT.

Contudo, uma nova geração de aplicações tem apresentado como requisito a necessidade de armazenar e recuperar informações mais complexas, como imagens digitais, sons, vídeos e gráficos. A sofisticação desses tipos de dados exige um suporte diferenciado, pois não obedecem a ROT e os operadores de igualdade são de utilidade muito pequena, pois dificilmente existem elementos iguais na base de dados [Faloutsos, 1996].

Para poder proporcionar a busca desses tipos de dados, em tempo hábil, com operações adequadas e diante do volume crescente de informações, faz-se necessário o uso de Métodos de Acesso (MA) que utilizem operações relevantes e adequadas aos dados a serem indexados. Por exemplo, em um Sistema de Informação Geográfica (SIG), toda informação é referenciada espacialmente por suas coordenadas geográficas (norte, sul,

leste, oeste).

Os Métodos de Acesso Multidimensionais (MAMd) preocupam-se em disponibilizar consultas em que existe a noção de dimensão. No entanto, muitos dados, como imagens, séries temporais, textos longos, seqüências genéticas, não apresentam a ROT e nem a noção de dimensão. Conseqüentemente, não é possível realizar consultas utilizando os operadores relacionais e nem consultas topológicas ou cardinais. Por exemplo, em sistemas que utilizam Recuperação de Imagens Baseada em Conteúdo (CBIR), o fator "similaridade" é fundamental. Por isso, os tipos de consultas requisitadas pelas aplicações afetam diretamente o método de acesso utilizado. Não há sentido em considerar que uma imagem está antes de outra, ou que uma seqüência genética é adjacente ou está ao norte de outra. De fato, não é possível ordenar esses elementos, a menos que sejam associados a um atributo não complexo ou em uma ordem que tenha um sentido útil para operações de recuperação, como nome, data, tamanho, etc. Para esses tipos de dados, foram desenvolvidos MA que suportam consultas por similaridade [Faloutsos, 1997]. Os Métodos de Acesso Métricos (MAM) suprem essas consultas. Para isso é utilizada uma função de distância para comparação, em que a proximidade de elementos indica sua similaridade.

Nos MAM, a distância entre pares de elementos é definida por uma função de distância (detalhada melhor no capítulo 2) que respeita as propriedades de simetria, não negatividade e desigualdade triangular, conhecida como métrica. Esta função recebe como parâmetro um par de elementos e retorna um valor não negativo que indica o grau de dissimilaridade entre esses elementos. O domínio desses dados é conhecido como espaço (ou domínio) métrico. Com isso, qualquer dado que possa ser comparado com outro utilizando uma métrica pode ser indexado por um MAM, não necessitando da noção de dimensão ou mesmo da ROT.

Para a indexação de elementos e a construção de qualquer MA existem pelo menos dois tipos de operações de inserção possíveis: elemento a elemento (*tuple-loading*) ou utilizando a operação de carga-rápida (*bulk-loading*). A inserção elemento a elemento é uma operação básica, obrigatória e disponível para qualquer método de acesso utilizado em um SGBD, pois os elementos sempre podem ser inseridos um a um na base de dados. A carga-rápida (detalhada melhor no capítulo 4), embora não seja obrigatória, é utilizada quando tem-se um conjunto significativo de dados e com isso pode-se aproveitar algumas características intrínsecas para criar o índice de forma otimizada e rápida.

Operações como restauração de *backups* em base de dados, criação posterior ou otimização do índice são exemplos do uso de operações de carga-rápida. Por ter todos os dados disponíveis, pode-se explorar as propriedades do conjunto, aproveitando o conhecimento de sua distribuição ou alguma característica fundamental que pode ser significativa para o método de acesso alvo, permitindo acelerar a construção da estrutura e, principalmente, melhorar as operações de busca em MA não tradicionais.

1.2 Objetivos

Uma das operações básicas que os MA tradicionais oferecem aos SGBD é a carga-rápida dos dados, principalmente nas *B-trees* [Comer, 1979] [Johnson & Shasha, 1989] [Lomet, 2001] e suas derivadas, como a B^**-tree* e a B^+*-tree* [Johnson & Shasha, 1993a] [Johnson & Shasha, 1993b] [Chong et al., 2001], amplamente utilizadas em SGBD comerciais. Neste caso, com os dados ordenados, o algoritmo de carga-rápida possui complexidade $\Theta(n)$, onde n é o número de elementos a serem indexados, sendo que para os mesmo elementos indexados com operações de inserção individual, a complexidade é $\Theta(n \cdot \log(n)$ [Bercken & Seeger, 2001]. Esse tipo de operação é conhecido como carga-rápida baseada em ordenação (*sort-based bulk loading*). Os elementos são ordenados de alguma maneira e, então, é construída toda a estrutura. Em adição às técnicas baseadas em ordenação, existem as baseadas em buffer (*buffer-based bulk loading*) e em amostragem (*sample-based bulk loading*).

As operações de carga-rápida baseadas em *buffer* foram desenvolvidas através de técnicas que utilizam áreas de armazenamento temporário para evitar o alto custo de acesso a disco. Uma técnica bastante conhecida na literatura é a *Buffer-tree* [Arge, 2003]. De maneira sucinta, quando os dados não cabem mais em memória, essa técnica pode ser utilizada para manter a eficiência de algoritmos que funcionam apenas em memória principal. A maior desvantagem dessa técnica é que o desempenho depende da ordem de inserção dos dados e seu objetivo é apenas reduzir o custo de acesso ao disco, ignorando o custo de processamento [Ciaccia & Patella, 1998]. Já as operações de carga-rápida baseadas em amostragem escolhem um conjunto de dados (amostras) a partir do conjunto a ser indexado com tamanho suficiente e compatível com o espaço disponível em memória principal e, após isso, tenta construir a estrutura de indexação com as informações obtidas nessas amostras. A vantagem dessa técnica é que os níveis inferiores da árvore podem ser construídos a partir do conjunto de amostras iniciais, criando agrupamentos dos elementos mais próximos, facilitando a consulta por similaridade entre os elementos. Um fator importante dessa técnica é que a qualidade das amostras obtidas pode influenciar no desempenho do algoritmo [Bercken & Seeger, 2001].

Apesar das operações em MA tradicionais estarem bem estabelecidas tanto na literatura quanto no meio comercial, pouca atenção foi despendida para os MA que utilizam dados complexos. Para esses tipos de dados não é recomendado, e às vezes nem possível, aplicar as técnicas de carga-rápida tradicionais baseadas em ordenação. Além disso, o modo de se fazer a carga-rápida em MA não tradicionais possui grande influência na qualidade de busca da estrutura de indexação.

A contribuição principal desse trabalho é um algoritmo de carga-rápida para MAM como a *Slim-tree*. O algoritmo proposto é baseado em amostragem e constrói uma árvore balanceada utilizando o limite controlado de elementos por nó para determinar o número

adequado de elementos em cada etapa do algoritmo.

Os experimentos realizados mostram que esse método de carga-rápida é seis vezes mais rápido na construção em relação a inserção seqüencial e otimiza também o desempenho de consulta, tanto em relação à inserção seqüencial quanto ao algoritmo existente para o MAM *M-tree*.

1.3 Organização do Documento

Este documento apresenta a seguinte organização:

- No **Capítulo 2** são apresentados os conceitos iniciais e fundamentais para o entendimento deste trabalho, tais como as definições de espaço vetorial e métrico, de funções de distância e os conceitos de árvore. Em seguida, são apresentados os dois tipos mais comuns de consultas por similaridade, finalizando com exemplos de aplicações que fazem uso destas consultas;

- No **Capítulo 3** são apresentadas as definições e exemplos dos principais métodos de acesso em domínios convencionais, multidimensionais e métricos encontrados na literatura, destacando os métodos de acesso métricos dinâmicos;

- No **Capítulo 4** são apresentadas as técnicas de operações de carga-rápida existentes e os algoritmos encontrados na literatura;

- No **Capítulo 5** são apresentados os algoritmos de carga-rápida desenvolvidos para o MAM *Slim-tree*, considerando nós com tamanhos fixos, tamanhos fixos por nível e tamanhos controlados para manter a estrutura balanceada.

- No **Capítulo 6** são apresentados os resultados dos experimentos realizados com o algoritmo de carga-rápida para a *Slim-tree*, comparando com a inserção seqüencial e o método de carga-rápida existente para a *M-tree*.

- O **Capítulo 7** conclui o trabalho e apresenta propostas para trabalhos futuros.

No final há ainda dois apêndices auxiliares para um melhor entendimento deste trabalho:

- O **Apêndice A** ilustra o uso da desigualdade triangular para otimização de consultas e mostra como um bom agrupamento dos dados pode otimizar o desempenho em consultas nos métodos de acesso métricos.

- O **Apêndice B** apresenta a arquitetura utilizada para o desenvolvimento do algoritmo de carga-rápida implementada na biblioteca de MAM Arboretum.

Capítulo

2

Conceitos Fundamentais

2.1 Considerações Iniciais

Dados armazenados e indexados em SGBD podem assumir diversos tipos; como exemplo, os MA tradicionais utilizam conjuntos numéricos ou pequenas seqüências de caracteres para a indexação. Já em MAMd, os valores utilizados são multidimensionais como vetores numéricos ou características extraídas de algum tipo de dado complexo. O domínio desses dados é conhecido como Espaço Multidimensional (detalhado na seção 2.2.1). Os dados indexados nesse tipo de domínio precisam ter uma dimensão definida, o que inviabiliza a indexação de dados adimensionais.

Em aplicações tradicionais, é comum a inserção de descrições textuais associadas a dados complexos para indexação e utilização em consultas. Entretanto, essa forma de indexação pode levar a erros oriundos da interpretação de cada usuário na inserção da descrição textual e algumas características dos dados complexos podem passar desapercebidas. Para evitar esses problemas, pode-se utilizar métodos automáticos para extração das características dos dados. Por exemplo, em domínios de imagens, são extraídos atributos como descritores de formas, texturas e cor para composição de vetores de características. Utilizam-se esses vetores para comparação entre as imagens [Felipe et al., 2005].

O número de informações a serem extraídas de dados de um determinado tipo pode ser extremamente grande, prejudicando o desempenho da estrutura [Gaede & Günther, 1998]. Além disso, o conceito de domínio espacial não pode ser aplicado a todos os tipos de dados complexos. Para resolver essas deficiências, foram criados os MAM, que são métodos que utilizam o conceito de "similaridade". Consultas utilizando essa abordagem (detalhadas na seção 2.4), podem trabalhar tanto com vetores de dimensão definida, quanto com dados adimensionais. O que permite essa flexibilidade no

tratamento dos dados são as propriedades do Espaço Métrico, que serão descritas na seção 2.2.2.

A seguir serão descritos os domínios de dados: convencional, multidimensional e métrico. Após isso, serão definidos os conceitos de árvores e consultas por similaridade, finalizando com exemplos de aplicações de consultas por similaridade em domínios métricos.

2.2 Domínio de Dados

O termo espaço ou domínio de dados é utilizado aqui como o conjunto de todos os valores possíveis que um elemento do domínio pode assumir. Os SGBD tradicionais foram desenvolvidos para utilizar dados que estão em domínios com ROT. Domínios que obedecem a ROT são amplamente utilizados e podem definir relações de precedência de um elemento de dado sobre outro diferente. Como exemplo, existem os domínios numérico e lexicográfico (pequenas seqüências de caracteres).

Os MA hierárquicos mais importantes em domínios que obedecem a ROT são as *B-Trees* e suas variantes, como a B^**-tree* e a B^+*-tree*, que serão detalhadas melhor na seção 3.2. Além disso, existem muitas outras estruturas, como índices simples (densos e esparsos) e tabelas de espalhamento (*hash*), que são métodos não hierárquicos e que, normalmente, são utilizados para indexar dados em domínios que obedecem a ROT. Contudo, existem domínios que não obedecem a ROT. As duas próximas seções descrevem dois domínios importantes que não atendem à ROT: o espaço multidimensional e o espaço métrico.

2.2.1 Espaço Multidimensional

O espaço multidimensional é o domínio utilizado pela grande maioria dos MA para indexar elementos com dimensão fixa. Esses elementos pode ser obtidos a partir de extração de características de um dado complexo, como por exemplo, histogramas de cores de imagens [Aslandogan & Yu, 1999], ou posições relativas no espaço.

Um exemplo de espaço multidimensional é o espaço vetorial, na qual os valores de todas as dimensões são elementos de um único domínio simples e as dimensões podem ser indexadas por uma seqüência em $\mathbb{N}$. Um espaço vetorial $\mathbb{V}$ é composto por um corpo $\mathbb{F}$, como os números reais ($\mathbb{R}$), uma operação de adição vetorial de $\mathbb{V} \times \mathbb{V} \rightarrow \mathbb{V}$, representada aqui por "+", e uma operação de multiplicação por escalar de $\mathbb{F} \times \mathbb{V} \rightarrow \mathbb{V}$, representada por "$\cdot$". Além disso, é necessário que oito propriedades sejam satisfeitas para todo elemento do espaço vetorial [Lima, 1997]. Essas propriedades são:

1. **propriedade associativa da adição**: $(u + v) + w = u + (v + w)$ para todo $u, v, w \in \mathbb{V}$;

2. **propriedade comutativa**: $u + v = v + u$ para todo $u, v \in \mathbb{V}$;
3. **existência do elemento nulo**: há um elemento O de $\mathbb{V}$, tal que $u + O = u$ para todo $u \in \mathbb{V}$;
4. **existência de negativo**: para todo elemento v de $\mathbb{V}$ há um elemento u tal que $v + u = O$;
5. **propriedade associativa da multiplicação**: $a \cdot (b \cdot u) = (a \cdot b) \cdot u$ para $a, b \in \mathbb{F}$ e $u \in \mathbb{V}$;
6. **existência da unidade**: se 1 é a unidade de $\mathbb{F}$, $1 \cdot u = u$ para $u \in \mathbb{V}$;
7. **propriedade distributiva I**: $a \cdot (u + v) = a \cdot u + a \cdot v$ para $a \in \mathbb{F}$ e u,v $\in \mathbb{V}$;
8. **propriedade distributiva II**: $(a + b) \cdot u = a \cdot u + b \cdot u$ para $a, b \in \mathbb{F}$ e $u \in \mathbb{V}$;

A maioria dos dados complexos não obedece essas propriedades. Além disso, alguns dados são ditos adimensionais, pois não podem ser descritos por vetores de coordenadas em eixos ortogonais.

Uma aplicação bastante utilizada no espaço vetorial é o de coordenadas georeferenciadas, pois fornece a noção de dimensão espacial e, por isso, é possível utilizar propriedades geométricas tais como ângulos, áreas, entre outros. Note-se que nem todo espaço multidimensional é um espaço vetorial. Por exemplo, o domínio dos histogramas de cores de imagens não formam um espaço vetorial, mesmo considerando que todas as imagens tenham a mesma profundidade de cores, ou seja, todos os histogramas tenham a mesma dimensionalidade. Uma propriedade fácil de ser percebida que não é atendida é a de elemento negativo. Como histogramas são baseados em contagem, nenhuma dimensão pode ter valores negativos, assim não existem dois histogramas $v, u \in \mathbb{V} | v + u = O$. Um dos problemas de se utilizar somente o espaço multidimensional é a impossibilidade de se trabalhar com dados adimensionais.

2.2.2 Espaço Métrico

Uma característica dos espaços métricos é a possibilidade de englobar os espaços multidimensionais (detalhados na seção anterior), bem como espaços adimensionais onde os elementos não podem ser descritos por coordenadas ortogonais, como conjuntos de imagens, palavras, sons ou cadeias de DNA, desde que para isto haja uma função de distância métrica definida.

Seja a função $d()$:

$$d : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}^{+} \tag{2.1}$$

onde, $\mathbb{S}$ é um domínio de elementos possíveis ou elementos válidos. Se $d()$ satisfaz às seguintes propriedades:

1. **simetria**: $d(s_1, s_2) = d(s_2, s_1)$;

2. **não negatividade**: $0 < d(s_1, s_2) < \infty$ se $s_1 \neq s_2$ e $d(s_1, s_1) = 0$;

3. **desigualdade triangular**: $d(s_1, s_2) \leq d(s_1, s_3) + d(s_3, s_2)$;

onde $s_1, s_2, s_3 \in \mathbb{S}$, então espaço $\mathbb{M}$ definido pelo par $\langle \mathbb{S}, d() \rangle$ é definido como Espaço Métrico e a função $d()$ é chamada de Métrica ou Função de Distância Métrica [Munkres, 2000]. Essa função define valores correspondentes à medida de "distância", também chamada de "dissimilaridade", entre dois elementos. Isso significa que, quanto menor a distância entre dois elementos, mais próximos ou semelhantes serão [Chávez et al., 2001].

As funções mais conhecidas são as métricas de *Minkowski*, conhecida também como família (L_p) [Wilson & Martinez, 1997] (detalhadas na seção 2.2.3) e, para similaridade entre seqüências de caracteres, a métrica de *Levenshtein* - (L_{Edit}). A função de distância L_{Edit} retorna o número mínimo de operações de edições (substituições de caracteres, inserções e remoções) necessárias para transformar uma seqüência de caracteres na outra. No entanto, qualquer função que satisfaça às propriedades de uma métrica pode ser utilizada, funcionando como uma "caixa preta" para o método de acesso métrico, geralmente definida por um especialista.

2.2.3 Métricas de *Minkowski*

Vetores de valores numéricos podem ser comparados utilizando diversas funções. Entre as mais conhecidas estão as métricas da família *Minkowski* (L_p) [Wilson & Martinez, 1997]. Essa família pode ser definida da seguinte forma:

$$L_p((x_1, \cdots, x_E), (y_1, \cdots, y_E)) = \left(\sum_{i=1}^{E} |x_i - y_i|^p \right)^{1/p} \tag{2.2}$$

Nessa família de funções, existem três métricas que são bastante utilizadas: a L_1, a L_2 e a L_∞ (Figura 2.1). A L_1 é também conhecida como distância de bloco ou *Manhattan* e corresponde ao somatório dos módulos das diferenças entre as coordenadas dos vetores. A L_2 é a métrica mais usual para cálculos de distâncias entre vetores, conhecida como *Euclidiana*. Já, a L_1 ou *Chebychev* é obtida com o cálculo do limite da equação 2.2 quando p tende ao infinito. Com isso, tem-se a seguinte equação:

$$L_\infty((x_1, \cdots, x_E), (y_1, \cdots, y_E)) = \lim_{p \to \infty} \left(\sum_{i=1}^{E} |x_i - y_i|^p \right)^{1/p} = \max_{i=1}^{n} |x_i - y_i| \tag{2.3}$$

A família L_p é amplamente utilizada em espaços multidimensionais. Com uma dessas métricas, pode-se utilizar elementos multidimensionais em espaços métricos.

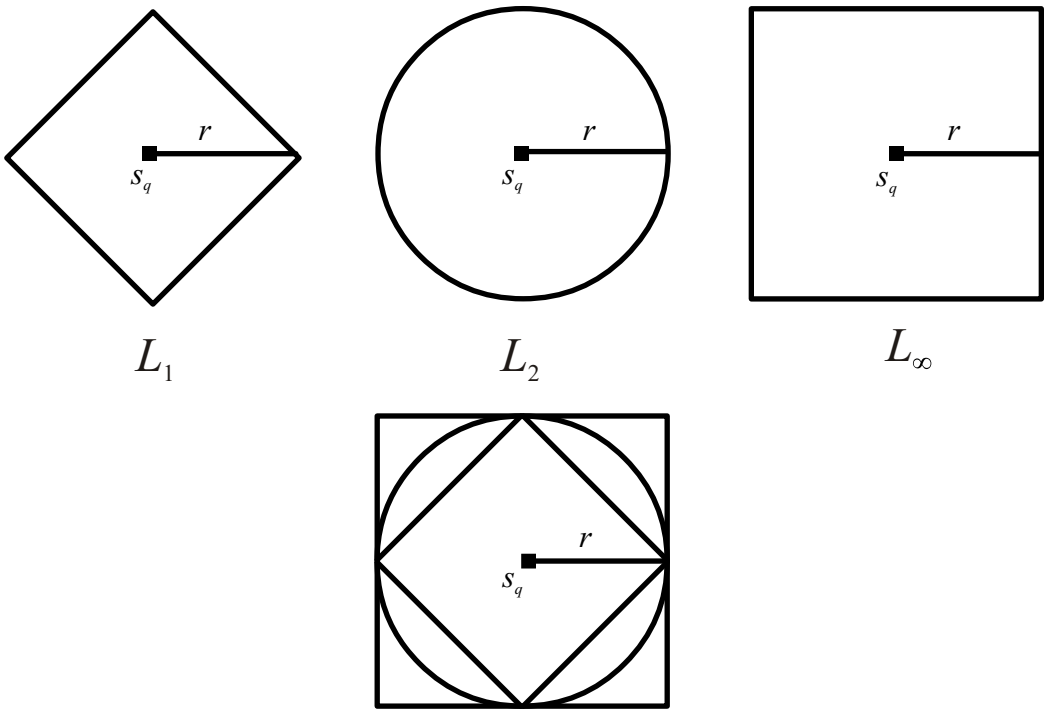

Figura 2.1: Representações bidimensionais de um mesmo ponto situado à distância r a partir do elemento s_q, considerando as três principais métricas da família de funções de distância métricas L_p: a L_1 ou *Manhattan*, a L_2 ou *Euclidiana* e a L_∞ ou *Chebychev*.

2.3 Árvores

Com exceção de alguns MA, como exemplo, os baseados em espalhamento (*hashing*), a abordagem mais utilizada pela grande maioria dos MA é recursiva, utilizando uma estrutura hierárquica, também conhecida como árvore.

Por definição, uma árvore é um grafo simples acíclico e conexo. Toda estrutura baseada em árvore é composta por blocos, conhecidos como nós (vértices). Cada um desses nós pode possuir uma ou mais ligações de saída, conhecidas como ponteiros (arestas), mas possuem somente uma ligação de entrada.

Cada árvore tem um único nó distinto designado por raiz. Somente esse nó não possui ligação de entrada. Dado um nó N_1 e um nó N_2, se N_1 possui um ponteiro para N_2, é dito que N_1 aponta para N_2. N_1 é chamado de nó pai (ou somente pai) de N_2 e N_2 é nó filho (ou somente filho) de N_1 (Figura 2.2(c)). Se um nó tiver um ou mais filhos ele é chamado de nó interno, caso contrário, é chamado de nó folha, ou simplesmente folha. Nós irmãos são nós que compartilham o mesmo nó pai. O sub-conjunto composto por um nó N e seus filhos diretos e não diretos é considerado uma sub-árvore ou ramo na qual N é raiz (Figura 2.2(a)).

Um nó pode possuir um ou mais elementos conhecidos como entradas do nó. A cardinalidade do nó é definida como o número de entradas deste nó. O grau do nó é definido como o número máximo de ponteiros (ligações de saída) que este nó pode possuir.

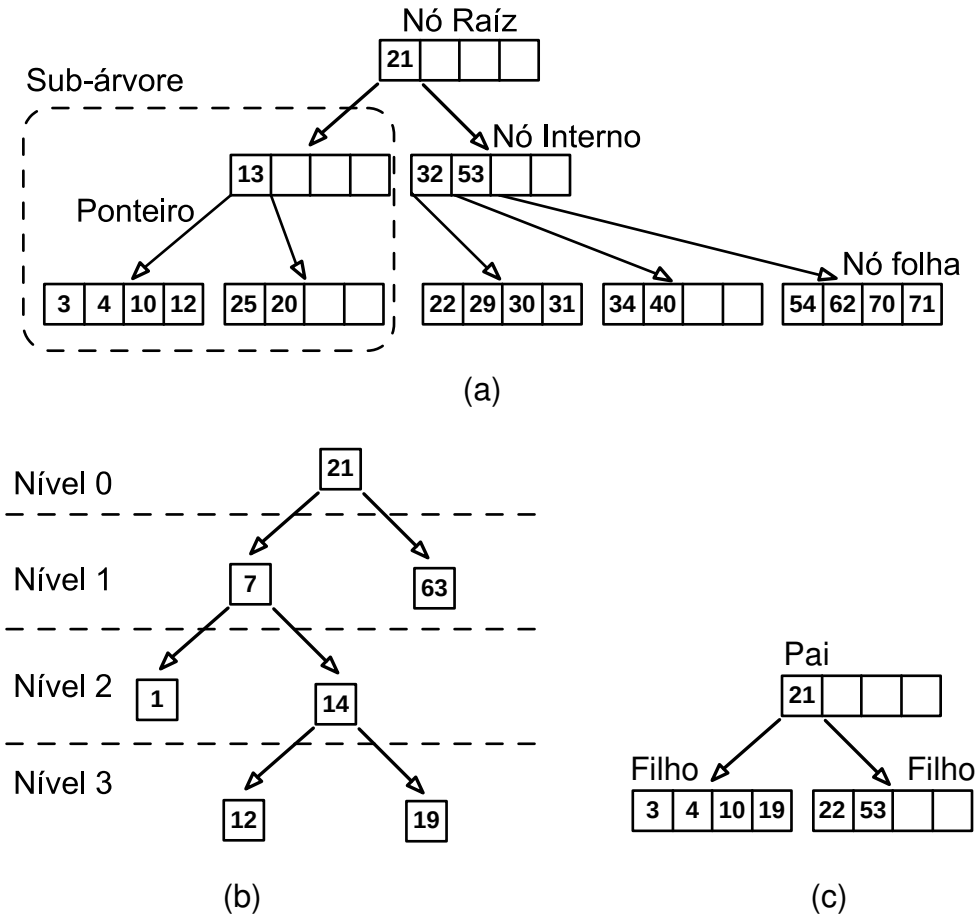

Figura 2.2: Exemplo de árvores destacando-se algumas definições: (a) árvore multivias com grau 5, contendo 4 entradas, pode-se observar sub-árvores, ponteiros entre os nós e a nomenclatura dos nós (raiz, interno, folha); (b) árvore binária, não balanceada, com altura 4, em destaque os níveis e (c) relação entre um nó pai e seus filhos.

O grau de uma árvore é definido como o máximo grau que um nó pode ter. Se uma árvore possuir grau maior que dois é chamada de árvore multivias, se possuir grau igual a dois é chamada de árvore binária e, neste caso, cada nó possui no máximo dois filhos e os filhos são conhecidos como filho esquerdo ou filho direito. Em MA hierárquicos, é comum que o grau da árvore obtida na indexação seja igual à cardinalidade do nó ou à cardinalidade do nó acrescida de uma unidade.

Um caminho é definido pelo conjunto de nós necessários para sair de um nó origem e chegar a um nó destino. O número de nós no caminho é conhecido como tamanho deste caminho. O tamanho do caminho máximo que pode ser feito em uma árvore considerando atingir qualquer nó a partir da raíz é conhecido como altura da árvore. Os nós que se encontram com o mesmo tamanho de caminho partindo da raiz são considerados no mesmo nível da árvore. A altura de um nó é o número de níveis dele até a raiz (Figura 2.2(b)). A árvore é dita balanceada se todas as sub-árvores de cada nó têm sempre a mesma altura, caso contrário é dita desbalanceada ou não balanceada.

Uma travesia em pré-ordem é feita com o processamento dos nós filhos apenas

após o nó corrente. Se os filhos forem processados previamente ao nó pai, a travessia é conhecida como em pós-ordem. Caso contrário, a travessia é dita em ordem simétrica.

2.4 Consultas por Similaridade

No espaço métrico, os dados geralmente são recuperados utilizando consultas por similaridade, pois a função de distância associada representa a dissimilaridade entre os elementos do espaço. Para esse tipo de recuperação, são dois os tipos de consultas mais utilizadas e importantes: Consultas por Abrangência (*Range Query* - RQ) e pelos k-Vizinhos Mais Próximos (*k-Nearest Neighbor Query* - kNNQ).

Seja o espaço métrico $\langle \mathbb{S}, d() \rangle$, e o conjunto $S \subseteq \mathbb{S}$. Então, os dois tipos mais comuns de consulta por similaridade podem ser descritos por:

1. **Consulta por Abrangência**: Esse tipo de consulta, expressa como $RQ(s_q,\ r_q)$, recupera os elementos que se encontram a uma distância máxima (r_q) a partir de um elemento de referência (s_q), onde $s_q \in \mathbb{S}$. Formalmente, pretende-se encontrar o sub-conjunto $A \subseteq O$ que atenda à seguinte equação:

$$RQ(s_q, r_q) = A = \{a | a \in S | d(a, s_q) \leq r_q\} \tag{2.4}$$

   O sub-conjunto A é conhecido como conjunto resposta.

2. **Consulta aos k-Vizinhos Mais Próximos**: Essa consulta, expressa como $kNNQ(s_q,\ k)$, recupera os k elementos mais próximos ao elemento de referência (s_q), no qual $s_q \in \mathbb{S}$. Formalmente, pretende-se encontrar o sub-conjunto $A \subseteq S$ que atenda a seguinte equação:

$$kNNQ(s_q, k) = A = \{a | a \in S | \forall s_i \in S - A, d(s_q, a) \leq d(s_q, s_i), |A| = k\} \tag{2.5}$$

Diversas aplicações podem utilizar essas duas consultas por similaridade para recuperação de informações. Um exemplo de consulta por abrangência é: "Selecione todas as palavras que sejam diferentes da palavra p em até 3 caracteres", onde o universo $\mathbb{S}$ é o conjunto de palavras. Nesse exemplo, s_q é a palavra p, o raio de busca r_q é igual a três caracteres, a métrica $d()$ é a função L_{Edit} e S é um banco de dados, ou sub-conjunto, contendo as palavras conhecidas, por isso, $RQ(p,\ 3) = A$ e $\mathbb{M} = \langle \mathbb{S}, L_{Edit} \rangle$, onde $A \subseteq S \subseteq \mathbb{S}$.

Diversas métricas podem ser utilizadas com o mesmo domínio de dados. É importante ressaltar que diferentes métricas caracterizam diferentes espaços métricos e, portanto, podem resultar em diferentes conjuntos resposta para o mesmo conjunto de dados.

A Figura 2.3 ilustra um exemplo de uma consulta por abrangência em um espaço bidimensional, utilizando as funções L_1, L_2 e L_∞. Os elementos contidos pelo raio de cobertura r_q definido pelas funções de distância pertencem ao conjunto resposta.

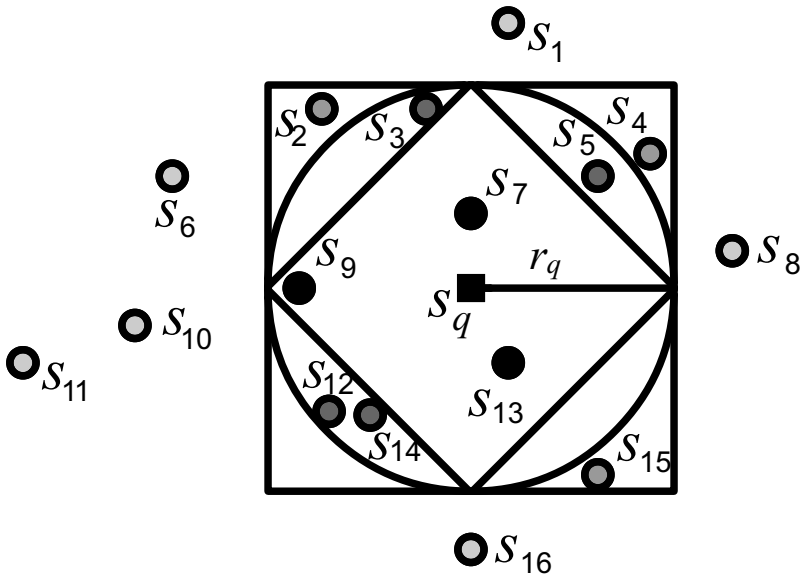

Figura 2.3: Exemplos de consultas por abrangência em um conjunto de elementos no espaço bidimensional, utilizando a família L_p: $RQ(s_q, r_q)$ sobre $\langle \mathbb{S}, L_1 \rangle = \{s_7, s_9, s_{13}\}$; $RQ(s_q, r_q)$ sobre $\langle \mathbb{S}, L_2 \rangle = \{s_3, s_5, s_7, s_9, s_{12}, s_{13}, s_{14}\}$; e $RQ(s_q, r_q)$ sobre $\langle \mathbb{S}, L_\infty \rangle = \{s_2, s_3, s_4, s_5, s_7, s_9, s_{12}, s_{13}, s_{14}, s_{15}\}$.

Utilizando o mesmo exemplo de palavras, para uma busca dos k-Vizinhos Mais Próximos, tem-se a seguinte consulta: "Selecione as 5 palavras mais semelhantes à palavra p", dessa forma, não se utiliza mais o raio de busca (r_q) e k é igual a cinco caracteres: $kNNQ(p, 5)$. É importante ressaltar que nesse tipo de consulta a quantidade k de elementos que se deseja no conjunto resposta é um parâmetro. Já na busca por abrangência é definido o grau de similaridade entre o elemento de busca em relação aos elementos desejados, não se conhecendo, portanto, quantos elementos existem no conjunto resposta.

A Figura 2.4 mostra uma consulta aos k-Vizinhos Mais Próximos, com $k = 3$ em um espaço bidimensional utilizando a função L_2. Representam-se os elementos que pertencem ao conjunto resposta conectados por uma linha numerada ao elemento de consulta. Esse número corresponde à ordem de proximidade em relação ao elemento de consulta.

É importante ressaltar que, caso a função de similaridade seja definida sobre um domínio discreto, como no exemplo de palavras para a função L_{Edit}, a probabilidade de ocorrer empate na escolha do vizinho mais próximo é maior e o conjunto resposta pode conter qualquer um dos elementos que fazem parte do empate, não necessariamente todos. Sendo assim, ou deve ser adotada uma política de escolha, ou é retornado um conjunto maior que o requisitado ou a mesma consulta solicitada outra vez pode retornar

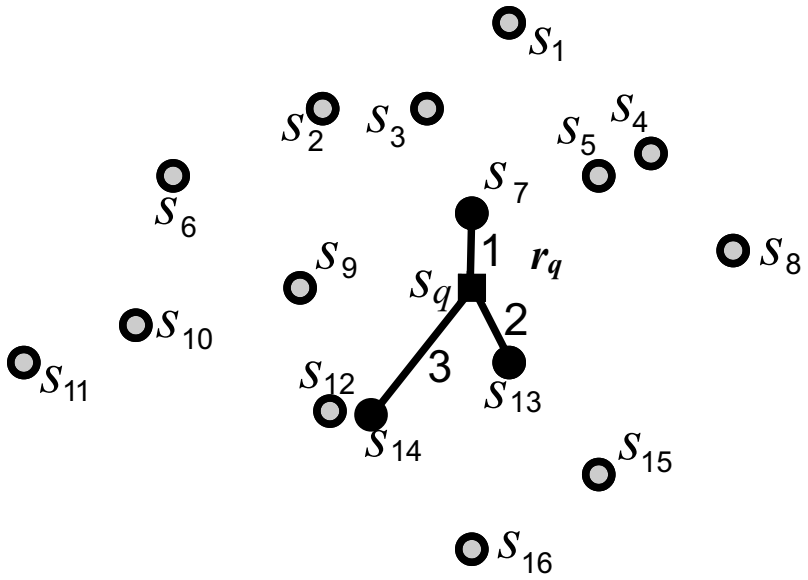

Figura 2.4: Exemplo de consulta por similaridade dos k-Vizinhos Mais Próximos utilizando a função de distância Euclidiana L_2 em um espaço bidimensional. O elemento s_q é o elemento de busca referencial enquanto os elementos escuros constituem o conjunto resposta: $kNNQ(s_q, 3) = \{s_7, s_{13}, s_{14}\}$.

Conjuntos resposta para $kNNQ$(gata, 5)
$A_1 = \{gata, gato, gota, gaia, lata\}$
$A_2 = \{gata, gato, gota, gaia, pata\}$
$A_3 = \{gata, gato, gota, lata, pata\}$
$A_4 = \{gata, gato, gaia, lata, pata\}$
$A_5 = \{gata, gota, gaia, lata, pata\}$

Tabela 2.1: Possíveis resultados para uma consulta por similaridade aos k-Vizinhos Mais Próximos - $kNNQ$(gata, 5), com $O = \{gata, gato, gota, gaia, lata, pata\}$.

um conjunto com elementos diferentes. Isso não ocorre na busca por abrangência, pois, todos os elementos com a mesma distância em relação à um elemento de consulta fazem parte do conjunto resposta. Como exemplo, se no conjunto de busca existem as palavras "gata", "gato", "gota", "gaia", "lata" e "pata", então L_{Edit}(gata, gata) = 0 e L_{Edit}(gata, gato) = L_{Edit}(gata, gota) = L_{Edit}(gata, gaia) = L_{Edit}(gata, lata) = L_{Edit}(gata, pata) = 1. Para consulta de abrangência com raio igual a um, o resultado é: $RQ(gata, 1) = \{gata, gato, gota, gaia, lata, pata\}$, ou seja, seis elementos. A tabela 2.1 mostra os possíveis resultados para uma consulta dos k-vizinhos mais próximos com k igual a cinco, sendo que todos eles são corretos.

2.5 Exemplo de Aplicações

Geralmente, os dados complexos são recuperados utilizando consultas por similaridade. Isso é feito utilizando uma função de dissimilaridade entre os elementos para comparação de um elemento com o outro. Como exemplo dessas aplicações, existem bases de dados compostas por seqüências de DNA que podem fazer parte do domínio de seqüências de caracteres, composto por A, G, C e T [Hunt et al., 2001]. Outro exemplo, que está se tornando bastante freqüente, é o uso de séries temporais, que são coleções de dados com informações temporais, como vídeos (seqüências de imagens com um fator temporal).

Em sistemas de informações geográficas, usa-se o domínio de coordenadas cartográficas (coordenadas bidimensionais) das cidades e acidentes geográficos e a métrica Euclidiana. Dessa forma, pode-se achar a proximidade (distância) entre as cidades, como por exemplo: "Selecione as cinco cidades mais próximas da cidade de São Carlos-SP" ou ainda, "Selecione as cidades que estão a no máximo 100Km da cidade de São Carlos-SP".

Outra aplicação emergente é o uso de bases com dados multimídia [Patella, 1999]. Normalmente, um método automático utiliza a extração de características (histograma, forma, cor) para posterior consulta por similaridade utilizando uma função de distância [Faloutsos, 1996]. Outros exemplos são as aplicações para reconhecimento de áudio (fala, por exemplo) utilizando transformadas, como *wavelets*, na busca por similaridade em trechos do áudio.

A consulta por similaridade utilizando imagens médicas é outro exemplo bastante estudado. Pode-se definir um domínio como o conjunto de imagens mamográficas, sendo que vetores de características devem ser extraídos para comparação, como exemplo, a técnica de Momentos de *Zernick* [Felipe et al., 2006]. Com isso pode-se identificar quais imagens são as mais adequadas nas consultas por similaridade.

Existem inúmeras aplicações que podem fazer o uso de consultas por similaridade e diversas outras são criadas constantemente. Por isso, faz-se necessário o uso de um suporte adequado e eficiente para a busca e recuperação satisfatória no uso de sistemas baseados em consultas por similaridade entre os elementos.

2.6 Considerações Finais

Como descrito neste capítulo, em domínios tradicionais, como números e pequenas seqüências de caracteres, a ordenação é possível, pois obedecem a ROT. Em virtude de sua simplicidade na manipulação dos dados, esses domínios são amplamente utilizados. Já para dados complexos, a ROT nem sempre é válida, e por isso é necessário explorar outras propriedades dos novos espaços, como o espaço multidimensional e o métrico. Para o uso desses dados, utiliza-se uma estrutura hierárquica, também chamada de árvore. Essa estrutura possui propriedades que podem ser usadas para otimização de consultas (as

quais serão detalhadas melhor no capítulo 3).

As consultas mais adequadas para o crescente número de dados complexos são as consultas por similaridade. Entre elas existem dois tipos que se destacam: as consultas por abrangência e pelos k-vizinhos mais próximos. As consultas do primeiro tipo têm como objetivo a recuperação de elementos a partir de um elemento referência e um determinado raio. O segundo tipo de consulta recupera os k elementos mais próximos em relação a um elemento pré-determinado, sem se preocupar com o tamanho do raio de busca necessário para essa recuperação.

O próximo capítulo detalhará melhor os métodos de acesso, com destaque para os MAM. Esses métodos são os mais adequados e utilizados para responder à consultas por similaridade. Para indexar elementos utilizando esses métodos são apenas necessárias as distâncias relativas entre os elementos.

Capítulo

3

Métodos de Acesso

3.1 Considerações Iniciais

Métodos de acesso são técnicas que agilizam operações de busca em bases de dados. Esses métodos são fundamentais em um SGBD e utilizam as propriedades inerentes ao domínio de indexação utilizado. Como descrito nas próximas seções, em domínios com ROT, a navegação em uma estrutura hierárquica divide o espaço de indexação em duas partes: os elementos que são maiores e os que são menores em relação à um dado elemento. Com isso, pode-se "podar" galhos da árvore de indexação, evitando o processamento sobre nós não necessários à busca, fazendo com que a consulta fique restrita a um único caminho da árvore.

Em domínios multidimensionais, é possível que haja sobreposição de regiões e, nesse caso, não há como garantir que a busca fique restrita a um único caminho da árvore. Nos espaços métricos é possível otimizar as consultas utilizando a propriedade da desigualdade triangular, realizando a poda de algumas regiões não significativas para consulta. Utilizando essas propriedades, consegue-se descartar um conjunto significativo de dados sem compará-los ou mesmo recuperá-los do disco, mas também não se pode garantir que precisa ser percorrido somente um caminho.

3.2 Métodos de Acesso em Domínios que atendem à Relação de Ordem Total

Os tipos de estruturas criadas por MA em domínios que atendem à ROT são destinados a indexar valores passíveis de ordenação, como números ou pequenas seqüências de

caracteres. As consultas mais utilizadas para esses métodos são as consultas por faixa de valores e por igualdade, como por exemplo: "Selecione todas a pessoas que possuem idade entre 18 e 20 anos" ou "Selecione o aluno com código de matrícula igual a 12345". Para esses tipos de dados e consultas existem diversos métodos de indexação que podem ser aplicados. Entre os mais importantes, têm-se os índices simples (densos e esparsos), tabelas de espalhamento (*hashing*) e as *B-trees*.

O *hashing*, ou espalhamento, é um método não hierárquico que divide o espaço de indexação em grupos a partir de uma função de espalhamento. A função de espalhamento associa um valor a um grupo identificado por uma chave, chamado de célula ou agrupamento. A figura 3.1 ilustra a aplicação da função de espalhamento sobre um conjunto de valores.

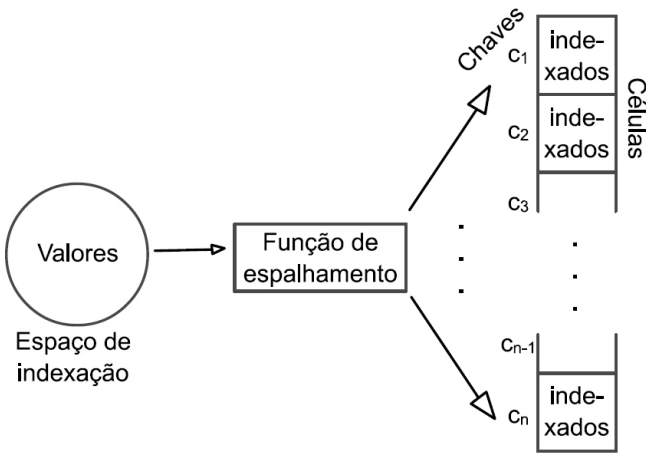

Figura 3.1: Funcionamento do método *hashing*. A chave de um elemento é processada pela função de espalhamento que associa um elemento a uma célula de armazenamento.

Quando uma busca é realizada, aplica-se a função ao valor da consulta e como resultado obtém-se a chave que identifica a célula onde o índice se encontra. Por dividir os índices em células, o *hashing* não é eficiente para consultas não pontuais, como as consultas por faixa de valores, pois provoca visitas em várias células.

Na operação de inserção, quando existe mais que um índice mapeado em um mesmo endereço, são necessários algoritmos para especificar onde o elemento será inserido, e essa ocorrência é conhecida como colisão. Uma colisão provoca efeito negativo no desempenho dessas estruturas, pois quando há várias colisões em uma mesma célula, o desempenho de busca nessa célula é semelhante ao da busca seqüencial.

Além do *hashing* existe a *B-tree* e suas variantes [Bayer & McCreight, 1972], as quais são bastante utilizadas nos bancos de dados relacionais comerciais atuais. Este tipo de estrutura de árvore tem ampla aceitação quanto ao uso junto às implementações de sistemas computacionais, pois proporciona bom desempenho e é de fácil compreensão e implantação.

Uma *B-tree* [Bayer & McCreight, 1972] é uma árvore balanceada multivias (Figura

3.2). O algoritmo de inserção garante 50% de ocupação mínima em cada nó, ou seja, se o nó tem capacidade máxima para n elementos, a inserção garante no mínimo $n/2$ elementos por nó, com exceção do nó raiz. A cardinalidade de uma *B-tree* somada de uma unidade é igual ao grau da estrutura. Os elementos de uma sub-árvore à esquerda de um determinado elemento são todos menores, e os à direita são maiores, que esse elemento.

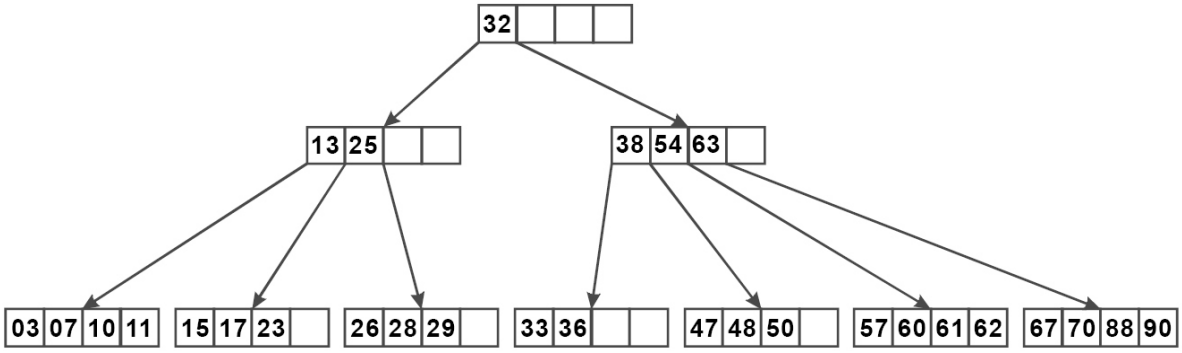

Figura 3.2: Exemplo de uma árvore *B-tree* numérica com grau 5 e cardinalidade 4. Note-se a relação de ordem total e o balanceamento da árvore.

Uma B-tree de ordem $n > 1$ possui as seguintes regras e propriedades:

- Todo nó deve ter no máximo n elementos;
- Todo nó, exceto a raiz, deve possuir no mínimo $\lceil n/2 \rceil$ elementos;
- Todo nó com m elementos e que não seja folha possui $m + 1$ nós filhos diretos;
- Os nós folha estão todos em um mesmo nível;
- Os descendentes do intervalo de dois elementos (s_i e s_j, sendo $s_i < s_j$), deve possuir apenas elementos que estejam entre os dois valores pais (maior que s_i e menor que s_j).

A inserção de novos elementos é sempre feita nos nós folha, por isso o crescimento da árvore é feito de baixo para cima (*bottom-up*). Quando um nó está cheio e deseja-se inserir um elemento neste nó, os elementos são divididos em dois nós e é feita a escolha do elemento pai (elemento central), inserindo este elemento no nó pai. Somente quando existe uma divisão no nó raiz a árvore cresce em tamanho. Devido à essa forma de particionamento, a *B-tree* é balanceada e independente da seqüência de inserção. Essa propriedade torna esta árvore uma das estruturas mais eficientes para armazenamento de índices que obedecem a ROT.

Na tentativa de aumentar a taxa de ocupação da *B-tree* (50%), foi criada a B^*-*tree*. O algoritmo de inserção desse método de acesso tenta adiar a divisão nodal até o momento em que dois nós irmãos estejam completamente cheios, o que é feito utilizando uma política de redistribuição local. Quando os dois nós estiverem totalmente cheios e

a inserção de um novo elemento for requisitada, a distribuição e divisão dos elementos são feitas em três nós, cada um com 2/3 das entradas preenchidas. A taxa de ocupação passa a ser de pelo menos 66% da capacidade do nó, melhorando a utilização do espaço de armazenamento e o espalhamento dos elementos na árvore. Com isso, tem-se uma otimização do número de acessos a disco e, conseqüentemente, do tempo de busca, pois a altura da árvore tende a ser menor.

Uma das variantes da *B-tree* mais utilizada é a B^+*-tree*. Nesse método de acesso, todas as chaves se localizam nos nós folha. Os nós internos são compostos por cópias de elementos dos nós folha, servindo de caminho para busca otimizada dos índices. Os nós folha são ligados por uma lista (Figura 3.3), formando um conjunto seqüencial ordenado [Comer, 1979]. Essa modificação permite uma otimização na recuperação de consultas seqüenciais e por faixas de valores. Por essas características, esse método de acesso é conhecido também como Método de Acesso Seqüencial Indexado (MASI).

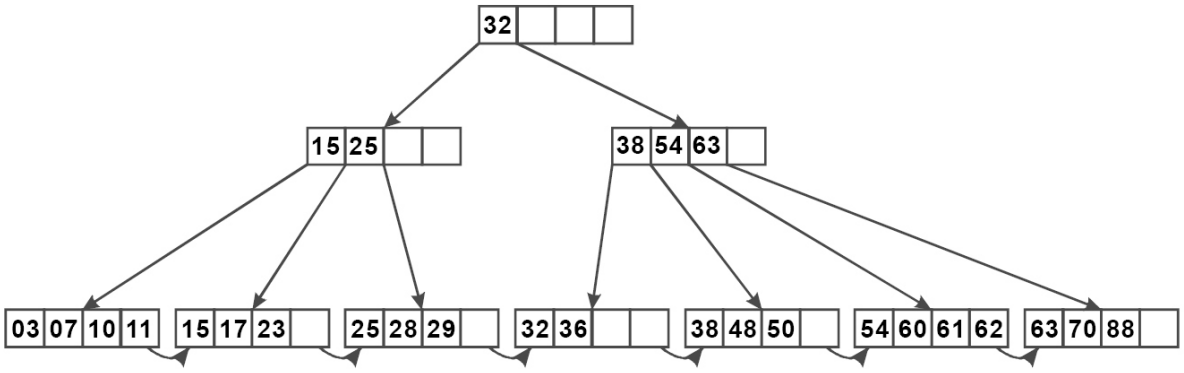

Figura 3.3: Exemplo de uma árvore B^+*-tree* (Método de Acesso Seqüencial Indexado) numérica com grau 5 e cardinalidade 4. Nos nós folha há uma lista para busca seqüencial.

Em todas a *B-tree*, como há uma relação de ordem total entre os elementos, consegue-se estabelecer duas partes disjuntas de dados. Por isso, a cada elemento comparado, somente uma sub-árvore precisa ser avaliada. Por esse motivo, existe a divisão de elementos maiores à direita e menores à esquerda (Figura 3.4). Conseqüentemente, ao se realizar uma consulta nessa árvore, se uma chave for menor que a contida na raiz, então a sub-árvore à direita pode ser descartada, otimizando o número de elementos necessários para avaliação. Por essa propriedade, o algoritmo de busca em uma *B-tree* possui complexidade $\Theta(\log n)$ para uma consulta pontual. Como será visto a seguir, essa propriedade não é válida para domínios espaciais e métricos quando há sobreposição de regiões.

3.3 Métodos de Acesso Multidimensionais

Com a necessidade do suporte a domínios mais elaborados, foi necessária a criação de métodos que pudessem trabalhar com elementos complexos, como vetores numéricos ou

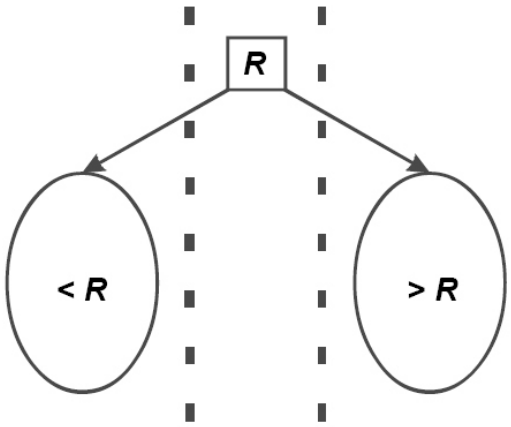

Figura 3.4: Exemplo de uma árvore B^+-*tree* (Divisão de duas partes disjuntas nas estruturas hierárquicas em domínios com relação de ordem total. Os elementos menores que R estão à esquerda e os maiores à direita.

elementos multidimensionais. Para suprir essa necessidade, foram desenvolvidos os Métodos de Acesso Multidimensionais.

Existem dois tipos de MAMd: os que indexam somente pontos n-dimensionais, conhecidos como Métodos de Acesso Multidimensionais Pontuais (MAMdP) e MA para dados não somente pontuais mas que indexam regiões no espaço, ou simplesmente, Métodos de Acesso Multidimensionais Não Pontuais (MAMdNP) [Gaede & Günther, 1998].

A transformação (ou *object mapping* em inglês) de um elemento em um ponto de dimensionalidade maior é um exemplo de uma técnica que transforma um MAMdP em um MAMdNP. Por exemplo, um retângulo bidimensional pode ser descrito por dois pontos (x_0, y_0) e (x_1, y_1); para conseguir indexar esse retângulo em uma MAMdP é necessário transformar esses dois pontos em um único: (x_0, y_0, x_1, y_1). Com isso, consegue-se indexar um retângulo bidimensional em um MAMdP para quatro dimensões [Gaede & Günther, 1998].

Outra técnica para MAMdNP é a sobreposição (ou *overlapping* no termo em inglês), que consiste em estabelecer regiões delimitadas como nós da estrutura. Existem ainda as técnicas de corte (*clipping*) e de camadas múltiplas. A primeira consiste em quebrar as regiões eliminando a sobreposição, para evitar a busca em sub-árvores desnecessárias. A segunda consiste em ter várias representações do mesmo espaço (camadas) com regiões mutuamente disjuntas, contudo, as camadas se sobrepõem entre si.

Uma das principais representantes dos MAMdNP é a *R-tree*. Baseada nas *B-trees*, cada nó desta árvore corresponde a uma região retangular que é armazenada em uma página no disco [Guttman, 1984]. O intervalo contido em um nó não-folha contém os intervalos de seus descendentes, sendo que os nós que estão no mesmo nível da árvore podem conter sobreposições dessas regiões. A figura 3.5 ilustra o particionamento dos dados com sobreposição. Os nós folha da árvore contêm os retângulos limitantes mínimos (*Minimum Bounding Rectangle* - MBR) com uma referência para a página de dados.

Assim como nas *B-trees*, com exceção da raiz, os nós contêm um número de entradas entre os valores limites entre 50% da cardinalidade e o total da cardinalidade.

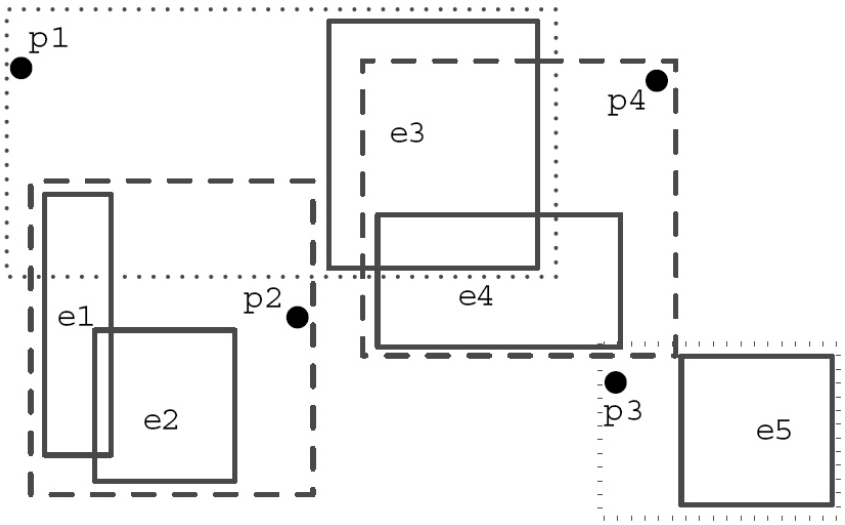

Figura 3.5: Representação de um espaço de indexação de uma árvore *R-tree*. Elementos pontuais (p_1, p_2, p_3, p_4), regiões no espaço (e_1, e_2, e_3, e_4, e_5) e retângulos limitantes mínimos (MBR).

Como existe sobreposição dos dados, a busca se torna menos eficiente, pois permite a ocorrência de consultas em múltiplos nós e muitas vezes desnecessárias. Além disso, não existe uma *R-tree* única para um conjunto de elementos e a operação de remoção é baseada em um algoritmo complicado, pois esta operação provoca o re-balanceamento da árvore, o que é uma operação custosa.

A operação de inserção é crítica, pois a construção da árvore depende da ordem em que os dados são inseridos, o que pode refletir em bons ou maus resultados durante as operações de buscas. Assim, no sentido de minimizar o esforço nas buscas, foi introduzida uma política de re-inserção forçada que originou a *R*-tree*. Esta é uma estrutura que minimiza a sobreposição de regiões no mesmo nível da árvore [Beckmann et al., 1990].

As *R+-trees* foram criadas com o intuito de evitar o problema da sobreposição que ocorria com as *R-trees* [Sellis et al., 1987]. Elementos que se interceptam em mais de um intervalo são armazenados em diferentes páginas em disco, com isso, a busca pontual pode se tornar mais rápida por ser restrita a um único caminho da árvore. No entanto, a remoção e divisão são operações mais complicadas, pois além de modificar a porção da árvore acima, pode-se provocar a modificação da árvore transversalmente. Como conseqüência, a operação de remoção pode ocasionar fragmentação dos dados causando uma má utilização do espaço de memória disponível.

Recentemente, houve um interesse no uso dessas estruturas para suporte de aplicações não convencionais em base de dados. As *R-trees* já foram implementadas em SGBD comerciais como o *Oracle*, principalmente para o suporte aos sistemas de informações geográficas [Kothuri et al., 2002].

3.4 Métodos de Acesso Métricos

Elementos adimensionais ou dados com dimensões elevadas não são indexados eficientemente pelos MAMd. Para suprir essa necessidade foram desenvolvidos os Métodos de Acesso Métricos - MAM. Os MAM indexam dados complexos utilizando uma função de distância associada, conforme visto na seção 2.2.2. É freqüente que elementos complexos sejam representados por características que são extraídas deles, em geral representadas por valores numéricos. Por exemplo, em um domínio de imagens, têm-se histogramas, descritores de formas e medidas de textura, e a comparação é realizada a partir de uma função de distância métrica pré-definida. A indexação métrica facilita operações em que a similaridade é fator primordial.

A idéia geral de um método de acesso métrico é selecionar um ou mais elementos e colocá-los como representantes de conjuntos. Pode-se diferenciar os MAM pela forma como esses representantes são escolhidos e pela forma como os demais elementos estão dispostos em relação à eles. Como as distâncias entre os elementos e seus representantes são armazenadas ou podem ser calculadas, pode-se utilizar a propriedade da desigualdade triangular para eliminar a necessidade de avaliar algumas regiões desnecessárias. Dessa forma, são reduzidos os números de acesso a disco e de cálculos de distâncias.

A poda pela desigualdade triangular pode ser feita da seguinte forma: Seja o espaço métrico $\mathbb{M} = \langle \mathbb{S}, d() \rangle$, o conjunto de elementos $S \subset \mathbb{S}$, o elemento de consulta $s_q \in \mathbb{S}$, o raio de consulta r_q e um representante s_{rep} (veja Figura 3.6). Todo elemento $s_i \in S$ pode ser descartado se:

$$d(s_{rep}, s_i) < d(s_{rep}, s_q) - r_q \tag{3.1}$$

ou se,

$$d(s_{rep}, s_i) > d(s_{rep}, s_q) + r_q \tag{3.2}$$

Com isso, consegue-se diminuir o espaço de busca, economizando consultas e acessos a disco desnecessários para recuperação de elementos em uma requisição. Pode-se observar na figura 3.6 que a região $R1$ é descartada da busca pela equação 3.1 e a região $R3$ é descartada pela equação 3.2. Portanto, não serão necessários cálculos de distâncias para os elementos s_5 e s_6 da região $R1$ e nem para os elementos s_1, s_7 e s_9 da região $R3$.

Existem dois tipos de MAM: aqueles usados com funções de distância discreta e aqueles usados com funções de distância contínua [Burkhard & Keller, 1973]. Os MAM com funções de distância discreta criam estruturas hierárquicas que possuem funções que retornam um pequeno conjunto de valores (conjunto discreto). Existem formas de adaptar essas estruturas para funções de distância contínua, mas isto implica em se ter muitos nós-filho para cada representante ou pivô, que são os elementos raiz de uma sub-árvore, o que

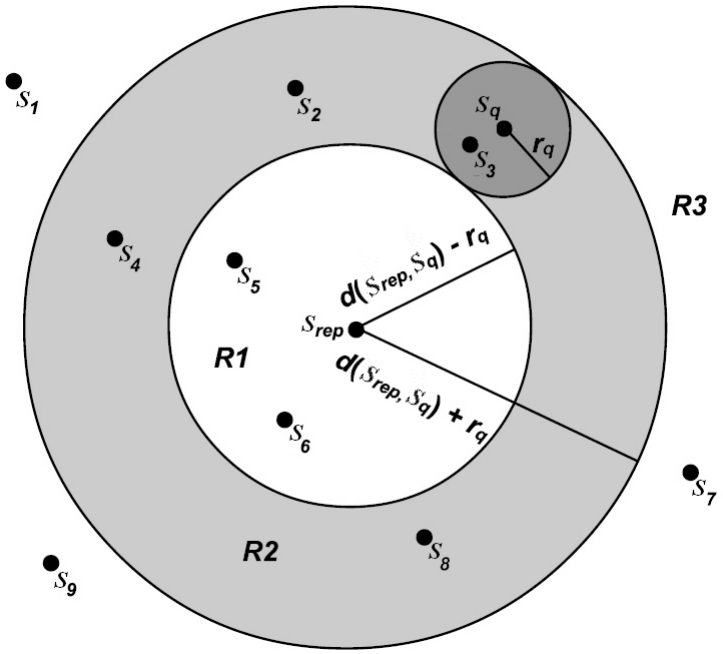

Figura 3.6: Representação de uma árvore *Slim-tree* (Descarte ou poda de regiões utilizando a propriedade da desigualdade triangular, s_q é o elemento de consulta, r_q o raio da consulta e s_{rep} é o representante. Os elementos da região $R1$ (s_5 e s_6) e os elementos da região $R3$ (s_1, s_7 e s_9) são descartados pela desigualdade triangular.

ocasiona perda de desempenho.

Além disso, existem MAM estáticos e dinâmicos. Um método de acesso estático constrói a estrutura de indexação utilizando todo o conjunto de dados disponível em uma única operação, não podendo ser utilizado para inserções, remoções e recuperação freqüentes do disco de maneira eficiente.

O primeiro método dinâmico criado foi a *M-tree* [Ciaccia et al., 1997], a qual será melhor detalhada na seção 3.4.2. O Grupo de Bases de Dados e de Imagens do ICMC/USP tem participado com o desenvolvimento de alguns dos mais rápidos MAM dinâmicos recentemente publicados, tais como a *Slim-tree* [Traina Jr. et al., 2000], a *DF-tree* [Traina Jr. et al., 2002], a *DBM-tree* [Vieira et al., 2006] e a família *OMNI* de MAM [Traina Jr. et al., 2006].

3.4.1 Métodos de Acesso Estáticos

Alguns MAM não permitem a inserção nem a remoção de elementos posterior à criação do índice. Eles constróem a estrutura de indexação utilizando todos os dados em uma única operação e, por isso, são conhecidos como métodos de acesso estáticos

O trabalho de *Burkhard* e *Keller* [Burkhard & Keller, 1973] é o primeiro trabalho encontrado na literatura sobre indexação de dados em domínios métricos de maneira estática. Naquele trabalho são propostas três técnicas que particionam o espaço métrico

hierarquicamente.

A primeira técnica é conhecida como decomposição hierárquica multivias. No primeiro nível, escolhe-se arbitrariamente um elemento representante do domínio e os demais elementos são agrupados de acordo com a distância em relação ao representante. Como os dados são discretos, é possível separar os elementos com distâncias iguais em grupos. Isso é feito recursivamente em todos os grupos formados, criando-se vários níveis, definindo-se, assim, uma estrutura hierárquica multivias.

A segunda técnica divide o espaço de dados em relação a um número fixo de representantes. Para cada partição é escolhido arbitrariamente um representante e o raio é calculado segundo a distância máxima entre todos os elementos em relação ao representante. Os elementos em cada conjunto são divididos da mesma maneira recursivamente, criando uma estrutura hierárquica multivias. Cada nó da estrutura guarda os representantes, os raios e ponteiros para os sub-conjuntos de elementos indexados. Para que um elemento faça parte de um sub-conjunto específico, sua distância ao representante tem que ser menor ou igual ao respectivo raio máximo.

A terceira técnica é semelhante à anterior, mas a distância máxima entre quaisquer dois elementos em um mesmo grupo não deve ser maior que uma constante c, cujo valor é diferente para cada nível da estrutura. O grupo satisfazendo esse critério é chamado de clique. A escolha do valor de c tem que garantir que todos os elementos do espaço estejam em pelo menos um dos cliques, podendo um elemento aparecer em mais que um clique. Em seguida, um elemento arbitrário de cada clique é escolhido como seu representante. Essas técnicas foram utilizadas para criação de alguns MAM.

Um método de acesso estático bastante conhecido é a *VP-tree* (*Vantage Point Tree*). Esse método particiona um conjunto de dados de acordo com a distância dos elementos em relação a um ponto de referência ou pivô [Yianilos, 1993]. O valor médio de tais distâncias é usado como separador para particionar os elementos em dois subconjuntos balanceados. Uma árvore binária é construída recursivamente utilizando qualquer elemento como raiz e os elementos cuja distância à raiz seja menor ou igual à mediana são inseridos na sub-árvore esquerda, enquanto aqueles cuja distância à raiz seja maior, são inseridos à direita. Esta estrutura de árvore não é balanceada, nem paginada e é dependente da escolha do pivô.

A *MVP-tree* ou *Multiple Vantage Points Tree* possui as mesmas funcionalidades da VP-tree, mas não mais utilizando uma árvore binária, pois a árvore MVP-tree utiliza múltiplos pontos de vantagem para particionar os elementos [Bozkaya & Özsoyoglu, 1997] [Bozkaya & Özsoyoglu, 1999]. A *MVP-tree* também usa distâncias pré-computadas (armazena na árvore as distâncias entre os elementos e seus representantes) para aumentar a velocidade da busca reduzindo o número de cálculos de distância necessários para executar a busca. Por ser uma estrutura que apresenta menor profundidade, possui um bom desempenho de consultas para a recuperação.

Outra generalização da *VP-tree* é a *Excluded Middle Vantage Point Forest*, que consiste em remover os elementos com distância intermediária (local de maior concentração) em relação ao pivô. Os elementos removidos são colocados em uma outra árvore e o mesmo processo é efetuado novamente. Com isso têm-se várias árvores, ou melhor, uma floresta. Essa estrutura é muito boa para responder a consultas por abrangência.

Existem outros métodos de indexação que não são baseados em estrutura de árvores. Seus principais representantes são as estruturas *AESA*, *LAESA*, variantes destas estruturas e *clustering* [Chávez et al., 2001].

O *Approximating Eliminating Search Algorithm* (*AESA*) [Vidal, 1986] é uma matriz triangular superior de ordem n com distâncias pré-computadas. Quando se necessita buscar algum elemento, escolhe-se um deles aleatoriamente e calcula-se sua distância em relação ao elemento da consulta. São eliminados os elementos que possuem distância à raiz maior ou igual, subtraindo-se o raio de tolerância ou que possuem a distância menor ou igual somando-se o raio de tolerância. Por possuir distâncias pré-computadas, esse algoritmo possui complexidade $\Theta(1)$ para tempo de busca, mas o tempo de construção e o espaço possuem $\Theta(n^2)$, o que é inviável mesmo para bases de dados pequenas.

O *LAESA* (*Linear AESA*) [Micó et al., 1994] e suas variantes são uma nova versão do *AESA*. Utilizam-se de k pivôs fixos para que o espaço e o tempo de construção sejam de ordem $\Theta(k \cdot n)$. Na *LAESA*, os elementos são percorridos linearmente, sendo que os elementos que não puderem ser eliminados após serem considerados os k pivôs são comparados diretamente. Para isso são utilizados ponteiros extras, que requisitam mais espaço. O objetivo principal é dividir o espaço em sub-conjuntos, sendo que elementos próximos uns aos outros são colocados no mesmo agrupamento. Assim, o agrupamento de um elemento constitui-se de seus m elementos mais próximos, sendo que cada agrupamento armazena seu raio de cobertura. Ao efetuar a busca, os agrupamentos são verificados um a um. Esta estrutura apresenta desempenho ruim para consultas por abrangência, pois é necessário percorrer vários agrupamentos.

3.4.2 Métodos de Acesso Dinâmicos

Nenhuma das estruturas criadas utilizando Métodos de Acesso Estátiscos são adequadas quando operações de inserções e remoções são necessárias após a construção da estrutura pois necessitam de custosas reorganizações ou a re-criação total da estrutura. Para evitar esse problema são necessários algoritmos mais eficientes e que trabalhem de maneira dinâmica, permitindo a auto-organização da estrutura ao se inserir novos elementos. Esses MA são conhecidos como MA dinâmicos. A seguir são descritos os MAM dinâmicos mais relevantes que são encontrados na literatura: a *M-tree*, a *Slim-tree*, a *DF-Tree*, a *DBM-Tree* e a *família OMNI*.

M-tree

O primeiro método de acesso métrico dinâmico apresentado na literatura foi a *M-tree*, baseada em uma das técnicas de [Burkhard & Keller, 1973]. Essa estrutura não necessita de reestruturações periódicas e a sua construção é de baixo para cima (*bottom-up*), garantindo o balanceamento da estrutura. Todos os elementos são armazenados nos nós folha, mas as regiões delimitadas pelo raio de abrangência de dois ou mais nós irmãos podem se sobrepor [Ciaccia et al., 1997].

Cada entrada de um nó folha é composta por um identificador de objeto (OId), os elementos inseridos no nó e a distância de um a um ao representante. Cada entrada no nó interno é composta por um ponteiro para o nó filho, um representante junto com a distância do elemento para o pai, com exceção do nó raiz, e o raio de cobertura de toda sub-árvore, que delimita a região que engloba todos os elementos filhos. Nas consultas é utilizada a propriedade da desigualdade triangular para evitar cálculos de distâncias desnecessários.

Na inserção, caso nenhum nó esteja qualificado para receber um novo elemento s_i, ou seja, não existe um s_{rep} tal que $d(s_{rep}, s_i) \geq r_{rep}$, onde r_{rep} é o raio de cobertura de s_{rep}, é escolhido o nó que possua a menor distância em relação ao seu representantivo aumentando o raio de cobertura para englobar s_i. Caso haja mais de uma região que cubra s_i, então é aplicado um algoritmo para escolha do nó a inserir; esse algoritmo é conhecido como *ChooseSubtree*. Existem duas diferentes maneiras de execução: **aleatório** e **distância mínima**.

O algoritmo **aleatório** escolhe aleatoriamente o nó de inserção. É o algoritmo mais veloz de inserção, mas o seu comportamento em operações de busca, nem sempre é satisfatório. O algoritmo **distância mínima** escolhe o nó que possui o representante mais próximo de s_i. Quando um nó atingiu sua capacidade máxima e há a necessidade da inserção de um novo elemento, é necessário particionar o nó, substituindo o elemento representante por outros dois elementos representantes e dois novos nós. Para isso são necessários algoritmos de escolha dos elementos representantes, conhecidos como algoritmos de promoção.

Dentre os inúmeros algoritmos existentes para escolha dos representantes, pode-se citar os seguintes:

- **m_RAD**: Avalia todos os pares de elementos como representativos, escolhendo o par que tiver a menor soma do raio de cobertura. É um dos algoritmos mais custosos para escolha dos elementos representantes, com complexidade $\Theta(n^3)$, mas é o que provê a melhor escolha para otimizar o desempenho das buscas.

- **mM_RAD**: Similar ao m_RAD, mas minimiza o máximo que cada um dos raios de cobertura pode atingir para escolha dos melhores representantes.

- **M_LB_DIST**: Conhecido como *Maximum Lower Bound on Distance*, esse algoritmo utiliza distâncias pré-computadas. Para escolher dois representantes (s_{rep1}, s_{rep2}), o algoritmo considera $s_{rep1} \equiv s_{rep}$, sendo que s_{rep2} deverá ser o objeto mais distante em relação a s_{rep}, ou seja: $d(s_{rep2}, s_{rep}) = max_j \{d(s_j, s_{rep})\}$.

- **RANDOM**: Seleciona os dois representantes aleatoriamente. É um algoritmo rápido, mas nem sempre provê bom desempenho na busca.

- **SAMPLING**: é escolhido um conjunto aleatório de elementos, e o par de elementos desse conjunto cuja soma do raio de abrangência seja mínima é escolhido como par de representantes.

Após a escolha dos elementos representantes, é necessário distribuir os elementos em dois conjuntos identificados por esses representantes. Isso pode ser feito de duas formas: distribuir em conjuntos com o mesmo número de elementos ou pela distância em relação aos representantes. A primeira técnica visa equilibrar a estrutura, escolhendo um elemento mais próximo do primeiro representante e, após isso, escolhendo-se o elemento mais próximo do segundo representante. Essas etapas são executadas até que não sobre mais elementos a serem distribuídos. Com isso têm-se dois novos conjuntos com quantidades de entradas iguais. A segunda técnica se preocupa em colocar os elementos o mais próximo de cada representante, ou seja, verifica-se de qual representante o elemento está mais próximo. A segunda técnica não garante que os dois subconjuntos resultantes possuam a mesma quantidade de elementos.

Apesar de ser um método dinâmico eficiente, o maior problema dessa estrutura é que não existe nenhum mecanismo para redução da sobreposição das regiões. Quando existe alta sobreposição, a propriedade da desigualdade triangular não consegue podar as regiões de consultas, causando perda de desempenho.

3.4.3 Slim-tree

A *Slim-tree* [Traina Jr. et al., 2000] aperfeiçoou a *M-tree*, sendo o primeiro método de acesso métrico a atacar o problema da sobreposição. O algoritmo *Slim-down* é uma técnica que elimina as regiões que se sobrepõem. Assim como a *M-tree*, esse método utiliza a desigualdade triangular para otimização das consultas, os elementos são armazenados nos nós folha e há uma hierarquia de representantes para cada nó.

Além dos dois algoritmos de escolha da sub-árvore existentes na *M-tree*, o **aleatório** e o de **distância mínima**, a *Slim-tree* introduziu um novo algoritmo para escolha de uma sub-árvore na inserção, o algoritmo de **ocupação mínima**, sendo necessária a inclusão de mais um atributo para cada nó interno: o **NEntries**, que é o número de entradas na sub-árvore de cada representante. O algoritmo de **ocupação mínima** seleciona o nó

que tenha a menor ocupação baseado no atributo **NEntries**. Esse algoritmo é o padrão para a *Slim-tree*, gerando árvores com melhor aproveitamento da taxa de ocupação.

Outra contribuição da *Slim-tree* é um novo algoritmo para o particionamento de elementos. Esse algoritmo é baseado na árvore geradora mínima (MST - *Minimum Spanning Tree*). Ele consiste em obter a árvore geradora mínima para os elementos e remover a maior aresta (aresta com maior distância entre um par de elementos). Na figura 3.7 (a) pode-se ver um exemplo de um nó que necessita ser dividido. A árvore geradora mínima é criada em (b) e então é removida a aresta com a maior distância e escolhidos dois novos representantes em (c). O algoritmo baseado na MST possui complexidade $\Theta(n^2 \cdot \log n)$.

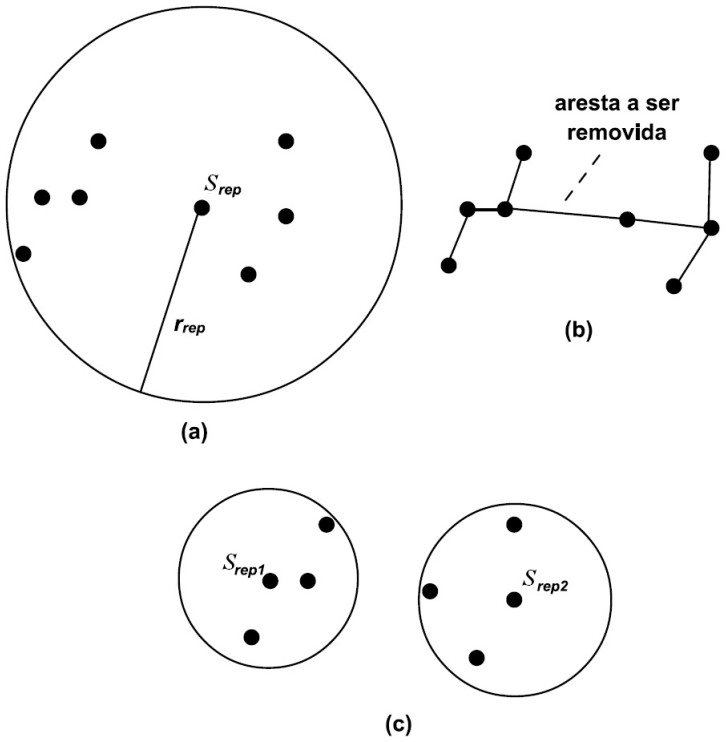

Figura 3.7: Algoritmo de particionamento baseado na árvore geradora mínima (MST - *Minimum Spanning Tree*) para escolha de dois novos representantes: (a) nó a ser particionado; (b) árvore geradora mínima, em destaque a aresta a ser removida e em (c) os dois novos subconjuntos representado por s_{rep1} e s_{rep2}.

Um dos algoritmos mais importantes apresentados em [Traina Jr. et al., 2000] foi uma forma de medir o grau de sobreposição de uma estrutura métrica, conhecido como *Fat Factor*. Essa função considera consultas por abrangência para estimar o grau de sobreposição de uma árvore, visto que, consultas por vizinhos mais próximos podem ser consideradas como um caso especial de consultas por abrangência [Berchtold et al., 1998b]. Dessa forma, o *Fat Factor* pode ser definido por:

$$fat(I_T) = \frac{(P_c - H \cdot N)}{N} \cdot \frac{1}{M - H} \tag{3.3}$$

onde T é uma árvore métrica com altura H e M nós, P_c é o número total de acessos

a nós necessários para responder uma consulta pontual para cada um dos N elementos indexados na árvore. Essa função possui valores entre 0 e 1, sendo que, quanto mais próximo de 0, menor é o grau de sobreposição.

Para tentar diminuir o valor do *Fat Factor* na *Slim-tree*, diminuindo, assim, o grau de sobreposição, foi proposto o algoritmo *Slim-down*. Esse algoritmo tenta reorganizar os nós nas regiões onde ocorre a sobreposição. Os elementos são transferidos de um nó origem para um nó destino sempre que há diminuição do raio do nó origem sem aumento do raio do nó destino, conforme mostrado na figura 3.8. Esse processo é feito até que não haja mais elementos para serem transferidos ou com sobreposição. Com isso, o *Slim-down* tenta minimizar o efeito negativo da sobreposição nodal. Esse é um processo caro e só é recomendado quando o *Fat Factor* ultrapassa um dado limiar.

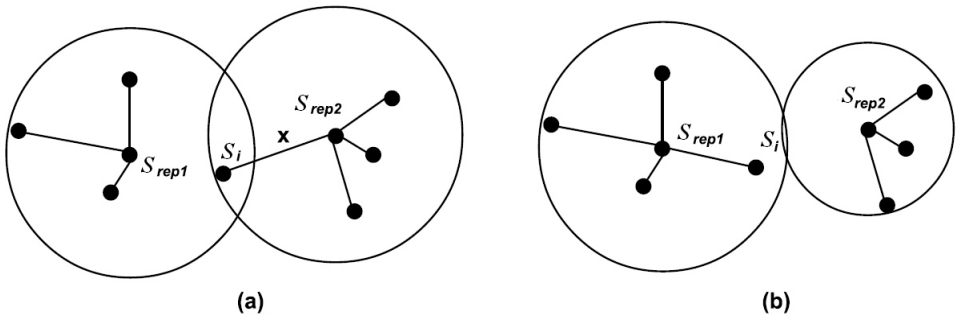

Figura 3.8: Exemplo do algoritmo *Slim-down* em ação: (a) dois nós com sobreposição do elemento s_i; (b) distribuição dos elementos após a execução do algoritmo *Slim-down*.

3.4.4 Família-*OMNI*

A família *OMNI* propõe uma técnica para geração de coordenadas para todos elementos do conjunto utilizando um conjunto de elementos representantes, chamados *omni-foci*, e cada representante individualmente é conhecido como foco [Santos et al., 2001]. O conceito *OMNI* consiste em escolher da melhor forma possível um conjunto de elementos $f_i \in S$ e torná-los globais para qualquer elemento indexado. Esse conjunto é conhecido como *omni-foci* base indicado como $F = \{f_1, \cdots, f_{D_f}\}$ e é utilizado para o cálculo das coordenadas *omni*. As coordenadas são obtidas com o cálculo de distância em relação aos *omni-foci*. Com isso, pode-se indexar qualquer elemento em uma MAM, MAMd ou MASI, gerando uma família de MAM [Traina Jr. et al., 2006].

A escolha dos representantes globais é importante para reduzir a quantidade de falsos positivos e aumentar a capacidade de poda (Figura 3.9). Em [Santos et al., 2001] é definida a escolha de $\lceil D_f \rceil + 1$ elementos *omni-foci*, onde D_f é a dimensão de correlação fractal como uma aproximação da dimensão intrínseca do conjunto de dados. Idealmente, os *omni-foci* devem estar na periferia do conjunto e igualmente distantes entre si. Para

isso, dados k *omni-foci* escolhidos, o próximo *focus* será o elementos $s_i \in S$ que minimizar a seguinte equação:

$$e(s_i) = \sum_{j=1}^{k} |d(f_1, f_2) - d(f_j, s_i)| \tag{3.4}$$

onde f_1 é o elemento mais distante em relação a um elemento $s_i \in S$ qualquer, f_2 o elemento mais distante em relação a f_1, f_j é um *focus* e s_i é o candidato a ser um *focus*;

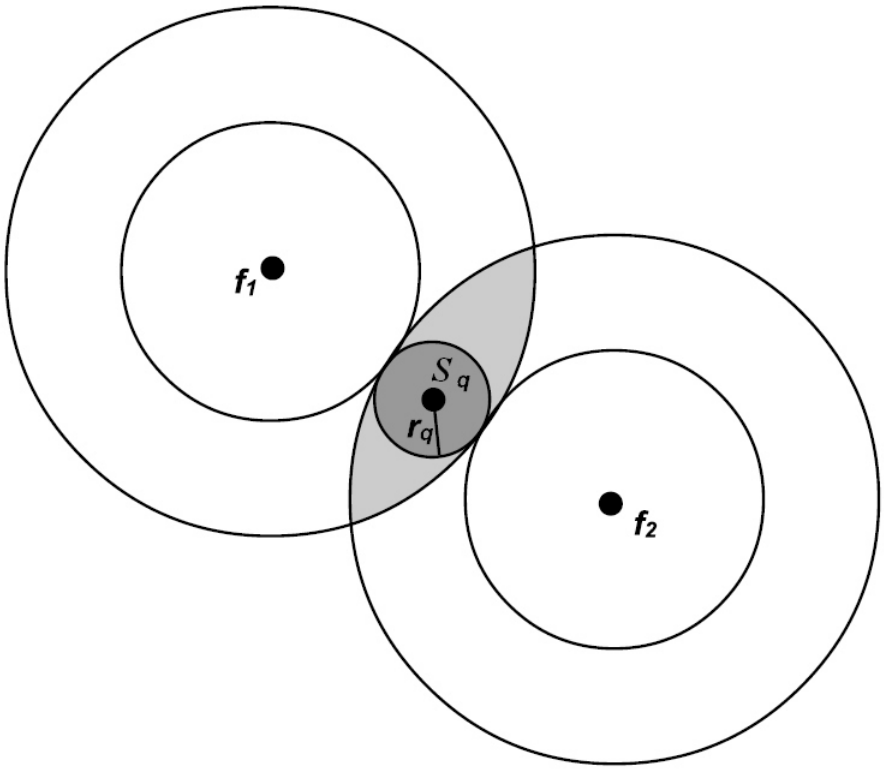

Figura 3.9: Uso do conceito *OMNI* para aumentar a capacidade de poda utilizando dois *omni-foci* como *omni-foci base*. A consulta $RQ(s_q, r_q)$ pode descartar os elementos fora da região escura sem nenhum cálculo de distância.

Com isso, o conceito *OMNI* pode ser aplicado a uma família de métodos de acesso, como por exemplo: busca seqüencial (*Omni-sequential*), com conjuntos de B^+-*trees* (*OmniB-tree*) e MAMd como a *R-tree* (*OmniR-tree*) [Santos et al., 2001]. Além disso, esse conceito pode ser utilizado em estruturas métricas para aumentar a capacidade de poda.

3.4.5 DF-tree

Semelhante à *Slim-tree*, a *DF-tree* (*Distance Field Tree*) é uma estrutura hieráquica, balanceada e multivias que utiliza as técnicas da *Slim-tree* associadas às técnicas *OMNI*. Por causa disso, a *DF-tree* possui desempenho melhor que a *Slim-tree* em relação ao número de cálculos de distâncias.

Conforme mostra a figura 3.10, a *DF-tree* utiliza as mesmas técnicas de inserção e divisão nodal da *Slim-tree*, mas com o uso do conceito *OMNI* escolhendo representantes globais além dos representantes de cada nó, para aumentar a capacidade de poda [Traina Jr. et al., 2002]. Note-se, no entanto, que os MAM da família *OMNI* utilizam os conceitos *OMNI* para gerar as coordenadas, e estas são indexadas, ou seja, a seqüência de operações é *OMNI* $\rightarrow$ indexação. Já a *DF-tree* usa tais conceitos de maneira invertida, ou seja, indexação $\rightarrow$ *OMNI*. Portanto, ao contrário da família *OMNI*, que pode ser utilizada com quaisquer métodos de acesso, a *DF-tree* está restrita a ser um método de acesso métrico. Esse trabalho também contribuiu com uma medida para estimativa de quantos cálculos de distâncias podem ser descartados, chamada de "*prunability*", que é útil para analisar a capacidade de poda dos vários níveis de um índice.

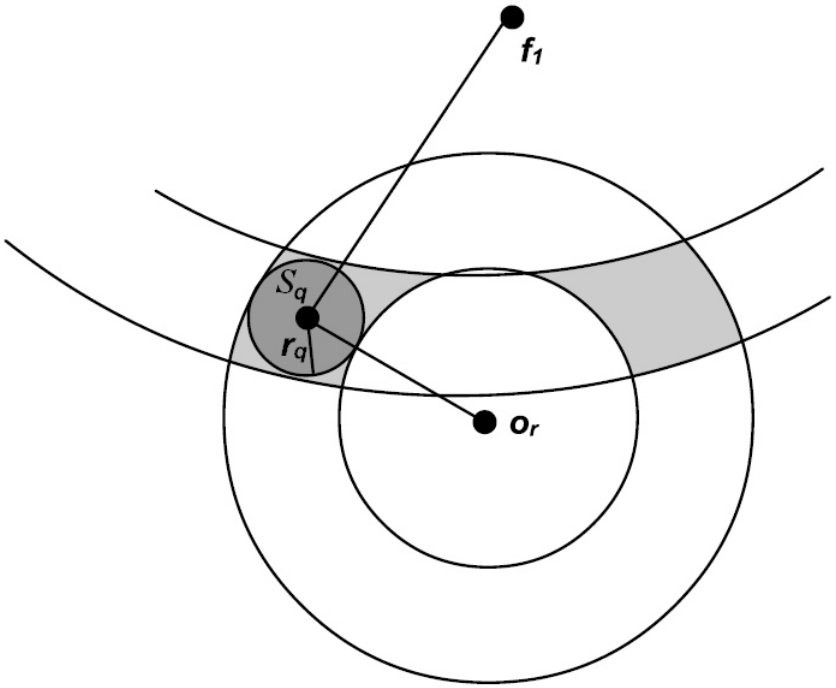

Figura 3.10: *DF-tree* utilizando o conceito *OMNI* para aumentar a capacidade de poda no espaço métrico. A consulta fica restrita apenas à região cinza escura.

3.4.6 DBM-tree

A *DBM-tree* (*Density-Based Metric Tree*) é um método de acesso métrico dinâmico não balanceado, permitindo que sub-árvores de regiões onde existe maior concentração de elementos se tornem mais profundas. Essa flexibilização na estrutura permite que haja uma minimização do grau de sobreposição em regiões mais densas [Vieira et al., 2006]. Com isso, há uma melhoria no desempenho, principalmente em cálculos de distâncias.

A profundidade das regiões na árvore pode ser controlada para que não haja um aumento excessivo na estrutura, prejudicando o desempenho em relação a acessos a disco. Assim como na *Slim-tree*, existe um algoritmo de reorganização da estrutura que permite otimizá-la. Esse algoritmo é conhecido como *Shrink*. Apesar dessa otimização, esse algoritmo possui um custo relativamente alto e, portanto, só deve ser executado em ocasiões de baixo uso da estrutura.

3.5 Considerações Finais

Os MA são técnicas que criam estruturas de indexação e são utilizados para melhorar o desempenho da busca em bases de dados. Em domínios onde há a ROT, o conjunto de elementos pode ser dividido em duas partes disjuntas. Com isso consegue-se descartar (podar) regiões irrelevantes para a consulta, restringindo a busca a um único caminho da árvore.

Já em estruturas que tem sobreposição nodal, a busca ocorre em mais de um caminho da árvore e a poda é feita utilizando algumas propriedades intrínsecas do domínio. Em domínios multidimensionais, utilizam-se as propriedades geométricas para realizar a poda. Em domínios métricos, a desigualdade triangular exerce uma importância fundamental para otimização de consultas. Com isso, esses MA tentam reduzir o número de acesso aos nós utilizando a tentativa de poda de regiões que não são de interesse para a consulta a ser realizada.

Os MAMd que indexam somente pontos em um espaço são conhecidos como Métodos de Acesso Espaciais Pontuais (MAMdP) e utilizando algumas técnicas como transformações, sobreposição, corte ou camadas múltiplas podem também indexar regiões, esses métodos são conhecidos como Métodos de Acesso Espaciais Não Pontuais (MAMdNP).

Existem também MAM estáticos e dinâmicos. Os estáticos fazem a indexação em memória e em uma única operação. Os dinâmicos, que são os mais importantes para um SGBD, organizam a estrutura em nós (páginas em disco) e permitem o armazenamento e recuperação em disco. Os mais importantes são a *M-tree*, a *Slim-tree*, a *DF-tree*, a *DBM-tree* e a família *OMNI* de MAM.

Para a indexação, existe a inserção elemento a elemento e utilizando operações de carga-rápida. No próximo capítulo serão descritos as operações de carga-rápida existentes,

com foco nos MAM dinâmicos.

Capítulo

4

Operações de Carga-rápida

4.1 Considerações Iniciais

Os MA podem ser construídos de duas maneiras: elemento a elemento (*tuple loading*) ou por uma única operação utilizando todo o conjunto de dados em uma única operação. Essa operação é conhecida como carga-rápida dos dados (*bulk loading*, no termo em inglês).

As operações de carga-rápida tentam utilizar as propriedades do espaço de indexação, juntamente com algumas formas de otimização, construindo a estrutura de indexação melhorada e de forma mais rápida em relação a inserção um a um, pois todo o conjunto de dados está disponível e sua distribuição é conhecida *a priori*. Em métodos com ROT, a operação de carga-rápida pode ser feita de maneira simples, ordenando e carregando os dados de maneira direta e, com isso construindo os níveis superiores sem grandes dificuldades. Já em métodos que não possuem ROT, são necessárias outras técnicas para realizar as operações de carga rápida, como a ordenação utilizando linearização do espaço, *buffers* ou agrupamento utilizando amostras do conjunto de dados.

As próximas seções descrevem as três técnicas existentes para a realização da operação de carga-rápida: as baseadas em ordenação dos dados (*sort-based bulk loading*), as baseadas em buffers (*buffer-based bulk loading*) e as por amostragem dos dados (*sample-based bulk loading*). Finaliza-se apresentando algoritmos genéricos específicos para MAM.

4.2 Carga-Rápida em Métodos com ROT

A maioria dos algoritmos para estruturas que suportam ROT é baseada em ordenação dos dados e os algoritmos são conhecidos como *sort-based bulk loading*. Como exemplo, em uma B^+*-tree*, os elementos são ordenados e a estrutura é criada de baixo para cima (*bottom-*

up). Na execução do algoritmo de carga-rápida, define-se um novo nó raiz, com o primeiro ponteiro à esquerda apontando para o primeiro nó processado após a ordenação (Figura 4.1). Em seguida, cada nó é inserido no ponteiro à direita que ainda não aponta para nenhum nó. Quando o nó raiz está cheio, é feita a divisão do nó da maneira tradicional, criando um novo nó raiz, e assim a árvore cresce um nível. Apenas os nós do caminho à direita são mantidos em memória principal.

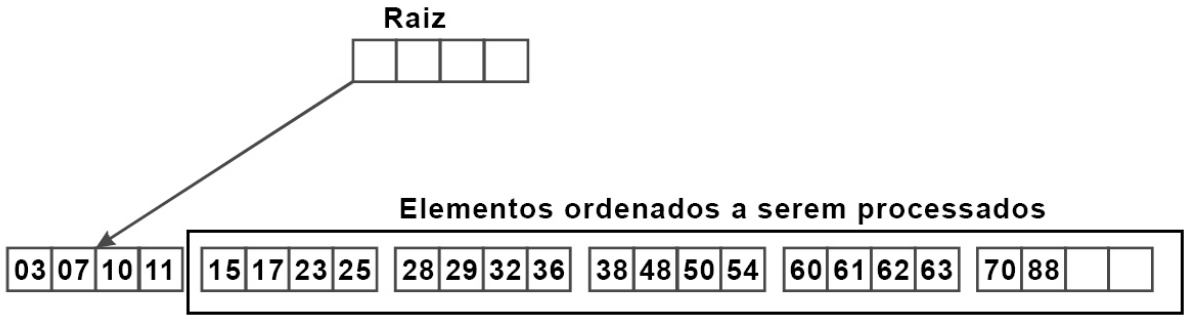

Figura 4.1: Operação de carga rápida em uma B^+-*tree*. Os elementos são ordenados e uma raiz é criada. A seguir, os nós em disco são inseridos à direita da estrutura.

Apesar de apenas manter o mesmo desempenho nas consultas que a inserção seqüencial, esse algoritmo de carga-rápida dos dados possui complexidade $\Theta(n)$, enquanto que a inserção seqüencial possui complexidade $\Theta(n \cdot \log n)$, onde n é o número de elementos a serem inseridos [Vitter, 1998].

4.3 Carga-Rápida para Dados Não Tradicionais

A operação de carga-rápida é bem conhecida para métodos como as *B-trees*, mas os algoritmos tradicionais de carga-rápida não são aconselháveis ou, até mesmo, não podem ser aplicados, pois não é possível aplicar a ROT para auxiliar na criação dessas estruturas. Além disso, em uma construção *bottom-up*, não há como prever a construção dos níveis superiores, o que torna difícil a operação de carga-rápida dos dados utilizando a ordenação dos dados da maneira convencional.

Algumas abordagens tentam linearizar o espaço de indexação utilizando curvas de preenchimento do espaço, para então aplicar a técnica de carga-rápida baseada em ordenação. Outras utilizam *buffers* para construir a estrutura sem sobrecarregar a memória principal e otimizar o acesso a disco, mas não levam em consideração o custo de processamento. Outra forma de realizar a carga-rápida é escolher amostras do conjunto a ser indexado, agrupar os elementos com relação à essas amostras e então construir a estrutura de cima para baixo (*top-down*).

As próximas seções descrevem as técnicas para operação de carga-rápida em MA sem ROT. Além disso, são descritos o algoritmo da *M-tree* e os algoritmos genéricos para MAMd e MAM encontrados na literatura.

4.3.1 Carga-Rápida Baseadas em Ordenação dos Dados

A execução da técnica de ordenação em alguns MA não tradicionais necessita que os dados sejam ordenados. Para isso, existem algumas técnicas conhecidas como técnicas de ordenação do espaço.

Uma técnica de ordenação consiste em, de alguma forma, ordenar o espaço em questão, linearizando-o para aplicação das propriedades da ROT. A mais tradicional técnica de ordenação para dados multidimensionais é o de curvas de preenchimento do espaço. Uma linha é traçada pelo espaço até preenchê-lo totalmente. Isso permite diminuir a dimensão para um sistema de dimensão um. Entre os exemplos de curvas de preenchimento do espaço existem: a *Z-order* (curva de *Peano*), a *Row Wise*, a curva de *Hilbert* e a *G-Gray* [Faloutsos & Roseman, 1989]. Entre essas, a que provê melhor desempenho em consultas é a curva de *Hilbert* [Gaede & Günther, 1998]. A figura 4.2 mostra exemplos de curvas de preenchimento do espaço.

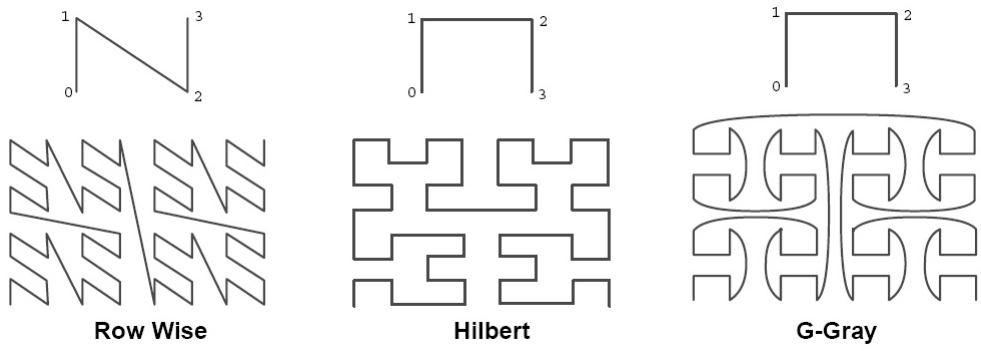

Figura 4.2: Curvas de preenchimento do espaço: *Row Wise*, *Hilbert* e *G-Gray*. Acima, ordem um. Abaixo, ordem três.

Outra forma de reduzir a dimensão é usar um algoritmo baseado em uma forma especial de particionamento do espaço. É conhecido como técnica da pirâmide [Berchtold et al., 1998b]. A idéia básica é primeiramente dividir o espaço de dados em pirâmides bidimensionais partilhando o ponto central do espaço como topo, conforme apresentado na figura 4.3. Em um segundo passo, cada pirâmide é dividida em partes paralelas à base. Após isso, valores são atribuídos, seguindo um critério, aos pontos de cada partição. Esses valores são conhecidos como *pyramid value*.

O problema é que não existe uma ordem total entre os elementos não convencionais que preserve a proximidade espacial em nenhum dos métodos de redução de dimensionalidade [Gaede & Günther, 1998]. Não há como ordenar esses elementos da mesma maneira que é feito com elementos em domínios com ROT. Essa característica inviabiliza o uso de operações de carga-rápida da maneira tradicional. Para resolver esse problema, foram criadas duas técnicas que utilizam as propriedades do conjunto para construir a estrutura

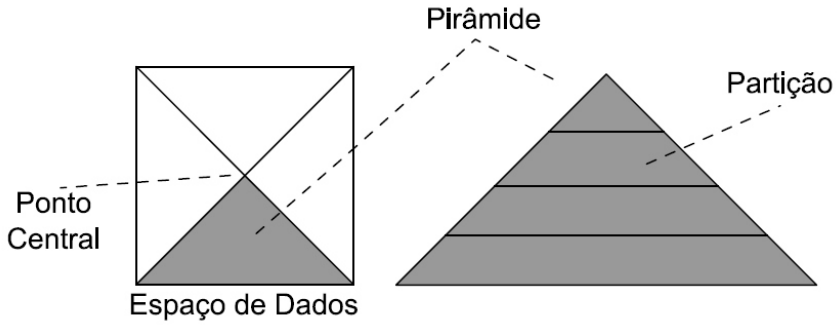

Figura 4.3: Técnica da Pirâmide. O espaço de dados é divido em pirâmides partilhando um ponto central e essas pirâmides são divididas paralelamente à base.

utilizando algoritmos de carga-rápida. A próxima seção descreve essas duas técnicas.

4.3.2 Carga-Rápida Baseadas em Buffer e por Amostragem

Em adição aos métodos baseados em ordenação, existem duas outras formas para fazer a carga-rápida: baseando-se no uso de *buffers* e por amostragem dos dados. A primeira forma, também conhecida como *buffer-based bulk loading*, utiliza a técnica da *Buffer-tree* [Arge, 2003]. Essa técnica pode ser utilizada para manter a eficiência de algoritmos de otimização que funcionam apenas em memória principal. O problema é que o seu objetivo é apenas reduzir o custo de acesso ao disco, ignorando o custo de processamento [Ciaccia & Patella, 1998]. Um dos trabalhos que utiliza a técnica de *buffer-tree* é o algoritmo genérico para carga-rápida de índices multidimensionais, que pode ser adaptado a domínios métricos, proposto por [Bercken & Seeger, 2001].

A técnica baseada em amostragem, ou *sample-based bulk loading*, utiliza um subconjunto, que possa ser colocado em memória, de elementos representantes escolhidos do conjunto a ser indexado [Lang & Singh, 2001]. A partir disso, inferem-se algumas propriedades desse conjunto para a construção de toda a estrutura.

Na literatura, encontram-se alguns algoritmos para operações de carga-rápida em MAMd, como para as *R-tree* e suas variantes [Berchtold et al., 1998a] [Arge et al., 2002] [Lin & Su, 2004] e algoritmos genéricos para diversos tipos de MA classificados em duas categorias: *GP-trees - Grow and Post trees* e *OP-trees - Overlapping Predicate trees*. Esse algoritmos são considerados genéricos pois eles trabalham sobre interfaces definidas e que devem existir no MA utilizado. A segunda categoria (descrita na seção 4.3.4) é composta por uma sub-classe da *GP-trees* e está inclusa a maioria dos MA que possuem sobreposição [Bercken & Seeger, 2001]. Em [Ghanem et al., 2004] são propostos algoritmos baseados no particionamento de espaço para alocação de regiões que aproveitam a disponibilidade de *buffers*.

Sem considerar o algoritmo genérico da *GP-trees*, existem apenas dois algoritmos de operação de carga-rápida em espaços métricos: o algoritmo da *M-tree* e uma técnica

para realizar a persistência da estrutura em disco, baseada no uso de *buffers* [Leal, 2001].

4.3.3 Carga-Rápida na *M-tree*

Um algoritmo de carga-rápida utilizando a técnica baseada em amostragem dos dados foi desenvolvido para a *M-tree*[Ciaccia & Patella, 1998]. As sub-árvores, inicialmente não balenceadas, são construídas de maneira *top-down* e de acordo com a proximidade entre os elementos. Após essa primeira etapa, é feito o balanceamento para tentar gerar uma *M-tree* completa e balanceada.

Esse algoritmo pode ser descrito da seguinte maneira: Seja $S = \{s_1, \cdots, s_n\}$ o conjunto dos dados a serem indexados. Inicialmente, escolhe-se aleatoriamente uma amostra de k elementos $\{s_{f1}, \cdots, s_{fk}\}$ de S, inserindo em um conjunto de amostras F. Associam-se, então, cada elemento de S à amostra mais próxima, produzindo k conjuntos $\{F_1, \cdots, F_k\}$. Cada nó representante, é então associado a um sub-conjunto. Isso é feito recursivamente até que todos os elementos sejam inseridos. Depois dessa etapa, tem-se uma árvore não balanceada.

O particionamento do conjunto de dados depende fortemente da escolha da amostra: amostras em regiões menos densas deverão produzir sub-árvores mais baixas, uma vez que os conjuntos correspondentes terão cardinalidades mais baixas, enquanto amostras em regiões densas deverão produzir sub-árvores mais altas (Figura 4.4).

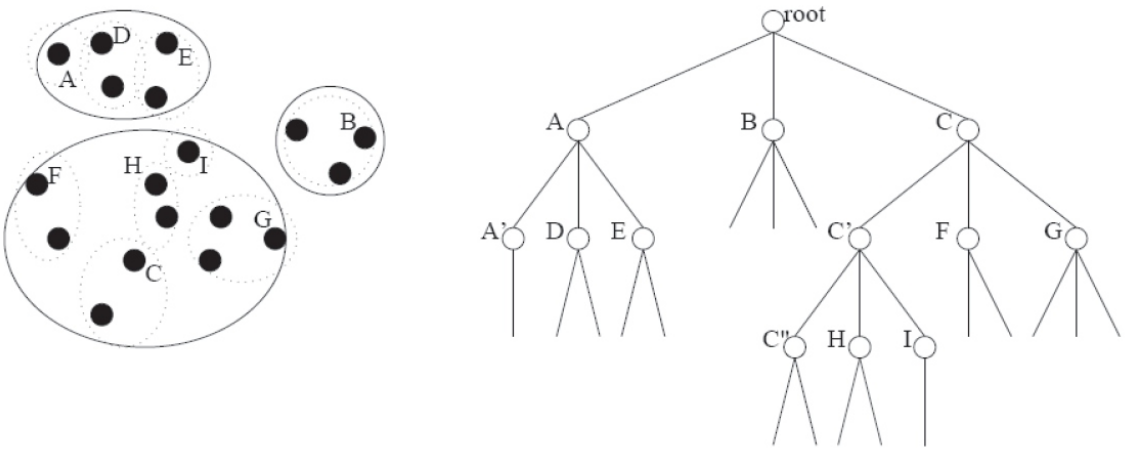

Figura 4.4: Exemplo do algoritmo de carga-rápida para *M-tree*. Desbalanceamento da estrutura: amostras em regiões mais densas possuem sub-árvores mais altas [Ciaccia & Patella, 1998].

Para balancear a estrutura podem ser utilizadas duas técnicas:

1. associam-se novamente os elementos dos conjuntos não preenchidos em outros conjuntos e elimina-se a amostra correspondente de F; ou

2. quebram-se as sub-árvores maiores, obtendo um número de sub-árvores menores. As raízes das sub-árvores obtidas serão inseridas no conjunto amostra F, substituindo a amostra original.

Com isso, pode-se ter uma *M-tree* completa e balanceada, construída de maneira mais rápida que a inserção elemento a elemento. O problema é que esse algoritmo não garante a ocupação mínima e pode produzir árvores com apenas um único representante e conjuntos com apenas um único elemento em vários nós. Quando isso ocorre, é necessário repetir todo processo do começo escolhendo um novo conjunto de amostras. Esse processo pode ser bastante custoso e há chances de o algoritmo nunca produzir uma *M-tree* válida, portanto esse algoritmo de carga-rápida não pode ser utilizado como parte de um SGBD.

4.3.4 Carga-Rápida Genérica Para Estruturas com Sobreposição

Em [Bercken & Seeger, 2001] foram propostos algoritmos genéricos para diversos tipos de MA classificados em duas categorias: *GP-trees* (*Grow and Post trees*) e *OP-trees* (*Overlapping Predicate trees*). A primeira categoria é dedicada a MA que possuem algoritmos de escolha de sub-árvores e algoritmos de divisão que trabalham de forma recursiva, enviando informações sobre a divisão como retorno. Para isso, o algoritmo assume que as estruturas tenham uma interface com as seguintes funções:

- **chooseSubtree** - Recebe como entrada um elemento e o seu nó índice. Como resultado, é retornada uma referência à sub-árvore na qual o elemento de entrada deverá ser inserido.
- **grow** - Recebe como entrada um elemento e um nó, inserindo o elemento neste nó.
- **split & post** - Recebe como entrada um nó com um elemento a mais do que a sua capacidade máxima. Então, divide o nó em outros dois e envia informações sobre a divisão como, por exemplo, uma nova referência ao nó pai criado.
- **search** - Recebe como entrada um tipo de consulta e um nó. Retorna todos os elementos armazenados no nó relevantes à consulta.

Com essas informações, a inserção é feita por nível utilizando *buffers*. Quando todo o conjunto cabe em memória a estrutura é criada de uma única vez. Caso contrário, cada elemento é inserido utilizando o método **chooseSubtree** recursivamente até chegar em um nó folha. A seguir, o elemento é inserido em um *buffer* associado ao seu respectivo nó folha (Figura 4.5). Quando os elementos não cabem mais em memória, são criados novos nós com os elementos que estão nos *buffers* e são invocados os outros métodos para contruir a estrutura válida.

O algoritmo descrito anteriormente é conhecido como *Path-based Bulk Loading*. Sua maior desvantagem está na construção de estruturas que possuem sobreposição, pois o método **chooseSubtree** pode não ficar restrito a um único caminho. Outro problema é que o algoritmo ignora o custo de processamento intrínseco ao método **chooseSubtree**. No caso médio de inserção, os elementos são igualmente distribuídos nos *buckets*, a complexidade é de $\Theta(n \cdot \log_m n)$, onde n é o número de elementos a serem inseridos e m é o número de *buckets* em memória. No pior caso, onde todos os elementos são inseridos em um único *buffer*, a complexidade é de $\Theta(n^2/m)$.

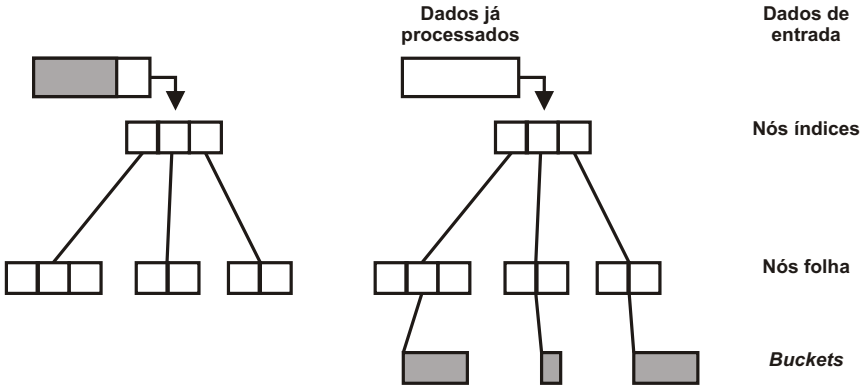

Figura 4.5: Algoritmo de carga-rápida *Path-based Bulk Loading* em ação. Na primeira parte a árvore é construída até não caber todos os dados em memória, a partir daí é feita a separação em *buckets* utilizando o método *chooseSubtree*.

Além dos MA englobados por esse algoritmo, existe outra categoria: as *OP-trees*. A categoria das *OP-trees* é uma sub-categoria das *GP-trees*. Nela estão inclusos os MA que possuem sobreposição, como por exemplo, a *M-tree*, a *Slim-tree* e a *R-tree*. O algoritmo para execução de carga-rápida genérica para essas estruturas é conhecido como *Quickload*.

A idéia básica do *Quickload* é retirar uma amostra de elementos dividindo-os em partições. O algoritmo é então aplicado de maneira recursiva para cada partição. Se toda a entrada couber em memória principal, as páginas em memória interna correspondem aos nós folha da estrutura alvo. Caso contrário, o algoritmo processa os nós até que a memória esteja cheia. Depois são criados *buckets* para armazenamento temporário e anexados aos nós folha. Dessa forma, as inserções e distribuições dos elementos utilizando o algoritmo de escolha do nó de inserção (**chooseSubtree**) são realizadas nesses *buckets*. Apesar de efetuar as atualizações dos ponteiros, a estrutura não será modificada enquanto os elementos estiverem sendo distribuídos entre os *buckets*.

Quando todos os elementos da entrada inicial forem tratados, podem ocorrer dois casos: *bucket* vazio ou não. Se existir um *bucket* vazio, o nó folha que possui esse *bucket* associado é inserido em disco, liberando espaço em memória. Por outro lado, se o *bucket* não estiver vazio, um par, consistindo em um ponteiro para a folha correspondente e um ponteiro para a estrutura de armazenamento, é inserido em uma lista para processamento

do algoritmo.

O algoritmo é então aplicado recursivamente utilizando os elementos dessa lista. Quando a lista torna-se vazia, têm-se os ponteiros para os nós folha da estrutura. Esses ponteiros são utilizados pelo *Quickload* para a criação do próximo nível. Quando houver apenas um ponteiro nesta lista, a execução do algoritmo chegou ao fim, ou seja, existe apenas o ponteiro para o nó raiz da estrutura.

A Figura 4.6 ilustra a execução do algoritmo para uma capacidade máxima de três nós em memória depois que os elementos de entrada foram inseridos nos buckets correspondentes. Na figura superior podemos ver que a árvore é criada a partir dos elementos de entrada $E_1, \cdots, E_{12}$. A folha que corresponde ao elemento S_3 já pode ser armazenada em disco, pois seu *bucket* está vazio (figura inferior). No entanto, o algoritmo tem que ser aplicado mais uma vez para os outros nós folha.

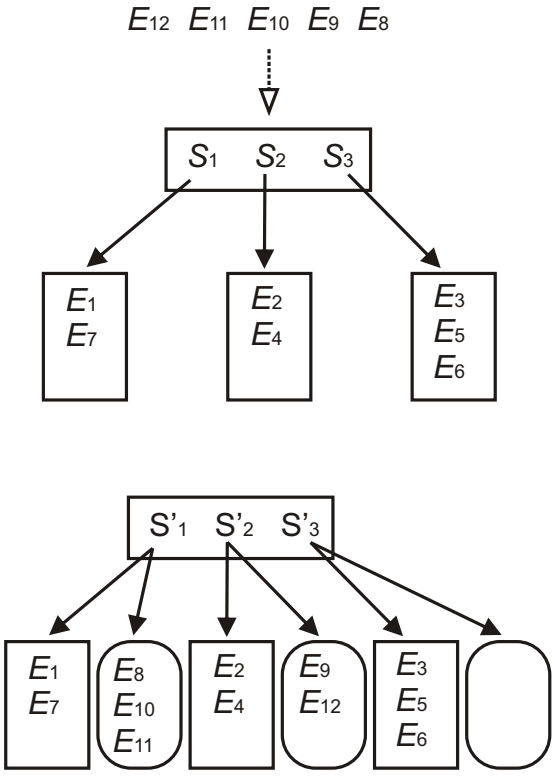

Figura 4.6: Quickload para no máximo três nós em memória. Retângulos arredondados são os *buckets*. Como o *bucket* associado ao nó folha S_3 está vazio, já pode ser realizado o armazenado em disco deste nó

Esses dois algoritmos não garantem a ocupação mínima do nó e, por essa razão, constróem estruturas com um desempenho muito menor em relação à estrutura construída com uma inserção seqüencial. Além disso, não é levado em consideração o custo de cálculos de distâncias. Dessa forma, não é viável a implementação desses algoritmos em SGBD.

4.4 Considerações Finais

É importante ressaltar que os MAM dinâmicos sempre apresentam algoritmos para implementar a operação de inserção individual de dados. Porém, quase nenhum tem tratado o caso das operações de carga-rápida dos dados. Esses tipos de operações são bem conhecidos em MA tradicionais.

Restauração de backups, criação posterior ou otimização do índice são exemplos importantes de uso dessas operações. Elas permitem explorar melhor as propriedades do conjunto e a distribuição para acelerar a construção da estrutura e melhorar o desempenho das operações de busca.

Na literatura existem poucos algoritmos que tentam realizar a carga rápida dos dados em espaços sem a ROT. Para espaços métricos existe o algoritmo de carga-rápida da *M-tree* e o algoritmo *Quickload*. O algoritmo de carga-rápida da *M-tree* utiliza amostragem dos dados e constrói uma estrutura não balanceada na etapa intermediária do algoritmo, para posteriormente organizá-la para obter uma *M-tree* válida. O *Quickload* também utiliza amostragem dos dados com utilização de *buffers* para a construção da estrutura, mas, por não ser específico para dados métricos, não leva em consideração o custo de processamento nos cálculos de distância. Esses dois algoritmos de carga-rápida não garantem a ocupação mínima do nó e podem ocasionar problemas na tentativa de balancear a estrutura ou afetar o desempenho de busca.

O próximo capítulo irá mostrar como solucionar esses problemas utilizando o conhecimento do conjunto de dados e, a partir daí, calculando as propriedades da estrutura a ser construinda pela operação de carga-rápida.

Capítulo

5

Carga-rápida na Slim-tree

5.1 Considerações Iniciais

A maioria dos MAM visa reduzir o número de acessos a disco e cálculos de distâncias em suas operações de buscas e inserções. Para isso, se faz necessário o conhecimento dos dados e suas características intrínsecas. Na inserção elemento a elemento, não há como ter esse conhecimento, sendo que otimizações, como o *Slim-down* só são feitas quando já existe uma boa quantidade de dados. Na carga-rápida, como todos os dados já são conhecidos, pode-se ter uma análise prévia das características do conjunto, inserindo blocos de dados na estrutura de maneira otimizada, melhorando a distribuição dos elementos para uma redução do tempo de consulta.

Neste capítulo são apresentadas as características e o funcionamento dos algoritmos de carga-rápida desenvolvidos para o MAM *Slim-Tree*, que são as principais contribuições deste trabalho. Na primeira parte é apresentado o conceito dos algoritmos desenvolvidos e a seguir é feita a descrição das três abordagens utilizadas:

- **Nós de Tamanho Fixo** - Descrita na seção 5.3.
- **Nós de Tamanho Fixo por Nível** - Descrita na seção 5.4.
- **Nós de Tamanho Controlado** - Descrita na seção 5.5.

5.2 Conceito dos Algoritmos

O algoritmo inicial proposto constrói a estrutura de índices de cima para baixo, utilizando a técnica de amostragem. Entretanto, diferentemente do algoritmo de carga-rápida da *M-*

tree, a estrura é construída e mantida balanceada desde o começo do algoritmo, limitando o número de elementos associados à cada amostra obtida.

Para isso, a idéia principal é considerar que uma árvore pode ser descrita pelo número de nós, número de elementos por nó e lista de elementos em cada nó. Os algoritmos existentes na literatura trabalham com os dados para produzir a lista de elementos em cada nó, obtendo os outros dados como conseqüência dessas listas.

A proposta deste algoritmo é estimar o número de nós na árvore e o número de elementos em cada nó antes de associar a sua respectiva lista. Dessa forma, a estrutura é mantida balanceada em todas as etapas do algoritmo, distribuindo os elementos nesses nós de acordo com os valores obtidos [Vespa et al., 2007].

Por essa razão, o primeiro passo do algoritmo é determinar o número de nós que irão compor a estrutura final. Os parâmetros utilizados nesse passo do algoritmo são o número N de elementos a serem indexados, a capacidade máxima C_M dos nós e a capacidade mínima C_m a ser garantida. Para chegar nessa solução, foram utilizadas três abordagens: nós fixos em toda árvore, nós fixos por nível e nós de tamanho controlado.

5.3 Carga-Rápida com Nós de Tamanho Fixo

A idéia principal da abordagem de nós de tamanho fixo é construir toda a estrutura partindo do topo até o seus nós folha (abordagem *top-down*) utilizando a mesma quantidade de elementos para todos os nós que irão compor a estrutura final. A primeira etapa do algoritmo é escolher como amostra inicial um número fixo de elementos que irão compor o nó pai. Na segunda etapa, para cada nó filho escolhe-se o mesmo número de elementos, sempre respeitando a proximidade em relação a todos os elementos da amostra. Isso é feito até não se ter mais elementos a inserir, sendo que na última etapa serão construídos os nós folha, dividindo igualmente o número de elementos por nó. Ao manter fixo o número de elementos por nó, a estrutura criada será mantida balanceada desde o começo do algoritmo e sempre com a taxa de ocupação desejada.

Esse algoritmo de carga-rápida (algoritmo 5.3.1) inicia-se com a construção de uma estrutura vazia e passando como argumento uma referência para o conjunto de dados (I), um valor nulo como representante (r) e a capacidade máxima do nó (C_M) a ser considerada. Se o número de elementos do conjunto for menor que a capacidade máxima (linha 1), então todos os elementos são inseridos em um único nó (linha 2). Se o representante é nulo (linha 23), então a raíz da árvore é criada (linha 24). Caso o número de elementos a ser inserido seja maior que C_M, são escolhidas C_M amostras do conjunto inicial (linha 4) que serão associadas a conjuntos (linha 5). A partir daí, todos os elementos do conjunto I são distribuídos nos conjuntos associados aos representantes que possuem distância mínima em relação ao representante (linha 8). Se o conjunto chegar em sua capacidade máxima (linha 13) então ele é inserido no conjunto K (linha 14). Se o elemento

for associado a uma amostra que possua um conjunto cheio (linha 9 e 10), então ele será inserido nessa amostra e o elemento mais distante em relação a amostra associada será removido desse conjunto e inserido em outro (linha 11). Isso é feito até que esse elemento mais distante seja inserido em um conjunto que não esteja cheio (linha 10). A seguir, as amostras são inseridas em um nó (linha 16) e seu representante é criado (linha 17). Os conjuntos que não estavam cheios serão inseridos no conjunto K (linha 18 a 20), sendo que o processamento será feito recursivamente (linha 21 e 22) para que todas as outras sub-árvores sejam criadas.

Algoritmo 5.3.1 *BulkLoadSlimFixedNode(I,r,C_M)* - Carga-rápida com nó de tamanho fixo

Entrada: I: conjunto de elementos a serem inseridos
r: representante da sub-árvore
C_M: ocupação máxima do nó
Saída: Inserir todos os elementos $i_n \in I$ com representante r

1: **se** $\|I\| \leq C_M$ **então**
2: insere todos os elementos $i_n \in I$ no nó folha N_i e atribui r como o representante desse nó
3: **senão**
4: escolhe $k = C_M$ amostras $s_1 \dots s_k$ do conjunto I
5: cria k conjuntos vazios $S_1 \dots S_k$ e associa cada conjunto com a sua respectiva amostra (representante do conjunto) $L = \{\{s_1, S_1\} \dots \{s_k, S_k\}\}$
6: cria um conjunto vazio $K = \{\}$
7: **para** $p = 0$ até $\|I\|$ **faça**
8: atribui i_p ao conjunto S_j, onde s_j é a amostra mais próxima do elemento i_p associado ao S_j
9: $V \leftarrow \{s_j, S_j\}$
10: **enquanto** $V \in K$ **faça**
11: seja $V = \{s_p, S_p\}$, o elemento mais distante de S_p será inserido em S_m, onde $m \neq p$ e s_m é o elemento mais próximo de s_p, $\forall \{s_m, S_m\} \in L$
12: $V \leftarrow \{s_m, S_m\}$
13: **se** $\|S_j\| \geq \frac{\|I\|}{C_M}$ **então**
14: insere $\{s_j, S_j\}$ no conjunto K
15: remove $\{s_j, S_j\}$ do conjunto L
16: insere $s_1 \dots s_k$ no nó N_i
17: atribua r como o representante do nó
18: **para** $p = 0$ até $\|L\|$ **faça**
19: insere $\{s_p, S_p\}$ no conjunto K
20: remove $\{s_p, S_p\}$ do conjunto L
21: **para** $m = 0$ até $\|K\|$ **faça**
22: *BulkLoadSlimFixedNode(S_m, s_m, C_M)*
23: **se** $r =$ nulo **então**
24: atribua N_i como raiz da árvore

O grande problema dessa abordagem é que a construção partindo de cima e com o número de elementos fixo por nó tende a produzir árvores com muitos nós-folha. Para

ilustrar o problema, suponha que seja necessário inserir $N = 101$ elementos em uma árvore com $C_M = 10$ elementos por nó (Figura 5.1). De forma *top-down*, no primeiro passo do algorimto serão atribuídos 10 elementos ao nó raíz. O segundo nível terá 10 nós com 10 elementos cada e o teceiro nível terá 99 nós folhas com um elemento cada e um nó folha com 2 elementos. A árvore será construída com 111 nós e muito espaço livre nos nós folha, mas nenhum espaço livre nos nós internos.

Se fosse possível obter a mesma abordagem seguindo a construção *bottom-up*, o algoritmo iria produzir um árvore com apenas 11 nós folhas: 2 nós com 10 elementos e 9 nós com 9 elementos. O segundo nível teria 2 nós: um com 5 e outro com 6 elementos e a raíz teria apenas 2 elementos. A árvore construída de baixo para cima teria apenas 14 nós, melhorando o aproveitamento de disco e o número de acessos necessários ao executar consultas nessas árvores. Essas duas alternativas são mostradas na Figura 5.1.

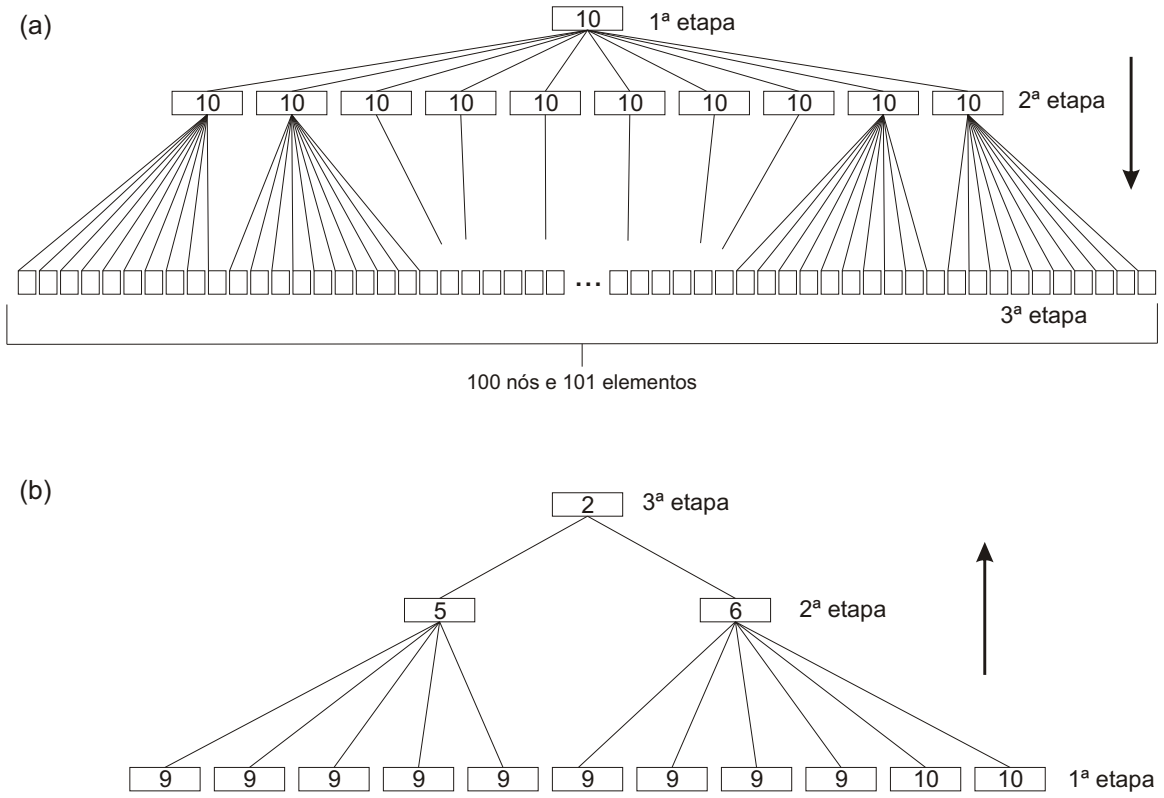

Figura 5.1: Carga-Rápida com Nós de Tamanho Fixo com $N = 101$ elementos a serem inseridos e com taxa da ocupação $C_M = 10$. Em (a) a estrutura é construída de cima para baixo e possui 111 nós. Em (b) a estrutura é construída de baixo para cima e possui apenas 14 nós e com o mesmo número de elementos contidos na estrura descrita em (a).

Apesar do baixo número de nós, a estrutura construída de baixo para cima não garante um bom desempenho e nem baixo número de cálculos de distância nas consultas, pois não há como prever a construção dos níveis superiores com bom agrupamento dos dados. Na maneira *top-down*, o agrupamento dos dados é sempre realizado, pois são pri-

meiro escolhidos os representantes para construção dos nós inferiores, mas isso não ocorre na construção *bottom-up*. Por essa razão, os próximos algoritmos que foram desenvolvidos constrõem a estrutura partindo de cima, mas escolhendo representantes para agrupar os elementos.

5.4 Carga-Rápida com Nós de Tamanho Fixo por Nível

A segunda abordagem segue as mesmas etapas da abordagem anterior, mas diferentemente, esse algoritmo fixa o número de elementos em cada nó considerando cada nível na árvore e não um número fixo para todos os nós. Para tentar obter o mesmo desempenho de acesso a disco da construção *bottom-up* descrito anteriormente e com o conhecimento de quantos elementos por nível se tem nessa construção, pode-se determinar o número de elementos que deve-se ter para um bom aproveitamento do disco.

Esse algoritmo descobre o número ideal de elementos por nó em cada nível, para se ter um melhor aproveitamento do espaço em memória secundária. Para isso, ele obtém o número de nós em cada nível e distribui igualmente os elementos nesses nós.

O número de nós em cada nível é calculado como $k = \left\lceil \frac{N}{(C_M)^{H-l}} \right\rceil$, onde H é a altura da árvore obtida por $H = \left\lceil \log_{C_M} N \right\rceil$ e l é o nível da árvore definido no capítulo 2.3. Conseqüentemente, o número de elementos em cada nó por nível é dado por: $\eta = \frac{\|I\|}{k}$, onde $\|I\|$ é o total de elementos a serem inseridos em cada passo do algoritmo.

O algoritmo de carga-rápida com nós de tamanho fixo por nível (algoritmo 5.4.1) inicia-se da mesma forma que o algoritmo 5.3.1. Na primeira etapa do algoritmo, se o número de elementos do conjunto for menor que a capacidade máxima (linha 1) todos os elementos são inseridos em um único nó (linha 2). Se o representante é nulo (linha 23) a raíz da árvore é criada (linha 24). Caso o número de elementos a ser inserido seja maior que C_M, são escolhidas $k = \left\lceil \frac{N}{(C_M)^{H-l}} \right\rceil$ amostras do conjunto inicial (linha 4). A partir daí, todos os elementos do conjunto I são distribuídos nos conjuntos associados à amostras levando em consideração a proximidade entre os elementos (linha 8). Se o conjunto chegar em sua capacidade máxima considerando o número de elementos por nível (linha 13) então ele é inserido no conjunto K (linha 14). É importante observar que essa abordagem leva a um melhor aproveitamento do espaço em disco, pois o algoritmo leva em consideração o número de elementos necessários para se obter o melhor aproveitamento do espaço.

Após essa atribuição, se o elemento for associado a uma amostra que possua um conjunto cheio (linha 9 e 10) ele será inserido nessa amostra e o elemento mais distante em relação a amostra associada será removido desse conjunto e inserido em outro (linha 11). Isso é feito até que esse elemento mais distante seja inserido em um conjunto que não esteja cheio (linha 10). A seguir, as amostras são inseridas em um nó (linha 16) e seu

representante é criado (linha 17). Os conjuntos que não estavam cheios serão inseridos no conjunto K (linha 18 a 20), sendo que o processamento será feito recursivamente (linha 21 e 22) para que todas as outras sub-árvores sejam criadas, da mesma maneira que o algoritmo anterior.

Algoritmo 5.4.1 *BulkLoadSlimFixedNodeByLevel(I,r,C_M)* - Carga-rápida com nó de tamanho fixo por nível

Entrada: I: conjunto de elementos a serem inseridos
r: representante da sub-árvore
C_M: ocupação máxima do nó
Saída: Inserir todos os elementos $i_n \in I$ com representante r

1: **se** $\|I\| \leq C_M$ **então**
2: insere todos os elementos $i_n \in I$ no nó folha N_i e atribui r como o representante desse nó
3: **senão**
4: escolhe $k = \left\lceil \frac{N}{(C_M)^{H-l}} \right\rceil$ amostras $s_1 \ldots s_k$ do conjunto I
5: cria k conjuntos vazios $S_1 \ldots S_k$ e associa cada conjunto com a sua respectiva amostra (representante do conjunto) $L = \{\{s_1, S_1\} \ldots \{s_k, S_k\}\}$
6: cria um conjunto vazio $K = \{\}$
7: **para** $p = 0$ até $\|I\|$ **faça**
8: associa i_p ao conjunto S_j, onde s_j é a amostra mais próxima do elemento i_p associado ao S_j
9: $V \leftarrow \{s_j, S_j\}$
10: **enquanto** $V \in K$ **faça**
11: seja $V = \{s_p, S_p\}$. O elemento mais distante de S_p será inserido em S_m, onde $m \neq p$ e s_m é o elemento mais próximo de s_p, $\forall \{s_m, S_m\} \in L$
12: $V \leftarrow \{s_m, S_m\}$
13: **se** $\|S_j\| \geq \|I\| / \left\lceil \frac{N}{(C_M)^{H-l}} \right\rceil$ **então**
14: insere $\{s_j, S_j\}$ no conjunto K
15: remove $\{s_j, S_j\}$ do conjunto L
16: insere $s_1 \ldots s_k$ no nó N_i
17: atribui r como o representante do nó
18: **para** $p = 0$ até $\|L\|$ **faça**
19: insere $\{s_p, S_p\}$ no conjunto K
20: remove $\{s_p, S_p\}$ do conjunto L
21: **para** $m = 0$ até $\|K\|$ **faça**
22: *BulkLoadSlimFixedNodeByLevel(S_m, s_m, C_M)*
23: **se** $r =$ nulo **então**
24: atribui N_i como raiz da árvore

Apesar dessa abordagem levar a um bom aproveitamento de espaço em disco e a um pequeno número de nós, ela pode produzir uma ávore com alta sobreposição, pois, a construção é feita de maneira *top-down*, embora, com o mesmo número de elementos da construção *bottom-up*. O número de elementos em cada nó está sendo fixado e, por essa razão não são consideradas as propriedades de agrupamento e distribuição de dados como um todo. Nessa abordagem, os elementos em regiões mais densas terão a mesma

construção que os elementos em regiões esparsas.

5.5 Carga-Rápida com Nós de Tamanho Controlado

As duas abordagens anteriores não levam em conta a distribuição dos dados. A terceira abordagem tenta acompanhar a distribuição dos dados no espaço, agrupando mais elementos em regiões mais densas e menos em regiões esparsas, utilizando um limite controlado para definir o número de elementos necessários para manter a árvore balanceada. A estrutura é construída sempre encontrando os elementos mais próximos de cada amostra. Uma boa escolha de amostras pode levar a um melhor agrupamento dos elementos, melhorando o desempenho da estrutura.

Essa abordagem usa a proximidade definida entre os elementos para definir o número de elementos por nó, sempre considerando o limite controlado para construir uma árvore balanceada. O limite controlado é obtido pelo número mínimo e máximo de elementos para manter a árvore com altura $H = \lceil \log_{C_M} N \rceil$. Para isso são utilizados dois algoritmos: *BalancedMax* (Algoritmo 5.5.1) e *BalancedMin* (Algoritmo 5.5.2).

Algoritmo 5.5.1 *BalancedMax(S,C_m)* - Avaliador de árvore balanceada utilizando o valor de ocupação mínima do nó para verificar o tamanho máximo da estrutura gerada

Entrada: S: elementos a serem inseridos
C_m: ocupação mínima do nó
Saída: Retorna falso se a sub-árvore resultante irá tornar a árvore desbalanceada. Retorna verdadeiro caso contrário

```
1: seja ℓ o nível da sub-árvore onde os elementos de S serão inseridos
2: h ← ⌈log_{C_M} N⌉, esse valor é igual a altura da futura árvore
3: υ ← ⌈log_{C_m} ||S||⌉, esse valor é igual a altura máxima da sub-árvore resultante
4: se υ > h − ℓ então
5:     retorna falso
6: senão
7:     retorna verdadeiro
```

Os algoritmos *BalancedMin* e *BalancedMax* irão retornar falso se os números de elementos em um conjunto irão construir uma árvore balanceada e, verdadeiro caso contrário. Esses dois algoritmos funcionam como um limite para os números de elementos a serem inseridos em cada conjunto associado com uma amostra. Esses números podem ser obtidos da seguinte maneira: determina-se a altura máxima/mínima da sub-árvore que pode armazenar o número de elementos dado (linha 3). A seguir, a altura obtida é comparada (linha 4) com a altura desejada (linha 2). Com essa informação, pode-se controlar o número de elementos máximo e mínimo de um nó para construir uma estrutura balanceada.

O algoritmo de carga rápida com nós de tamanho controlado utiliza essas duas funções e pode ser dividido em três etapas: amostragem, atribuição e refinamento.

Algoritmo 5.5.2 *BalancedMin(S,C_M)* - Avaliador de árvore balanceada utilizando o valor de ocupação máxima do nó para verificar o tamanho mínimo da estrutura gerada

Entrada: S: elementos a serem inseridos
C_M: ocupação máxima do nó
Saída: Retorna falso se a sub-árvore resultante irá tornar a árvore desbalanceada. Retorna verdadeiro caso contrário

1: seja ℓ o nível da sub-árvore onde os elementos de S serão inseridos
2: $h \leftarrow \lceil \log_{C_M} N \rceil$, esse valor é igual a altura da futura árvore
3: $\upsilon \leftarrow \lceil \log_{C_M} \|S\| \rceil$, esse valor é igual a altura mínima da sub-árvore resultante
4: **se** $\upsilon < h - \ell$ **então**
5: retorna falso
6: **senão**
7: retorna verdadeiro

1. A etapa de amostragem inicia o algoritmo escolhendo amostras (Figura 5.2(a)).

2. A etapa de atribuição utiliza o algoritmo *balancedMax* e a proximidade entre cada elemento do conjunto e as amostras para criar conjuntos associados a cada amostra (Figura 5.2(b)).

3. A etapa de refinamento re-associa os elementos que não passam no critério definido pelo algoritmo *balancedMin* (Figura 5.2(c))

O método de carga-rápida proposto constrói uma estrutura balanceada de cima para baixo e não fixa o número de elementos por nó, utilizando uma função de distância e um limite para agrupar e determinar o número de elementos por conjunto em cada passo do algoritmo, que é mostrado no algoritmo 5.5.3.

O algoritmo de carga-rápida inicia com a criação de uma árvore vazia e com os seguintes argumentos: conjunto a ser carregado (I) e a ocupação máxima (C_M) e mínima (C_m) dos nós. A seguir, o algoritmo seleciona randomicamente C_M elementos (amostras) do conjunto como representativos (linha 1 e 4) - fase de amostragem. Depois todo o conjunto é particionando e é associado um sub-conjunto dos dados a cada representante (linha 5). Cada elemento é associado ao seu representante mais próximo (linha 8 e 9), respeitando a ocupação mínima do nó para manter a árvore balanceada (linha 10 a 17) - fase de atribuição. Se um subconjunto não tem a quantidade mínima de elementos para manter a árvore balanceada (linha 20), ele é descartado e seus elementos são re-inseridos em outros subconjuntos, sempre mantendo a árvore balanceada e considerando a distância entre os elementos e os representantes (linha 23) - fase de refinamento. A seguir, um nó com os representantes (linha 27) é criado e atribuído um representante para esse nó (linha 31). Então, o processo pode ser repetido, construindo a árvore de maneira recursiva (linha 34).

Os três algoritmos apresentados constróem estruturas balanceadas, a diferença principal se encontra na distribuição dos elementos nos nós da estrutura. Tanto o primeiro

Algoritmo 5.5.3 *BulkLoadSlim(I,r,C_M,C_m)* - Algoritmo de carga-rápida para a *Slim-tree*

Entrada: I: conjunto de elementos a serem carregados
r: representante da sub-árvore
C_M: ocupação máxima do nó
C_m: ocupação mínima do nó
Saída: Insere todos os elementos $i_n \in I$ com representante r

1: **se** $\|I\| \leq C_M$ **então**
2: insere todos os elementos $i_n \in I$ em um nó folha N_i e atribui r como representante do nó
3: **senão**
4: escolhe $k = C_M$ amostras $s_1 \dots s_k$ de I
5: cria k conjuntos vazios $S_1 \dots S_k$ e associa cada conjunto com sua respectiva amostra (representante do conjunto) $L = \{\{s_1, S_1\} \dots \{s_k, S_k\}\}$
6: cria um conjunto vazio $K = \{\}$
7: **para** $p = 0$ até $\|I\|$ **faça**
8: atribui i_p ao conjunto S_j, onde s_j é a amostra mais próximo de i_p que está associado ao conjunto S_j
9: $V \leftarrow \{s_j, S_j\}$
10: **enquanto** $V \in K$ **faça**
11: seja $V = \{s_p, S_p\}$. O elemento mais distante de S_p será inserido em outro conjunto S_m, onde $m \neq p$
12: $V \leftarrow \{s_m, S_m\}$
13: **se** *balancedMin(S_j,C_M)=falso* **então**
14: insere $\{s_j, S_j\}$ em um conjunto K
15: remove $\{s_j, S_j\}$ de L
16: **para** $q = 0$ até $\|L\|$ **faça**
17: **se** *balancedMax(S_q,C_m)=falso* **então**
18: distribui todos os elementos de S_q nos outros conjuntos, com a mesma estratégia utilizada na linhas 10 a 19
19: **senão**
20: insere $\{s_q, S_q\}$ em um conjunto K
21: remove $\{s_q, S_q\}$ de L
22: insere $s_1 \dots s_k$ em um nó N_i
23: atribui r como o representante do nó
24: **para** $m = 0$ to $\|K\|$ **faça**
25: *BulkLoadSlim(S_m, s_m, C_M, C_m)*
26: **se** $r =$ nulo **então**
27: define N_i como a raiz da da árvore

(Carga-Rápida com Nós de Tamanho Fixo), quanto o segundo algoritmo (Carga-Rápida com Nós de Tamanho Fixo por Nível) mantêm uma estrutura rígida fixando o número de elementos por nó, o algoritmo de Carga-Rápida com Nós de Tamanho Controlado tenta utilizar a distribuição dos dados no espaço para otimizar a estrutura, sem perder o balanceamento.

5.6 Considerações Finais

Nessa seção foram apresentadas três abordagens para a construção de um MAM utilizando a operação de carga-rápida. Além disso, foi proposta uma técnica de carga-rápida que constrói de cima para baixo um MAM hierárquico mantendo sempre a estrutura balanceada. Foi mostrado o algoritmo baseado nessa técnica para construir uma *Slim-tree*.

Esse é o primeiro algoritmo baseado em amostras para MAM dinâmicos e hierárquicos que sempre leva em consideração a distribuição dos dados e o balanceamento da estrutura, desde a sua fase inicial. Como ele sempre produz uma estrutura válida, esse algoritmo é o único encontrado na literatura que pode ser incluído em um SGBD.

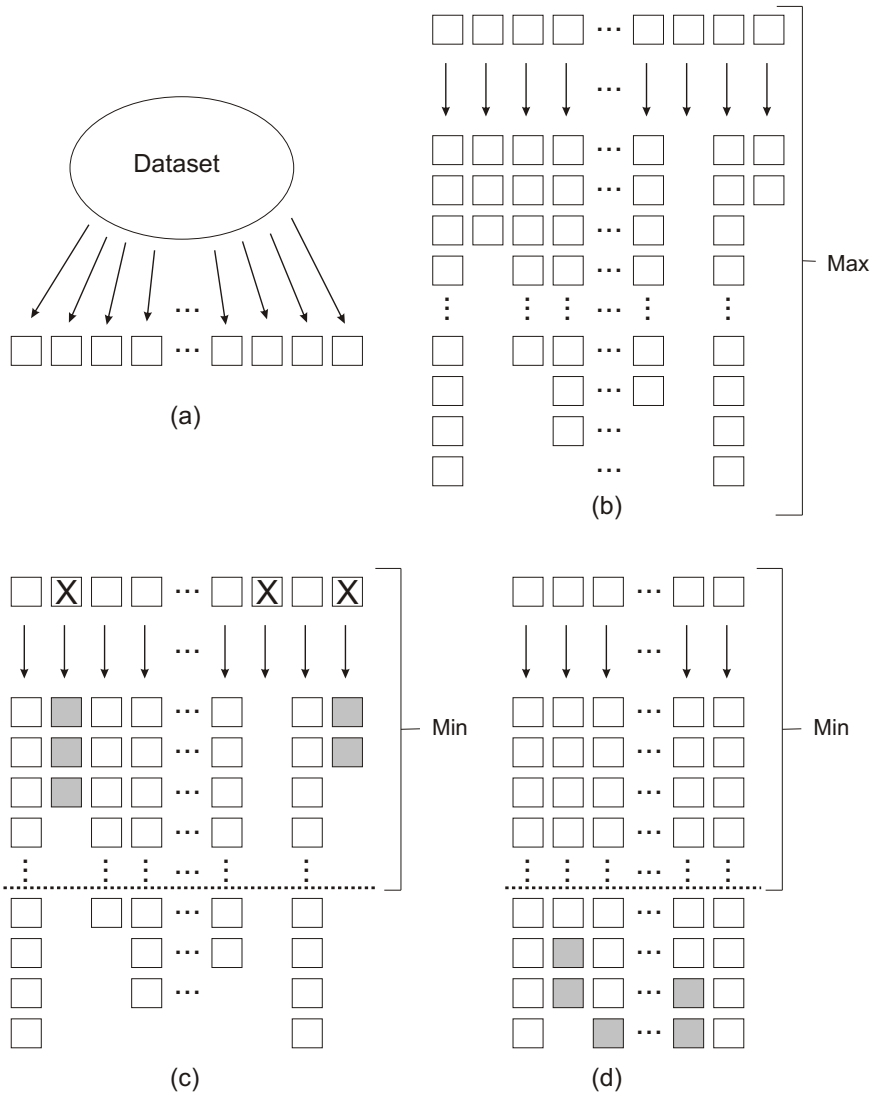

Figura 5.2: As três etapas do algoritmo de carga-rápida: (a) a etapa de amostragem escolhe C_M amostras; (b) a etapa de atribuição divide os elementos considerando a função *balancedMax* como o limite máximo (Max) para o número de elementos; (c) a etapa de refinamento atribui novamente os elementos (d) considerando um limite mínimo (Min) utilizando a função *balancedMin*.

Capítulo

6

Experimentos e Resultados

6.1 Considerações Iniciais

Nesta seção são mostrados experimentos realizados e os conjuntos e métodos utilizados para a realização desses experimentos e são discutidos os resultados obtidos. Para comparação com o método desenvolvido, utilizando a abordagem mais eficiente: nós de tamanho controlado, foi utilizado o algoritmo da *M-tree*, que é o único algoritmo de carga-rápida baseado em amostragem específico para dados métricos encontrado na literatura. Além disso, foi feita uma comparação da carga-rápida dos dados com a inserção elemento a elemento.

O processo de experimentação envolve medir o desempenho em testes práticos, comparando os algoritmos desenvolvidos com o de melhor desempenho encontrado na literatura. Os dados avaliados são os mais comuns e que expressam de maneira mais coerente o desempenho do algoritmo. Esses dados são: o tempo total de consulta, o número de cálculos de distância e o número de acessos a disco.

6.2 Conjunto de Dados Utilizados e Métodos

Para avaliar o desempenho dos algoritmos de forma justa, foram escolhidos dados reais e sintéticos com dimensionalidades variadas. A tabela 6.1 apresenta os conjuntos de dados utilizados na avaliação de desempenho, indicando seus nomes, números de elementos (#Elem), dimensionalidade de imersão (D), o tamanho da página (Pg), a métrica utilizada ($d()$) e uma breve descrição de cada conjunto. A capacidade máxima do nó C_M é obtida pelo tamanho do elemento armazenado pelo tamanho da página utilizada.

Os experimentos foram executados em um computador com processador Intel Pen-

Nome	# Elem.	D	Pg	$d()$	Descrição
Cities	5.507	2	1KB	L_2	Coordenadas geográficas das cidades do Brasil (www.ibge.gov.br)
Eigenfaces	11.900	16	4KB	L_2	Projeto Informedia da Carnegie Mellon University [Wactlar et al., 1996]
MedHisto	4.247	-	4KB	L_M	Histogramas métricos de níveis de cinza de imagens médicas. Conjunto gerado no GBDI, ICMC, USP [Traina et al., 2002].
Synt256D	10.000	256	10KB	L_2	Vetores de dados sintéticos com distribuição Gaussiana contendo 100 agrupamentos em um hipercubo com dimensão 256. O procedimento para gerar este conjunto se encontra descrito em [Ciaccia et al., 1997].

Tabela 6.1: Conjuntos de dados utilizados nos experimentos.

tium 4 HT 3.0 Ghz, com 2GB de memória RAM e 160GB de espaço em disco. O algoritmo proposto foi implementado na biblioteca Arboretum (descrita no apêndice A) utilizando a linguagem de programação C++. Em cada teste foi escolhido como algoritmo de divisão de nós o *minMax* e como algotitmo *chooseSubtree* de escolha de sub-árvore o algoritmo *minDist*. Foi utilizada a taxa de ocupação de 25% como taxa de ocupação mínima C_m, tanto na *Slim-tree* quanto na *M-tree*. Para a *Slim-tree* foi executado o algoritmo *Slim-Down* e os resultados medidos após sua execução.

6.3 Tempo de Construção

Comparando a inserção seqüencial com a operação de carga-rápida na *Slim-tree* (tabela 6.2), o tempo total para inserir 5.257 cidades no conjunto Cities utilizando o algoritmo de carga-rápida, foi 6,3 vezes mais rápido que a inserção seqüencial. Para os conjuntos Eigenfaces, MedHisto and Synth256D, a operação de carga-rápida foi, respectivamente, 4,88, 3,55 e 1,96 vezes mais rápida, em relação à inserção seqüencial.

Considerando o número de acessos a disco, o algoritmo de carga-rápida requer 44 vezes menos acessos a disco no conjunto Cities, 22,81, 5,22 e 3,38 vezes menos acesso a disco nos conjuntos Eigenfaces, MedHisto e Synth256D. Realizando a carga-rápida nesses mesmos conjuntos, é necessário, 6,1, 3,63, 1,8 e 1,79 vezes menos cálculos de distância, respectivamente, comparado com a inserção seqüêncial.

Como demonstrado, o desempenho nas operações de inserção utilizando o algoritmo de carga-rápida é melhor em relação ao da inserção seqüêncial. De fato, a operação de carga-rápida pode ser 44 vezes mais eficiente na construção em relação à inserção seqüencial. Com isso, para todos os casos analisados, o algoritmo de carga-rápida se mostrou

Conjunto de dados	Tempo total	Núm. Ac. Disco	Núm. Cálc. Dist.
Cities	0,593	36.524	206.683
Cities - Carga-rápida	0,094	830	33.852
Eigenfaces	25,703	96.646	1.594.920
Eigenfaces - Carga-rápida	5,265	4.236	439.134
MedHisto	47,659	79.140	2.845.373
MedHisto - Carga-rápida	13,419	15.153	1.511.716
Synt256D	55,978	99.446	4.463.498
Synt256D - Carga-rápida	28,563	29.446	2.487.495

Tabela 6.2: Desempenho da inserção. Comparativo entre inserção seqüêncial e a operação de carga-rápida na *Slim-tree*.

superior e mais rápido que a inserção seqüencial e com menor número de acesso a disco e cálculos de distância. A tabela 6.2 mostra as medidas obtidas.

6.4 Desempenho em Consultas

Para medir o desempenho em consultas, foi preparado um conjunto com 500 elementos para serem utilizados como centro das consultas. Esses elementos foram escolhidos aleatoriamente do conjunto de dados inicial e 250 deles foram removidos do conjunto original antes da inserção. Portanto, metade dos conjuntos de consultas foram indexados e a outra metade não. Para cada medida no gráfico, foram calculados o número médio de acessos a disco, o número médio de cálculos de distância e o tempo total de processamento em segundos para executar as 500 consultas variando o raio ξ ou o número de elementos k considerando os 500 centros de consultas.

A Figura 6.1 mostra o resultado das consultas aos k-vizinhos mais próximos realizadas sobre os conjuntos Cities, MedHisto e Synth256D. Como é possível notar, a utilização do método de carga-rápida reduz em mais de 41% o número de acessos a disco quando comparado com a inserção seqüencial na *Slim-tree* e reduz em 28% quando comparado com o algoritmo de carga-rápida para a *M-tree*.

Além disso, o algoritmo de carga-rápida para a *Slim-tree* requer 34% menos cálculos de distâncias (Figura 6.2) e é 41% mais rápido considerando o tempo de processamento (Figura 6.3) em relação à inserção sequëncial e utiliza 24% menos cálculos de distância sendo 26% mais rápido que o algoritmo desenvolvido para a *M-tree*.

Em consultas por abrangência, quando são utilizados raios pequenos em relação ao conjunto, o algoritmo de carga-rápida para a *Slim-tree* possui desempenho similar à inserção seqüencial e ao algoritmo da *M-tree*. Entretanto, ao aumentar o raio da consulta, o desempenho do algoritmo de carga-rápida da *Slim-tree* melhora em relação aos outros algoritmos. Com um raio de 32% do diâmetro do conjunto, o algoritmo da *Slim-tree* utiliza 26% menos acessos a disco (Figura 6.4), 13% menos cálculos de distância (Figura

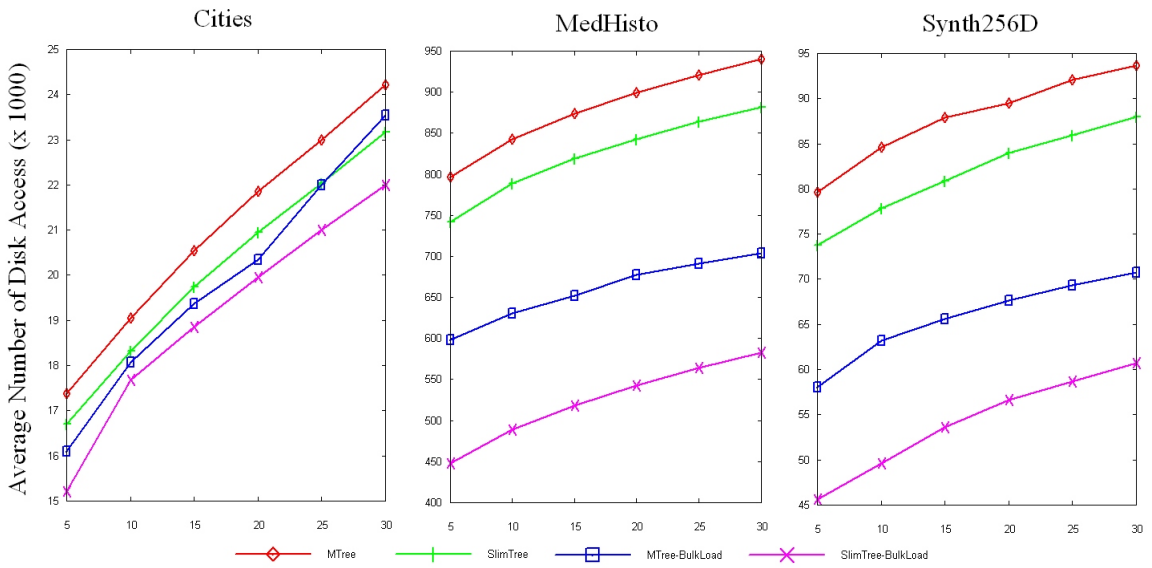

Figura 6.1: Resultado das consultas aos k-vizinhos mais próximos, comparando o número médio de acessos a disco para 500 consultas após a realização da inserção seqüêncial e carga-rápida para *M-tree* e a *Slim-Tree* utilizando os conjuntos Cities, MedHisto e Synth256D.

6.5) e é 11% mais rápido comparado com a inserção seqüêncial.

Comparando os dois algoritmos de carga-rápida, o algoritmo da *Slim-tree* utiliza 32% menos acessos a disco, 15% menos cálculos de distância e é 15% mais rápido que o algoritmo da *M-tree*.

6.5 Considerações Finais

Os resultados demonstraram que o algoritmo de carga-rápida proposto para a *Slim-tree* é seis vezes mais rápido que a construção seqüencial chegando a melhorar em até 41% o número de acessos a disco, 34% menos cálculos de distância e 41% no tempo de acesso em relação a inserção seqüêncial.

Em comparação com o algoritmo da *M-tree*, o algoritmo proposto superou o desempenho em todos os experimentos, considernado o tempo de consulta, número de acesso disco e tempo para executar as operações, tanto em consultas aos k-vizinhos mais próximos quanto por abrangência.

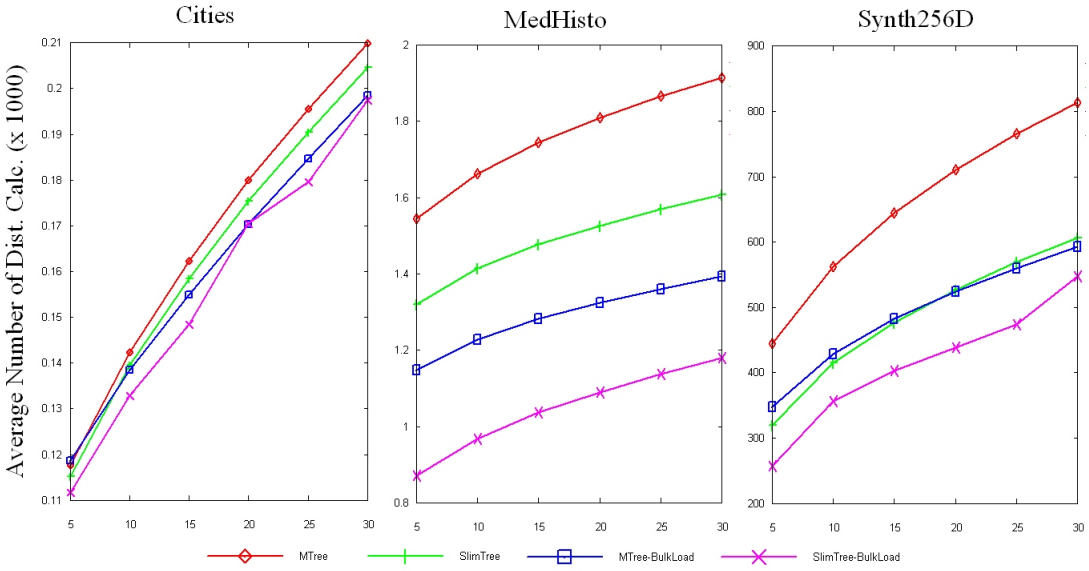

Figura 6.2: Resultado das consultas aos k-vizinhos mais próximos, comparando o número médio de cálculos de distância para 500 consultas após a realização da inserção seqüêncial e carga-rápida para *M-tree* e a *Slim-Tree* utilizando os conjuntos Cities, MedHisto e Synth256D.

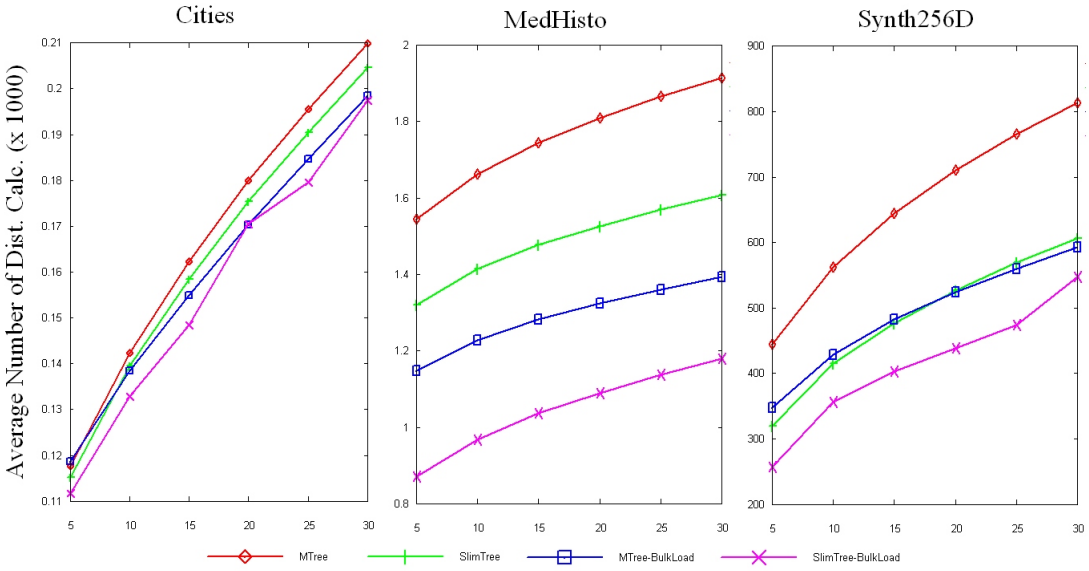

Figura 6.3: Resultado das consultas aos k-vizinhos mais próximos, comparando o tempo total para realizar 500 consultas após a a inserção seqüêncial e carga-rápida para *M-tree* e a *Slim-Tree* utilizando os conjuntos Cities, MedHisto e Synth256D.

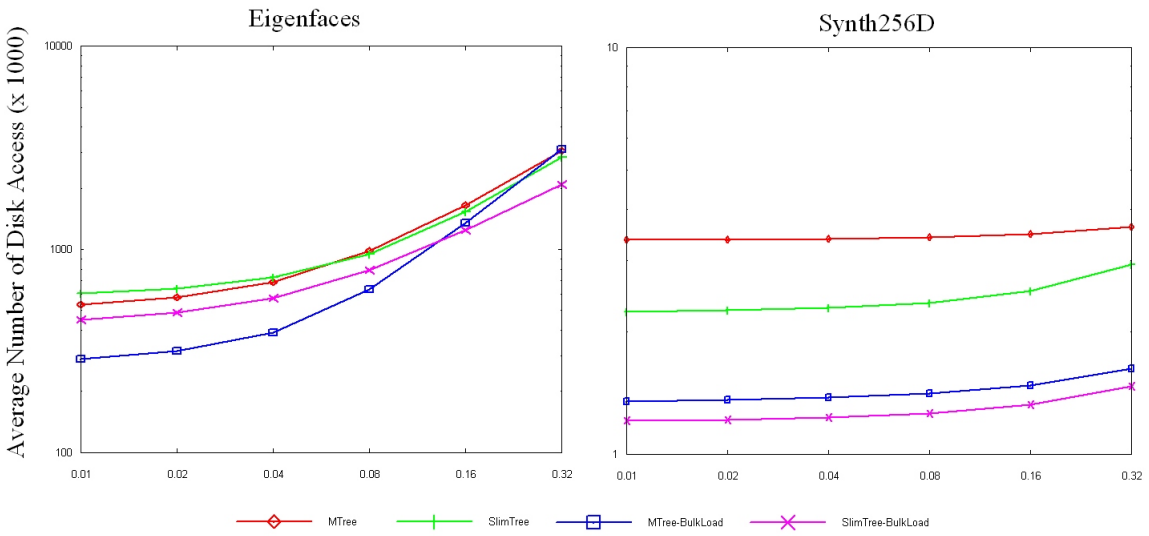

Figura 6.4: Resultado das consultas por abrangência, comparando o número médio de acessos a disco para 500 consultas após a realização da inserção seqüêncial e carga-rápida para *M-tree* e a *Slim-Tree* utilizando os conjuntos EigenFaces e Synth256D.

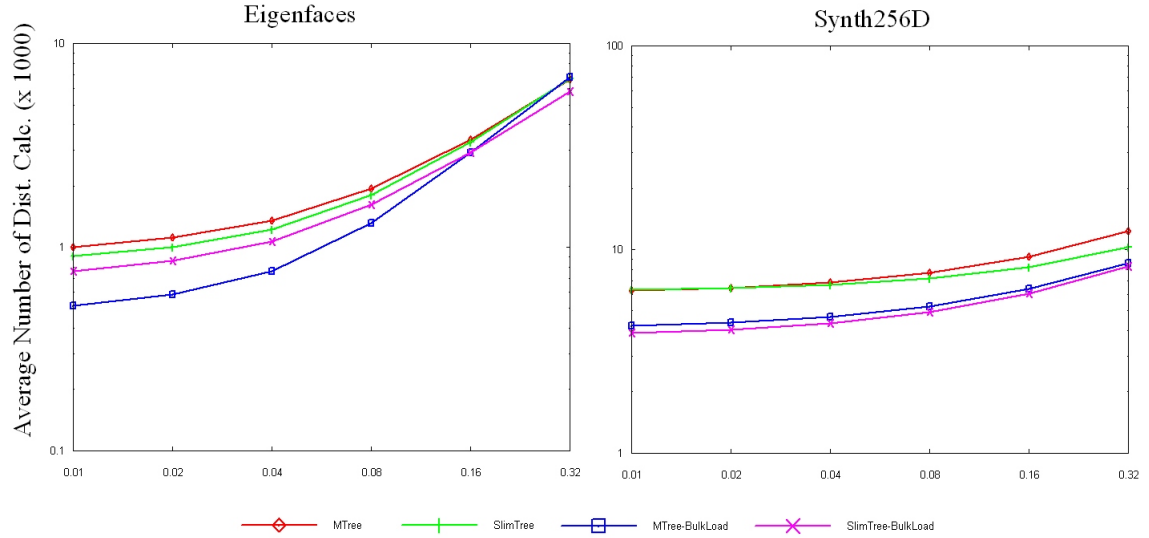

Figura 6.5: Resultado das consultas por abrangência, comparando o número médio de cálculos de distância para 500 consultas após a realização da inserção seqüêncial e carga-rápida para *M-tree* e a *Slim-Tree* utilizando os conjuntos EigenFaces e Synth256D.

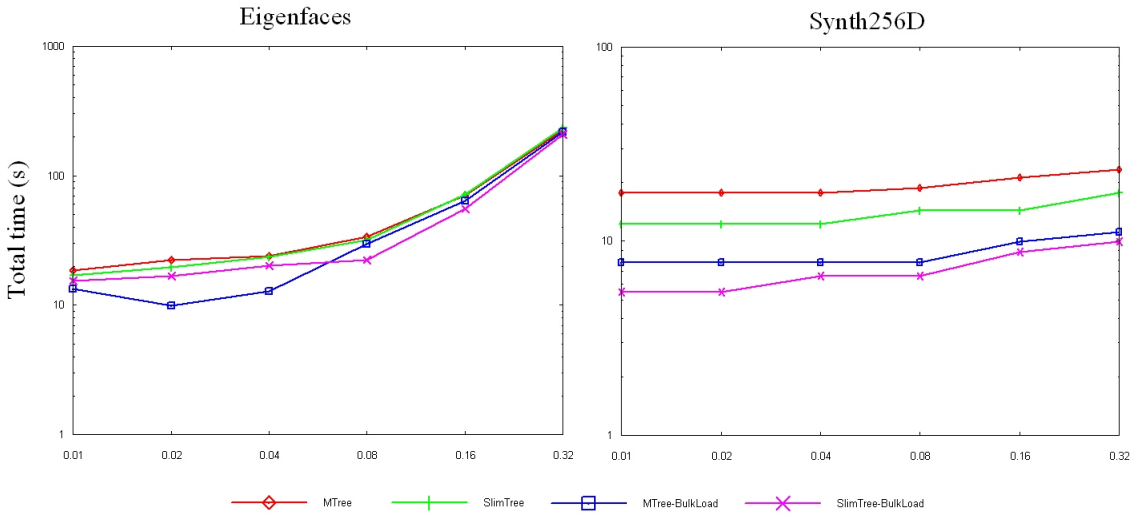

Figura 6.6: Resultado das consultas por abrangência, comparando o tempo total para realizar 500 consultas após a a inserção seqüêncial e carga-rápida para *M-tree* e a *Slim-Tree* utilizando os conjuntos EigenFaces e Synth256D.

Capítulo

7

Conclusão

Nesse trabalho, foram proposta três técnicas para carga-rápida dos dados para MAM e foram mostrado três algoritmos baseados nessas técnicas para construir uma *Slim-tree*. Esses são os primeiros algoritmos de carga-rápida baseada em amostragem para MAM hierárquicos dinâmicos e que sempre produzem um MAM válido, como demonstrado para o MAM *Slim-tree*. Portanto esses são os primeiros algoritmos de carga-rápida descritos na literatura que podem ser incluído em um SGBD [Vespa et al., 2007].

Os experimentos mostraram que o algoritmo proposto utilizando nós de tamanho controlado mantém boa clusterização dos dados e supera o desempenho dos métodos de inserção sequencial considerando tanto construção quanto desempenho nas consultas.

7.1 Principais Contribuições deste Trabalho

As principais contribuições deste trabalho são:

- Análise de três abordagens utilizando amostragem dos dados para construção de MAM hierárquicos:

    1. **Nós de Tamanho Fixo** - A estrutura é construída com o mesmo número de elementos por nó.
    2. **Nós de Tamanho Fixo por Nível** - Cada nível da estrutura é construída com o mesmo número de elementos por nó, mas níveis diferentes podem possuir quantidades diferentes de elementos por nó.
    3. **Nós de Tamanho Controlado** - O número de elementos em cada nó não é fixo e é obtido de acordo com a distribuição dos dados no conjunto a ser

indexado, mas sempre mantendo um limite mínimo e máximo para que a estrutura seja construída balanceada.

- Desenvolvimento de técnicas de carga rápida para métodos de acesso métricos hierárquicos e dinâmicos. A partir dessas técnicas, foram desenvolvidos algoritmos de carga-rápida para a *Slim-tree*. Estes são os únicos algoritmos para a *Slim-tree* baseados em amostragem que sempre geram uma estrutura válida, pois os algoritmos manteêm a estrutura balanceada desde a sua etapa inicial.

- Estudo comparativo utilizando experimentos entre os métodos de inserção: elemento a elemento e carga-rápida, mostrando que a operação de carga-rápida é mais eficiente tanto na construção, quanto no desempenho das consultas.

- Estudo comparativo utilizando experimentos entre a operação de carga-rápida da *M-tree* e da *Slim-tree*. Com esse estudo, podemos demonstrar a superioridade do algoritmo com nós de tamanho controlado, tanto no tempo de inserção, quanto ao realizar consultas.

7.2 Propostas para Trabalhos Futuros

As propostas de trabalhos futuros para esse trabalho são:

- Identificação dos casos na qual a técnica de carga-rápida com Nós de Tamanho Controlado pode ser melhorada e desenvolvimento de um algoritmo para melhorá-la.

- Desenvolvimento de um método para verificar se as amostras obtidas são os centros de seus conjunto associado e mudá-las se necessário.

- Criação e implementação de algoritmos para utilizar técnicas de clusterização para escolher as melhores amostras ao realizar a carga-rápida na estrutura.

Referências Bibliográficas

[Arge, 2003] Arge, L. (2003). The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24.

[Arge et al., 2002] Arge, L., Hinrichs, K., Vahrenhold, J., e Vitter, J. S. (2002). Efficient bulk operations on dynamic r-trees. *Algorithmica*, 33(1):104–128.

[Aslandogan & Yu, 1999] Aslandogan, Y. A. e Yu, C. T. (1999). Techniques and systems for image and video retrieval. *Knowledge and Data Engineering*, 11(1):56–63.

[Bayer & McCreight, 1972] Bayer, R. e McCreight, E. M. (1972). Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189.

[Beckmann et al., 1990] Beckmann, N., Kriegel, H.-P., Schneider, R., e Seeger, B. (1990). The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, p. 322–331, Atlantic City, USA. ACM Press.

[Berchtold et al., 1998a] Berchtold, S., Böhm, C., e Kriegel, H.-P. (1998a). Improving the query performance of high-dimensional index structures by bulk load operations. *Lecture Notes in Computer Science*, 1377:216.

[Berchtold et al., 1998b] Berchtold, S., Böhm, C., e Kriegel, H.-P. (1998b). The pyramid-technique: Towards breaking the curse of dimensionality. In Haas, L. M. e Tiwary, A., editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, p. 142–153. ACM Press.

[Bercken & Seeger, 2001] Bercken, J. V. d. e Seeger, B. (2001). An evaluation of generic bulk loading techniques. *International Conference on Very Large Databases (VLDB)*, p. 461–470.

[Bozkaya & Özsoyoglu, 1997] Bozkaya, T. e Özsoyoglu, Z. M. (1997). Distance-based indexing for high-dimensional metric spaces. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, p. 357–368, Tucson, USA. ACM Press.

[Bozkaya & Özsoyoglu, 1999] Bozkaya, T. e Özsoyoglu, Z. M. (1999). Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems (TODS)*, 24(3):361–404.

[Burkhard & Keller, 1973] Burkhard, W. A. e Keller, R. M. (1973). Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236.

[Chong et al., 2001] Chong, E. I., Das, S. Freiwald, C., Srinivasan, J., Yalamanchi, A., Jagannath, M., Tran, A.-T., e Krishnan, R. (2001). B^+-tree indexes with hybrid row identifiers in oracle8i. *IEEE International Conference on Data Engineering (ICDE)*, p. 341–348.

[Chávez et al., 2001] Chávez, E., Navarro, G., Baeza-Yates, R., e Marroquín, J. L. (2001). Proximity searching in metric spaces. *ACM Computing Surveys (CSUR)*, 33(3):273–321.

[Ciaccia & Patella, 1998] Ciaccia, P. e Patella, M. (1998). Bulk loading the m-tree. *Proceedings of the 9th Australasian Database Conference (ADC 98)*, p. 15–26.

[Ciaccia et al., 1997] Ciaccia, P., Patella, M., e Zezula, P. (1997). M-tree: An efficient access method for similarity search in metric spaces. *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB)*, p. 426–435.

[Comer, 1979] Comer, D. (1979). The ubiquitous b-tree. *ACM Computing Surveys*, 11(2):121–137.

[Faloutsos, 1996] Faloutsos, C. (1996). *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, Boston, USA.

[Faloutsos, 1997] Faloutsos, C. (1997). Indexing of multimedia data. In *Multimedia Databases in Perspective*, p. 219–245. Springer.

[Faloutsos & Roseman, 1989] Faloutsos, C. e Roseman, S. (1989). Fractals for secondary key retrieval. In *PODS '89: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, p. 247–252, New York, NY, USA. ACM Press.

[Felipe et al., 2005] Felipe, J. C., Traina, A. J. M., e Traina Jr., C. (2005). Global warp metric distance: Boosting content-based image retrieval through histograms. In *IEEE International Symposium on Multimedia (ISM 2005)*, Irvine, CA, USA.

[Felipe et al., 2006] Felipe, J. C., Traina, A. J. M., Traina Jr., C., Sousa, E. P. M. d., e Ribeiro, M. X. (2006). Effective shape-based retrieval and classification of mammograms. In *21st Annual ACM Symposium on Applied Computing (SAC 2006)*, Dijon, France.

[Gaede & Günther, 1998] Gaede, V. e Günther, O. (1998). Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231.

[GBDI-ICMC-USP, 2004] GBDI-ICMC-USP (2004). GBDI Arboretum Library. http://gbdi.icmc.usp.br/arboretum/.

[Ghanem et al., 2004] Ghanem, T. M., Shah, R., Mokbel, M. F., Aref, W. G., e Vitter, J. S. (2004). Bulk operations for space-partitioning trees. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 29, Washington, DC, USA. IEEE Computer Society.

[Guttman, 1984] Guttman, A. (1984). R-tree : A dynamic index structure for spatial searching. In Yormack, B., editor, *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, p. 47–57, Boston, USA. ACM Press.

[Hunt et al., 2001] Hunt, E., Atkinson, M. P., e Irving, R. W. (2001). A database index to large biological sequences. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, p. 139–148, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Johnson & Shasha, 1989] Johnson, T. e Shasha, D. (1989). Utilization of b-trees with inserts, deletes and modifies. In *PODS '89: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, p. 235–246, New York, NY, USA. ACM Press.

[Johnson & Shasha, 1993a] Johnson, T. e Shasha, D. (1993a). B-trees with inserts and deletes: Why free-at-empty is better than merge-at-half. *Journal of Computer and System Sciences*, 47(1):45–76.

[Johnson & Shasha, 1993b] Johnson, T. e Shasha, D. (1993b). The performance of current b-tree algorithms. *ACM Transactions on Database Systems (TODS)*, 18(1):51–101.

[Kothuri et al., 2002] Kothuri, R. K. V., Ravada, S., e Abugov, D. (2002). Quadtree and r-tree indexes in oracle spatial: a comparison using gis data. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, p. 546–557, New York, NY, USA. ACM Press.

[Lang & Singh, 2001] Lang, C. A. e Singh, A. K. (2001). Modeling high-dimensional index structures using sampling. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(2):389–400.

[Leal, 2001] Leal, E. (2001). Definition of persistence and building of a bulk loading algorithm for the slim-tree. Master's thesis, Federal University of São Carlos, São Carlos, Brazil.

[Lima, 1997] Lima, E. L. (1997). *Análise Real*. Instituto de Matemática Pura e Aplicada, 3 edition.

[Lin & Su, 2004] Lin, B. e Su, J. (2004). On bulk loading tpr-tree. *IEEE International Conference on Mobile Data Management (MDM'04)*, p. 114–124.

[Lomet, 2001] Lomet, D. B. (2001). The evolution of effective b-tree: Page organization and techniques: A personal account. *SIGMOD Record*, 30(3):64–69.

[Melton et al., 2002] Melton, J., Michels, J.-E., Josifovski, V., Kulkarni, K. G., e Schwarz, P. M. (2002). Sql/med - a status report. *ACM SIGMOD Records*, 31(3):81–89.

[Micó et al., 1994] Micó, L., Oncina, J., e Vidal, E. (1994). A new version of the nearest-neighbor approximating and eliminating search (aesa) with linear preprocessing-time and memory requirements. *Patt. Recog. Lett.*, 15:9–17.

[Munkres, 2000] Munkres, J. R. (2000). *Topology: A First Course*. Prentice-Hall, Upper Saddle River, NJ, 2nd edition.

[Patella, 1999] Patella, M. (1999). *Similarity Search in Multimedia Databases*. PhD thesis, Universita a degli Studi di Bologna, Bologna, Italy.

[Santos et al., 2001] Santos, Roberto Figueira, F., Traina, A. J. M., Traina, Caetano, J., e Faloutsos, C. (2001). Similarity search without tears: The OMNI family of all-purpose access methods. In *International Conference on Data Engineering (ICDE)*, p. 623–630, Heidelberg, Germany. IEEE Computer Society.

[Sellis et al., 1987] Sellis, T. K., Roussopoulos, N., e Faloutsos, C. (1987). The R^+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of 13th International Conference on Very Large Databases (VLDB)*, p. 507–518, Brighton, England. Morgan Kaufmann Publishers.

[Traina et al., 2002] Traina, A. J. M., Traina Jr, C., Bueno, J. M., e Marques, P. M. A. (2002). The metric histogram: A new and efficient approach for content-based image retrieval. *IFIP Working Conference on Visual Database Systems (VDB)*.

[Traina Jr. et al., 2002] Traina Jr., C., J., Traina, A. J. M., Santos, Roberto Figueira, F., e Faloutsos, C. (2002). How to improve the pruning ability of dynamic metric access methods. *Proceedings of the eleventh International Conference on Information and Knowledge Management (CIKM)*, 1(11):219–226.

[Traina Jr. et al., 2006] Traina Jr., C., Santos Filho, R. F., Traina, A. J. M., Vieira, M. R., e Faloutsos, C. (2006). The omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. *The International Journal on Very Large Data Bases (VLDBJ)*. Avaliable online: http://dx.doi.org/10.1007/s00778-005-0178-0.

[Traina Jr. et al., 2000] Traina Jr., C., Traina, A. J. M., Seeger, B., e Faloutsos, C. (2000). Slim-trees: High performance metric trees minimizing overlap between nodes. *International Conference on Extending Database Technology, v. 1777 of Lecture Notes in Computer Science*, p. 51–65.

[Vespa et al., 2007] Vespa, T. G., Traina Jr., C., e Traina, A. J. M. (2007). Bulk-loading dynamic metric access methods. *XXII Simpósio Brasileiro de Banco de Dados (SBBD'2007)*.

[Vidal, 1986] Vidal, E. (1986). An algorithm for finding nearest neighbors in (approximately) constant average time. *Patt. Recogn. Lett.*, 4:145–157.

[Vieira, 2004] Vieira, M. R. (2004). Dbm-tree: Método de acesso méetrico sensível à densidade local. Master's thesis, USP, São Carlos, Brazil.

[Vieira et al., 2006] Vieira, M. R., Traina Jr., C., Chino, F. J. T., e Traina, A. J. M. (2006). Dbm-tree: Trading height-balancing for performance in metric access methods. *Journal of the Brazilian Computer Society*, 11(3):20.

[Vitter, 1998] Vitter, J. S. (1998). External memory algorithms. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, p. 119–128. ACM Press.

[Wactlar et al., 1996] Wactlar, H. D., Kanade, T., Smith, M. A., e Stevens, S. M. (1996). Intelligent access to digital video: Informedia project. *Computer*, 29(3):46–52.

[Wilson & Martinez, 1997] Wilson, D. R. e Martinez, T. R. (1997). Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34.

[Yianilos, 1993] Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, p. 311–321, Austin, USA.

Apêndice

A

A Biblioteca de MAM Arboretum

A.1 Introdução

Como descrito no capítulo 6, o algoritmo de carga-rápida para a *Slim-tree* foi implementado na biblioteca de MAM *Arboretum* [GBDI-ICMC-USP, 2004]. Esta biblioteca foi implementada utilizando a linguagem C++, sendo disponível em ambientes Windows e Linux. Seu código fonte, bem como a sua documentação completa estão disponíveis em http://gbdi.icmc.usp.br/arboretum. A descrição a seguir foi baseada em [Vieira, 2004].

A.2 Arquitetura da biblioteca

A biblioteca de MAM *Arboretum* foi desenvolvida para ser uma base uniforme de comparação entre MAM. Ela foi desenvolvida para ser uma biblioteca portável entre vários sistemas e compiladores, fácil de usar e estender, ser orientada a objetos, e o principal, ser um *framework* completo para a implementação de MAM.

A biblioteca está dividida em 3 camadas distintas:

- **Camada do Usuário**: Define os objetos e as funções de distância métricas. As classes desta camada devem ser fornecidas pelos desenvolvedores da aplicação. A biblioteca já possui várias funções de distâncias e objetos básicos implementados, que podem ser utilizados pelos desenvolvedores. Esta camada define apenas as interfaces `stObject` e `stMetricEvaluator`, que devem ser utilizadas pelos desenvolvedores de aplicação para implementar o objeto e a métrica para construir as árvores;

- **Camada Estrutural**: Esta camada é o núcleo da biblioteca. Ela contém as classes que implementam os MAMs, além de outras classes de estatísticas e de respostas para as consultas. A *Slim-tree* pertencem a esta camada;

- **Camada de Armazenamento**: Classes que fornecem acesso aos dispositivos de armazenamento (memória e disco). Elas fornecem os serviços de gerenciamento de páginas de disco e de memória para a Camada Estrutural.

A Figura A.1 ilustra a arquitetura da biblioteca *Arboretum*. Apenas algumas classes são ilustradas em cada camada. As setas indicam o fluxo de informações entre as camadas.

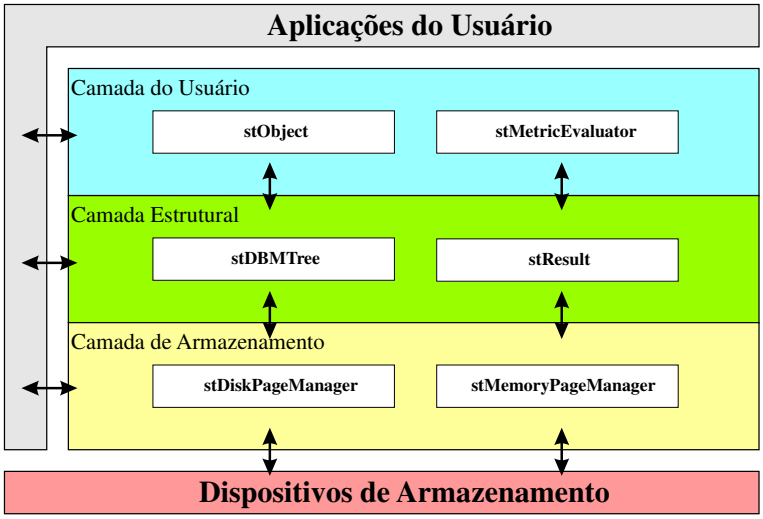

Figura A.1: Arquitetura da biblioteca *Arboretum* e as principais classes/interfaces de cada camada. As setas representam as interações entre as camadas.