Visualizando a organização e o comportamento de estruturas métricas: Aplicações em consultas por similaridade[1]

Fabio Jun Takada Chino

Orientadora: Profa. Dra. Agma Juci Machado Traina

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

USP - São Carlos
Março/2004

[1]Este trabalho contou com o apoio financeiro da Fapesp.

Agradecimentos

Aos meus pais e irmãos por tudo que fizeram para que eu pudesse concluir este trabalho.

À Profa. Dra. Agma Juci Machado Traina, minha orientadora neste projeto de mestrado e vários outros que o antecederam (ainda na graduação), não só pela oportunidade oferecida para meu ingresso em atividades de pesquisa mas também pela confiança, atenção e orientação indispensáveis para a minha formação como pesquisador.

Ao Prof. Caetano Traina Júnior, meu sempre presente "orientador extra-oficial", por todo o apoio e conhecimentos que ofereceu nestes muitos anos de trabalho no GBDI.

Ao Marcos Rodrigues Vieira, autor da *DBM-Tree*, cujas contribuições para este trabalho foram essenciais para o aprimoramento do sistema e sua validação em um ambiente real de desenvolvimento de MAM.

À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) pelo apoio financeiro.

A todos os membros da equipe de desenvolvimento da *GBDI Arboretum*, que ajudaram a transformar uma idéia simples, reimplementar a *Slim-Tree*, em um projeto mais amplo que poderá ser utilizado por muitos outros trabalhos.

A todos os membros do GBDI, em especial para o Enzo e o Humberto, por serem grades fontes de conhecimento e idéias para a elaboração deste trabalho.

Aos meus familiares, amigos, pessoal do ICMC-USP e a todos aqueles que, de uma forma ou de outra, contribuíram para a realização deste trabalho.

Resumo

O uso da computação em uma variedade cada vez maior de aplicações fez com que os Sistemas de Gerenciamento de Bases de Dados (SGBD) passassem a ser utilizados para armazenar os mais diversos tipos de dados complexos, como imagens, sons e cadeias de DNA entre outros. Consultas baseadas em relações de ordem total ou igualdade não podem ser aplicadas ou tem aplicações limitadas quando executadas nestes conjuntos de dados. Logo, efetua-se consultas por similaridade baseadas no conteúdo de dados desses tipos. Se tais conjuntos de dados podem ser representados em um espaço métrico, é possível utilizar os Métodos de Acesso Métricos (MAM), como a *Slim-Tree*, a *M-Tree* e a *DBM-Tree*, para otimizar as consultas por similaridade. Porém, os MAM são muito difíceis de compreender e analisar devido à complexidade de suas estruturas. Esta dissertação apresenta um sistema de visualização que permite a inspeção visual da organização e do comportamento de MAM, provendo aos desenvolvedores e administradores de SGBD uma forma rápida e fácil para obter informações essenciais sobre estas estruturas que podem levar a melhorias no desempenho de consultas e outras operações.

CHINO, F. J. T. *Visualizando a organização e o comportamento de estruturas métricas: Aplicações em consultas por similaridade*, São Carlos, 2004. 77 p. Dissertação de Mestrado - Instituto de Ciências Matemáticas e de Computação - ICMC, USP.

Abstract

The use of computers by an increasing variety of applications led the Database Management Systems (DBMS) to be used to store a wide range of complex data types, such as images, sounds, DNA chains, etc. Queries based on the total order relationship and/or equality can not be applied or have a limited range of applications when performed over these datasets. It is necessary to use similarity queries based on the contents of the data. If these datasets can be represented as metric spaces, it is possible to use the Metric Access Methods (MAM), such as the *Slim-Tree*, the *M-Tree* and the *DBM-Tree*, to optimize similarity queries. However, MAM are very hard to understand and analyze due to their complex structures. This work presents a visualization system that allows the visual inspection of the organization and the behavior of MAM. The usage of this system provides to MAM developers and database administrators, an easy and fast way to acquire information about key aspects of these structures, which can lead to improvements on the performance of queries and other operations.

Lista de Figuras

Lista de Tabelas

Siglas

cbPACS	*Content-Based Picture Anchiving and Communication Systems.*
CCIFM	Centro de Ciências de Imagens e Física Médica do Hospital das Clínicas de Ribeirão Preto - USP.
DBM-Tree	*Density-Based Metric Tree.*
GBDI-ICMC-USP	Grupo de Bases de Dados e Imagens - ICMC - USP
GH-Tree	*Generalized Hyperplane Decomposition Tree.*
GNAT	*Geometric Near-Neighbor Access Tree.*
HCRP-USP	Hospital das Clínicas de Ribeirão Preto - USP
ICMC-USP	Instituto de Ciências Matemáticas e de Computação - USP
MAM	*Metric Access Method* (Método de Acesso Métrico).
MDS	*Multidimensional Scale* (Escala multidimensional).
MST	*Minimal Spanning Tree* (Árvore do Caminho Mínimo).
MVP-Tree	*Multi-Vantage Point Tree.*
PACS	**Picture Anchiving and Communication Systems** [Marsh, 1997, Siegel & Kolodner, 1999, Cao & Huang, 2000],
SAM	*Spatial Access Method* (Método de Acesso Espacial).
SGBD	Sistema de Gerenciamento de Bases de Dados.
VP-Tree	*Vantage Point Tree.*

Tabelas de Símbolos

Símbolos utilizados no Capítulo 2 (Consultas por Similaridade e Espaços Métricos).

$\mathfrak{D}$	Um domínio de objetos.
$d()$	Função de distância métrica.
$dist()$	Função de distância.
k	Número de objetos de resposta.
$k - NN(q, k)$	Busca pelos k vizinhos mais próximos de q.
M	Um espaço métrico.
O	Subconjunto de $\mathfrak{D}$.
o_n	Elemento de O.
q	Objeto de busca pertencente a $\mathfrak{D}$.
$\Re$	Números reais.
r_q	Raio de uma busca por abrangência.
$range(q, r_q)$	Busca por abrangência com o objeto de busca q e raio r_q.

Símbolos utilizados no Capítulo 3 (Métodos de Acesso Métricos).

$\#Ent_i$	Número de objetos contidos na subárvore apontada pela entrada i de um nó índice.
C	Número de objetos em um nó.
$d()$	Função de distância métrica.
$fat(T)$	*FatFactor* de uma *Slim-Tree*.
H	Altura de uma *Slim-Tree*.
H_{min}	Altura mínima teórica de uma *Slim-Tree* com um determinado tamanho de nó e um certo número de objetos.
I_c	Número de acessos necessários para responder a uma consulta pontual em uma *Slim-Tree*.
M	Número de nós de uma *Slim-Tree*.
M_{min}	Número mínimo teórico de nós de uma *Slim-Tree* com um dado tamanho de nó e um determinado número de objetos.
N	Número de objetos de uma *Slim-Tree*.
Ptr_i	Ponteiro para o nó raiz da subárvore apontada pela entrada i de um nó índice.
R_i	Raio de cobertura da subárvore apontada pela entrada i de um nó índice.
$rfat(T)$	*FatFactor* absoluto de uma *Slim-Tree*.
S_i	Objeto da entrada i de um nó da árvore.
S_{rep}	Objeto representativo de uma subárvore.
T	Uma *Slim-Tree*.

Símbolos utilizados no Capítulo 4 (Visualizando Estruturas Métricas).

$\mathfrak{D}$	Um domínio de objetos.
$\mathfrak{D}_{src}$	Domínio de objetos do espaço métrico M_{src}.
$\mathfrak{D}_{trg}$	Domínio de objetos do espaço métrico M_{trg}.
$d()$	Função de distância métrica.
$\hat{d}()$	Função de distância euclidiana do espaço mapeado.
$d'()$	Função utilizada pelo *FastMap* para calcular as distâncias na iteração seguinte.
d_{ab}	Distância entre o_a e o_b.
d_{ai}	Distância entre o_a e o_i.
d_{bi}	Distância entre o_b e o_i.
d_{ij}	Distância entre o_i e o_j.
$\hat{d}_{ij}$	Distância entre as imagens dos objetos o_i e o_j.
$d_{src}()$	Função de distância do espaço métrico M_{src}.
$d_{src}()$	Função de distância do espaço métrico M_{trg}.
k	Número de dimensões do mapeamento.
m_i	Imagem do objeto o_i.
m_j	Imagem do objeto o_j.
M_{src}	Espaço métrico de origem de um algoritmo mapeador de distância.
M_{trg}	Espaço métrico de destino de um algoritmo mapeador de distância.
N	Número de objetos no conjunto de dados.
o_a	Pivô de um dos eixos de mapeamento.
o_b	Pivô de um dos eixos de mapeamento.
o_i	Objeto do conjunto a ser mapeado.
o_j	Objeto do conjunto a ser mapeado.
$stress$	Medida de erro entre as distâncias no espaço mapeado pelo *FastMap*.
x_i	Projeção do objeto o_i no eixo formado pelos objetos o_a e o_b.
x_j	Projeção do objeto o_j no eixo formado pelos objetos o_a e o_b.

Sumário

Capítulo

1

Introdução

1.1 Considerações Iniciais

O aumento da capacidade dos equipamentos e dos sistemas computacionais possibilitou o armazenamento de dados complexos e volumosos, tais como imagens, áudio, vídeo, cadeias de DNA, séries temporais entre outros. Organizar essas informações não é uma tarefa simples pois tais dados não possuem relações de ordem naturais como acontece com valores numéricos ($1 < 3 < 20$) e textos curtos ("José" < "Maria"). Porém, sobre dados complexos é possível estabelecer relações de similaridade que permitem comparar pares de objetos, estabelecendo o "quão" similares ou distintos estes são um do outro.

A partir dessas relações de similaridade, é possível definir, para esses conjuntos de dados complexos, operações de consulta baseadas na similaridade entre um dado objeto de consulta e os demais objetos do conjunto. Estas operações são denominadas *Consultas por Similaridade* e consistem em identificar todos os objetos do conjunto de dados cujo nível de similaridade com o objeto de consulta obedece a determinados critérios, como por exemplo, um limite mínimo de similaridade.

Para lidar com este tipo de consultas eficientemente, os Sistemas de Gerenciamento de Bases de Dados (SGBD), necessitam de estruturas de indexação especialmente desenvolvidas para isso. Estruturas de dados convencionais como árvores B e tabelas *Hash* são incapazes de responder a este tipo de consulta, surgem os Métodos de Acesso Métrico (MAM).

Estas estruturas abstraem os conjuntos de dados e suas relações de distância em espaços métricos, nos quais os objetos indexados constituem o domínio de dados e a

relação de similaridade é representada pela função de distância (dissimilaride) deste espaço métrico. Esta formalização das relações de distância permite que os MAM possam utilizar as relações algébricas presentes nestes espaços para responder eficientemente às consultas por similaridade.

Portanto, a capacidade dos SGBD de manipular dados complexos e permitir o uso de consultas por similaridade é bastante dependente do desenvolvimento de MAM eficientes. Infelizmente, o desenvolvimento deste tipo de estrutura é uma tarefa bastante árdua. Apesar de basear-se em um conceito aparentemente simples, o de espaços métricos, os MAM se manifestam na forma de estruturas extremamente complexas e difíceis de serem compreendidas, pois transcendem o modelo espacial que é mais facilmente assimilado pelos seres humanos.

Tendo em vista estas dificuldades, este projeto de mestrado apresenta o sistema *MAMView*, cujo principal objetivo é prover, aos desenvolvedores de estruturas métricas, meios para visualizar a organização e o comportamento de suas estruturas. Este sistema permite o acompanhamento visual de consultas por similaridade e outras operações realizadas por um MAM, aumentando a capacidade dos desenvolvedores de compreender o funcionamento destas estruturas, identificar novas propriedades e, principalmente, avaliar rapidamente os efeitos que modificações em seu comportamento provocam em sua organização levando ao aprimoramento dos MAM já estabelecidos e diminuindo os esforços necessários para o desenvolvimento de novos métodos.

1.2 Motivação e Objetivos

Diversas aplicações, como o sistema *cbPACS* (*Content-Based Picture Archiving and Communication System* [Bueno et al., 2002, Rosa et al., 2002], desenvolvido pelo Grupo de Bases de Dados e Imagens (GBDI) do ICMC-USP e pelo Centro de Ciências de Imagens e Física Médica (CCIFM) do Hospital das Clínicas de Ribeirão Preto da USP (HCRP-USP), são bastante dependentes de técnicas capazes de recuperar rapidamente dados complexos, como imagens, utilizando as relações de similaridade existentes entre estes dados.

Os Métodos de Acesso Métricos são capazes de suprir perfeitamente esta necessidade, porém, o desenvolvimento de tais estruturas de indexação é bastante árduo devido principalmente à dificuldade de compreender a organização e o comportamento destas estruturas.

Este projeto apresenta o *MAMView*, uma ferramenta que permite o acompanhamento visual dos processos de consulta por similaridade e outras operações em MAM, facilitando o estudo e o desenvolvimento destas estruturas. Ao acompanhar visualmente o comportamento e a organização de estruturas métricas, um desenvolvedor pode analisar mais facilmente tais informações, podendo explorar novas abordagens e descobrir novas

propriedades que possam promover a melhoria dos MAM já existentes ou então diminuir os esforços necessários para desenvolver novas estruturas.

Além disto, esta ferramenta pode ser utilizada ainda na atividade de ensino destas estruturas e para auxiliar desenvolvedores de aplicação que utilizam MAM a definir parâmetros para "ajustar" a estrutura de índices para permitir uma melhor organização dos dados, levando a execução de consultas mais eficientes.

O uso de um sistema de visualização de estruturas métricas, como o *MAMView*, no processo de desenvolvimento e aprimoramento de um MAM reduz consideravelmente os esforços necessários para a obtenção de uma boa estrutura, podendo até levar a descobertas que, de outro modo, não seriam obtidas. O uso deste sistema beneficia não só os desenvolvedores de MAM como também os desenvolvedores de aplicações que utilizam tais estruturas.

1.3 Organização deste Documento

Este capítulo apresentou a motivação para o desenvolvimento do trabalho, bem como os objetivos para a elaboração e organização desta dissertação. O Capítulo 2 introduz o conceito de consultas por similaridade em bases de dados multimídia, descrevendo também o uso de espaços métricos para abstrair tal tipo de consultas. Ambos os conceitos são essenciais para a compreensão dos Métodos de Acesso Métrico (MAM) descritos no Capítulo 3.

Este Capítulo descreve com detalhes a estrutura e o funcionamento da *Slim-Tree* e alguns dos aspectos mais importantes da *DBM-Tree* [Vieira & Traina Jr., 2004]. Estes dois MAM foram utilizados na validação do sistema proposto tanto em ambientes controlados (*Slim-Tree*) quanto em ambientes reais de desenvolvimento de novas estruturas (*DBM-Tree*).

O Capítulo 4 introduz o conceito de algoritmos mapeadores de distância, dando especial ênfase ao algoritmo *FastMap*. Estes algoritmos são utilizados pelo sistema *MAM-View* para criar representações visuais dos conjuntos de dados indexados por um MAM, mapeando estes objetos para pontos no espaço euclidiano tridimensional de modo a manter o máximo possível as relações de distâncias entre os objetos do conjunto original e suas imagens mapeadas no espaço euclidiano.

O modelo teórico envolvido na criação de uma ferramenta de visualização de MAM é descrito no Capítulo 5. Este modelo envolve a definição de um modelo para a representação da organização e comportamento de uma estrutura métrica genérica, um formato para apresentação visual das estruturas e a arquitetura básica de uma aplicação desta natureza. Já o Capítulo 6 descreve os detalhes da organização e da implementação do sistema *MAMView*, o protótipo criado para validar o modelo teórico de aplicação proposto

no Capitulo 5.

Os resultados obtidos com este projeto estão descritos no Capítulo 7 enquanto o Capítulo 8 encerra esta dissertação, apresentando as conclusões e propostas para trabalhos futuros.

Este documento possui ainda 4 Apêndices que trazem informações adicionais complementares ao entendimento deste projeto. O apêndice A descreve em detalhes o formato MVA, que é utilizado para a comunicação entre os módulos do *MAMView*. O apêndice B apresenta alguns detalhes sobre a implementação do sistema *MAMView* utilizado para validar este projeto.

O Apêndice C exemplifica o uso de um MAM para a indexação de dados multimídia em ambientes reais e, finalmente, o Apêndice D ilustra o uso da desigualdade triangular no descarte (poda) de objetos e ramos em uma estrutura métrica.

Capítulo

2

Consultas por Similaridade e Espaços Métricos

2.1 Introdução

Os Sistemas de Gerenciamento de Bases de Dados (SGBD) convencionais realizam buscas sobre os dados organizados através das relações de igualdade e de ordem existentes nos tipos de dados armazenados (números, datas, horários e textos curtos). Porém, quando se utilizam SGBD para armazenar dados multimídia, tais como imagens, sons e outras informações mais complexas, as buscas por igualdade (exemplo: *"Procure a pessoa cujo nome é Fulano"*) ou baseadas em ordem (exemplo: *"Procure as pessoas que recebem salário entre o dia 1 e o dia 10"*) são de pouca serventia ou simplesmente não se aplicam. Ou seja, como ordenar imagens ou vídeos sem depender de uma chave textual e numérica adicional? Porém, pode-se dizer se uma imagem ou vídeo é parecida ("similar") com outra imagem ou vídeo.

Esse tipo de busca, denominada **Consulta por Similaridade**, consiste em procurar por objetos em um conjunto que, segundo algum critério de similaridade, sejam mais "parecidos" ou mais "distintos" com/de um determinado objeto. Em outras palavras, estas são consultas que consistem em comparar todos os objetos do conjunto com um objeto escolhido, selecionando apenas os elementos que atendem a um certo critério de similaridade. Este é um processo bastante natural para o ser humano, pois é utilizado pelo cérebro para identificar o que é visto, ouvido e sentido.

As buscas por similaridade são fechadas em um domínio específico (não faz sentido

comparar imagens com sons ou com cadeias de DNA) e segundo um critério de similaridade definido para este domínio. Isto não impede que mais de um critério possa ser definido para um mesmo domínio.

Nos sistemas computacionais, o critério de similaridade se manifesta na forma de uma função *dist*() capaz de comparar dois objetos do domínio e retornar um valor numérico que quantifica o quão similares são estes objetos. Por definição, esses valores são sempre maiores ou iguais a zero, tendo valores próximos de zero para objetos muito similares (zero para objetos iguais) e valores maiores para objetos menos similares. Esta função é chamada de *função de dissimilaridade* ou de *função de distância* e deveria ser sempre definida por um especialista no domínio em questão.

Na realidade, a função de distância indica a dissimilaridade entre dois objetos, e convém colocar que a similaridade é medida pelo seu inverso. Para simplificar a compreensão do texto, muitas vezes os autores estabelecem que a função de distância indica a similaridade, já que esta é facilmente obtida pela dissimilaridade.

Em geral, a função de distância está sempre associada ao nível de semelhança entre dois objetos. Porém, a interpretação da função pode variar, alterando o resultado semântico da consulta. Por exemplo, pode-se utilizar como função de distância a distância física entre cidades, fazendo com que as cidades mais próximas sejam as mais "parecidas" e as mais distantes as mais "diferentes". Neste caso, as buscas por similaridade serão similares às buscas por proximidade em sistemas espaciais.

Existem vários tipos de consultas por similaridade, porém as mais comuns são as buscas por abrangência (limitam os resultados a um certo nível de similaridade) e as buscas pelos vizinhos mais próximos (limitam as buscas a um certo número de objetos mais similares). Esses dois tipos de consultas são detalhadas a seguir.

2.1.1 Consulta por Abrangência

As **Consultas por Abrangência** (*Range query*) consistem na busca de objetos que tenham até um determinado nível de similaridade com o objeto de referência. Isto é, dado um conjunto de objetos $O = \{o_1, o_2, o_3, ..., o_n\}$ pertencentes a um domínio $\mathfrak{D}$, uma função de distância $dist()$, um objeto de consulta q também pertencente ao domínio e uma distância máxima r_q, o resultado da busca por abrangência é dado por:

$$range(q, r_q) = A = \{a | a \in O, dist(a, q) \leq r_q\} \tag{2.1}$$

Perguntas como "Quais são as pessoas que diferem até X do Presidente do Brasil?" e "Quais são as cidades que ficam até Y km de São Paulo?" são exemplos desse tipo de consulta.

A Figura 2.1 (a) ilustra uma resposta de uma consulta por abrangência em um

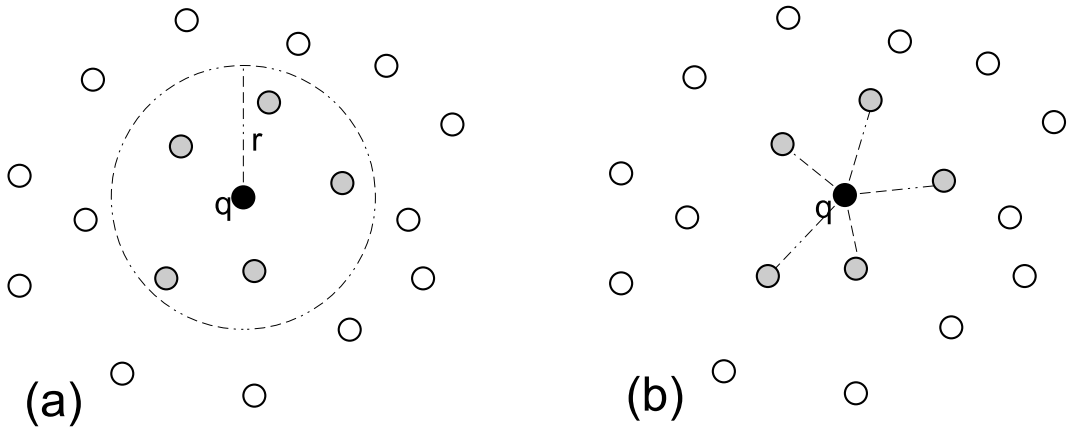

Figura 2.1: Consultas por similaridade em um domínio de pontos bidimensionais utilizando a função de distância euclidiana como função de dissimilaridade. O objeto preto q é o objeto de consulta enquanto os objetos cinza constituem as respostas. (a) Ilustra uma Consulta por abrangência de raio r e (b) ilustra uma Consulta pelos 5 vizinhos mais próximos.

domínio bidimensional com a função de distância euclidiana.

2.1.2 Consulta aos Vizinhos Mais Próximos

A **Consulta aos *k* Vizinhos mais Próximos** (*k-Nearest-Neighbor query*) consiste na busca dos k objetos mais similares ao objeto de consulta. Ou seja, dado um conjunto de objetos $O = \{o_1, o_2, o_3, ..., o_n\}$ pertencentes a um domínio $\mathfrak{D}$, uma função de distância $dist()$, um objeto de consulta q também pertencente ao domínio e um número inteiro k, o resultado da busca pelos vizinhos mais próximos é dado por:

$$k - NN(q,k) = A = \{a | a \in O, \forall o \in O - A, dist(q,a) \leq dist(q,o), |A| = k\} \qquad (2.2)$$

Perguntas como "Quais são as N pessoas mais parecidas com o Presidente do Brasil?" e "Quais são as M cidades mais próximas de São Paulo?" são exemplos desse tipo de consulta. No primeiro, k é igual a N e no segundo k é M.

A Figura 2.1 (b) ilustra uma resposta de uma consulta pelos vizinhos mais próximos em um domínio bidimensional com a função de distância euclidiana.

2.2 Espaços Métricos

O conceito de consulta (busca) por similaridade é bastante natural para o ser humano, porém, a elaboração de algoritmos capazes de executar este tipo de consulta exige a existência de leis algébricas para reger o comportamento do domínio de objetos e sua

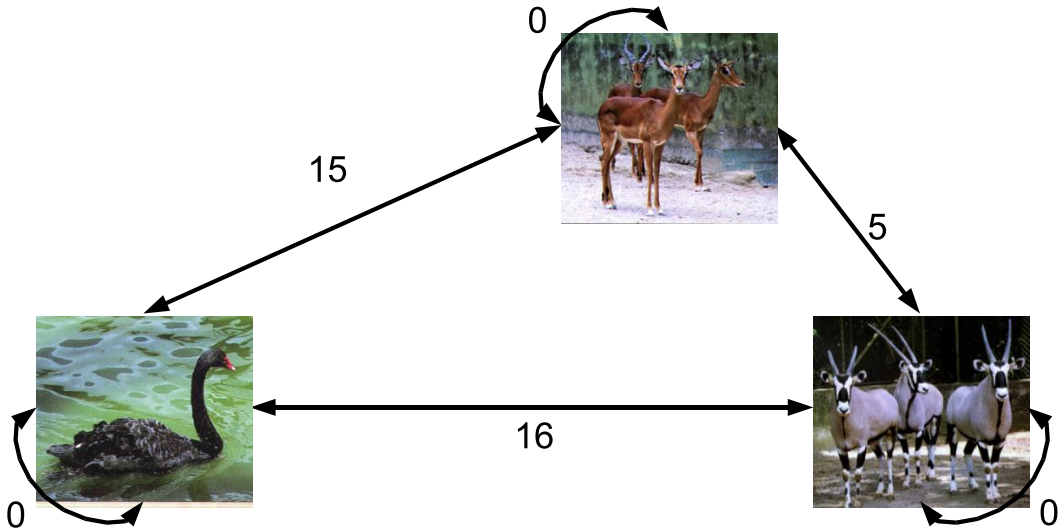

Figura 2.2: Ilustração de uma função de distância métrica imaginária para comparar fotos de animais. Cada seta representa a distância entre as figuras ligadas. As distâncias são todas simétricas (setas duplas) e não negativas, assumindo 0 apenas quando são iguais. As distâncias diretas entre os animais é sempre menor que as distâncias indiretas (desigualdade triangular).

relação com a função de distância.

Um **Espaço Métrico** é um par $M =< \mathfrak{D}, d() >$, onde $\mathfrak{D}$ é o domínio dos objetos e $d : \mathfrak{D} \times \mathfrak{D} \rightarrow \Re$ é uma **função de distância métrica** satisfazendo as seguintes propriedades:

1. **Simetria**: $d(o_1, o_2) = d(o_2, o_1)$
2. **Não negatividade**: $0 < d(o_1, o_2) < \infty$ se $o_1 \neq o_2$ e $d(o_1, o_1) = 0$
3. **Desigualdade Triangular**: $d(o_1, o_2) \leq d(o_1, o_3) + d(o_3, o_2)$

onde o_1, o_2, $o_3 \in \mathfrak{D}$.

A função de distância métrica $d()$ pode ser interpretada como uma função de dissimilaridade entre os objetos do domínio, retornando zero para objetos iguais, valores próximos a zero para objetos muito similares e valores maiores para objetos muito diferentes. Isto torna as operações de busca por similaridade naturais dentro de espaços métricos. A Figura 2.2 ilustra estas propriedades.

Outra característica dos espaços métricos é a possibilidade de englobar tanto espaços multidimensionais (como vetores numéricos) quanto espaços adimensionais[1] (como conjuntos de imagens, palavras, sons ou cadeias de DNA) desde que haja uma função de distância adequada. Infelizmente, a elaboração de uma função de distância métrica não é uma tarefa simples, mesmo para um especialista no domínio.

[1]Domínios onde os objetos não podem ser identificados por um conjunto de coordenadas em eixos ortogonais

As propriedades dos espaços métricos, principalmente a desigualdade triangular, permitem a elaboração de técnicas de indexação capazes de responder a consultas por similaridade de modo eficiente.

2.3 Conclusão

O uso crescente de sistemas de gerenciamento de bases de dados para armazenar informações multimídia criou a necessidade de responder a consultas por conteúdo nas quais apenas a similaridade entre os objetos pode ser considerada. Entre os diversos tipos de consultas por similaridade, as mais utilizadas são as consultas por abrangência e pelos vizinhos mais próximos.

Os espaços métricos introduzem uma formulação matemática propícia para abordar o problema, pois as consultas por similaridade são operações naturais nestes espaços. Baseando-se apenas em uma função de distância métrica e em suas propriedades é possível elaborar técnicas de indexação eficientes capazes de responder a consultas por similaridade, porém a definição de uma função de distância apropriada não é uma tarefa simples, existindo grupos de pesquisa trabalhando neste assunto.

Capítulo

3

Métodos de Acesso Métricos

3.1 Introdução

A difusão do uso de bases dados nas mais diversas áreas de atividades humanas gerou a necessidade de armazenar e buscar informações dos mais variados tipos, desde pequenos textos, números e datas até dados mais complexos como imagens, sons, e cadeias de DNA entre outros.

Para textos, números e datas, existem estruturas e técnicas bastante eficientes para a indexação e busca destas informações. Tais técnicas aproveitam-se das relações de ordem total existentes nestes domínios de dados. Para os tipos de dados mais complexos, como as imagens, sons e outros dados multimídia, a não existência de uma relação de ordem total nestes domínios inviabiliza o uso das técnicas de indexação convencionais, sendo necessário a utilização de novas tecnologias que permitam a realização de buscas por conteúdo ou similaridade (ver Capítulo 2).

Para solucionar esse problema, surgiram diversas propostas de métodos de indexação, dentre eles, os **Métodos de Acesso Espaciais** (do inglês *Spatial Access Methods* - SAM) [Gaede & Günther, 1998] e os **Métodos de Acesso Métrico** (do inglês *Metric Access Methods* - MAM) [Chávez et al., 2001]. Os SAM assumem que os objetos encontram-se em um espaço multidimensional qualquer, o que não é necessariamente aplicável para todos os domínios multimídia (conjuntos de imagens por exemplo, nos quais as características extraídas das imagens são utilizadas na indexação das mesmas). A abordagem dos MAM assume que os objetos estão em um espaço métrico (ver seção 2.2) que relaciona os objetos apenas por suas relações de similaridade, um conceito mais natural

para o ser humano em algumas aplicações, como já discutido no capítulo anterior.

Um modo bastante utilizado e eficiente de implementar um MAM é uma abordagem recursiva que geralmente se manifesta na forma de árvores (conhecidas como árvores métricas [Burkhard & Keller, 1973]). Estas estruturas são geralmente organizadas em páginas em disco de tamanho fixo (de forma similar às árvores B) e visam minimizar o número de comparações entre os objetos particionando o espaço em regiões definidas por um ou mais objetos escolhidos como referências.

Seguindo esta linha de raciocínio, foram criados métodos como a árvore métrica de Uhlmann [Uhlmann, 1991], a *VP-Tree* [Uhlmann, 1991, Yianilos, 1993] e a *MVP-Tree* [Bozkaya & Özsoyoglu, 1997, Bozkaya & Özsoyoglu, 1999], a *GH-Tree* [Uhlmann, 1991] e a *GNAT* [Brin, 1995], entre outras, que particionam o espaço escolhendo um ou mais objetos do conjunto para serem utilizados como referência para indexar os demais objetos do conjunto. Estas estruturas exigem que todos os objetos a serem indexados devem estar disponíveis durante o processo de criação das árvores, tornando-as estáticas (não permitem inserções posteriores), limitando seu uso prático em SGBD.

Para suprir essas deficiências, surgiu a *M-Tree* [Ciaccia et al., 1997] que inovou os MAM ao permitir a inserção de objetos durante toda a vida da estrutura, sendo construída de maneira similar à *R-Tree* [Guttman, 1984]. A *Slim-Tree* [Traina Jr. et al., 2000a] é uma evolução da *M-Tree*, propondo um novo método de divisão de nós, um modelo de avaliação do grau de sobreposição dos nós e um mecanismo para a otimização da árvore. Estas características tornam o desempenho geral da *Slim-Tree* superior ao da *M-Tree*, especialmente em seu tempo de criação e no tempo resposta às consultas por similaridade.

Tanto a *M-Tree* quanto a *Slim-Tree* tentam minimizar o número de acessos a disco e cálculos de distância necessários para realizar consultas mantendo o balanceamento das estruturas, minimizando assim a altura destas. Porém esta política tende a criar árvores com um maior grau de sobreposição entre os nós, aumentando a probabilidade de que os algoritmos de busca tenham que verificar múltiplas subárvores (buscas em largura) para obter os resultados, degradando seu desempenho.

A influência da sobreposição no desempenho das estruturas pode ser consideravelmente maior que a altura destas. Partindo deste princípio, uma nova estrutura métrica, a *DBM-Tree* [Vieira & Traina Jr., 2004] utiliza políticas de construção que priorizam um menor nível de sobreposição, permitindo um certo nível de flexibilização do balanceamento da estrutura em regiões mais densas dos conjuntos de dados.

Este capítulo descreve o funcionamento da *Slim-Tree* e da *DBM-Tree* para ilustrar duas árvores métricas com comportamentos distintos. Estas duas estruturas foram utilizadas para testar e validar os resultados deste trabalho. O conhecimento relativo ao funcionamento destas estruturas é de extrema importância para o entendimento de alguns dos problemas expostos nesta dissertação.

3.2 A Slim-Tree

A *Slim-Tree* [Traina Jr. et al., 2000a] é uma árvore multivias balanceada e dinâmica desenvolvida para indexar objetos contidos em um espaço métrico, possibilitando a execução de buscas por similaridade minimizando tanto o número de cálculos de distância quanto o número de acessos a disco.

Esse MAM divide o espaço em regiões e sub-regiões não disjuntas (representadas por cada subárvore) definidas por um objeto denominado **objeto representativo** (S_{rep}), utilizado como referência para os demais objetos da subárvore, e um raio de cobertura que deve ser grande o suficiente para abranger toda a subárvore. Desta maneira, os algoritmos de busca podem "podar" ramos inteiros da árvore, comparando a distância do objeto de consulta com o representativo, o raio da consulta e o raio da subárvore, valendo-se da propriedade da desigualdade triangular das funções de distância dos espaço métricos (ver Apêndice D).

A estrutura possui ainda recursos para a avaliação do grau de sobreposição entre seus nós (*Fat-Factor*) e um algoritmo de otimização da árvore (*Slim-Down*).

As próximas seções descrevem mais detalhes da *Slim-Tree*, como a sua organização (Seção 3.2.1), a construção da árvore e os algoritmos de divisão de nós (Seção 3.2.2), o funcionamento dos algoritmos de busca (Seção 3.2.3), o *Fat-Factor* (Seção 3.2.4) e finalmente o algoritmo de otimização *Slim-Down* (Seção 3.2.5).

3.2.1 Organização da Slim-Tree

A *Slim-Tree* é organizada em páginas em disco de tamanho fixo, cada uma contendo um nó da árvore. O balanceamento da estrutura é obtido através do crescimento de baixo para cima, das folhas para a raiz (assim como as árvores B), e os objetos são armazenados apenas no último nível da árvore - as folhas, assim como acontece nas árvores B+.

Os nós da *Slim-Tree* podem ser divididos em dois tipos, os *Nós Índice* (*IndexNode*), que habitam os níveis superiores da árvore, e os *Nós Folha* (*LeafNode*), que formam o nível mais baixo da árvore.

Os **Nós Índice** são compostos por C entradas, cada uma referente a uma de suas subárvores. Uma entrada é composta pelo objeto representativo S, o raio de cobertura R e o número de entradas da subárvore $\#Ent$. Ela ainda possui o ponteiro para a cabeça da subárvore Ptr e a distância entre o objeto S e o objeto representativo do nó S_{rep} que está armazenado no nível superior da árvore. A estrutura de um nó índice é definida por:

$$IndexNode = [\text{vetor } [1..C] \text{ de } <S,\ R,\ \#Ent,\ Ptr,\ d(S_{rep}, S)>]$$

Os **Nós Folha** são os responsáveis por armazenar todos os objetos, não possuindo subárvores associadas. Assim como os nós índice, os nós folha são compostos por C entra-

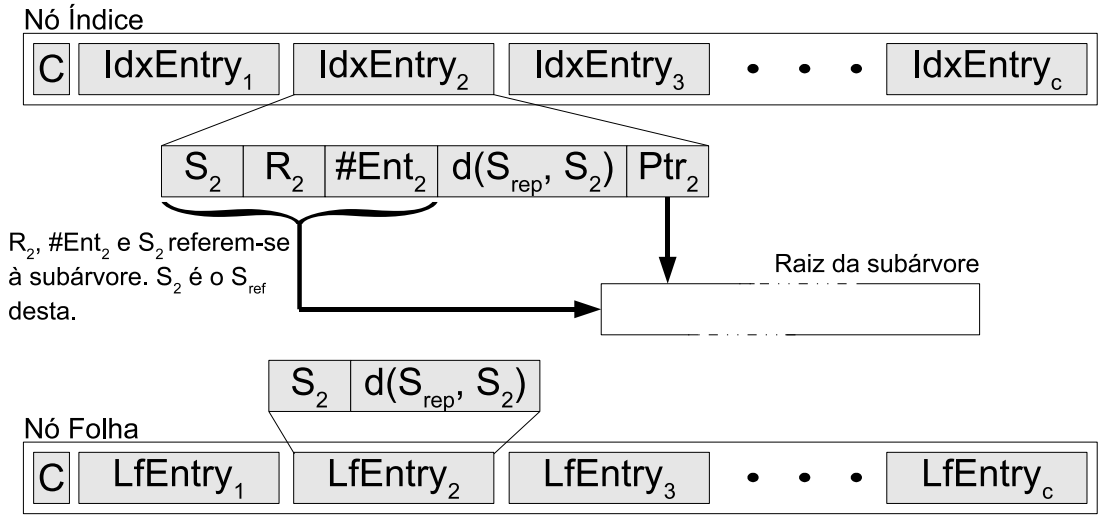

Figura 3.1: Estruturas dos nós da *Slim-Tree*. As informações referentes ao objeto representativo, o raio e o número de objetos de uma subárvore estão sempre armazenadas no nível imediatamente superior.

das, cada uma composta apenas por um objeto S e a distância do objeto ao representativo do nó S_{rep} (a estrutura original da *Slim-Tree* incluía um identificador de objetos, mas este foi removido na versão mais recente e incorporado à definição do objeto). Sua estrutura é definida por:

$$LeafNode = [\text{vetor } [1..C] \text{ de } <S,\ d(S_{rep}, S)>]$$

Um objeto só está presente no conjunto indexado se este for encontrado em um nó folha qualquer. A Figura 3.1 apresenta uma representação gráfica da estrutura dos nós índice e folha de uma *Slim-Tree*.

O objeto representativo de cada nó S_{rep} está armazenado no nível superior da árvore, porém o algoritmo de construção da árvore garante que uma cópia deste objeto estará presente neste nó, seja como o representativo de uma subárvore (nó índice) ou como objeto indexado (nó folha).

Esta organização leva a um hipotético problema com o representativo da raiz. Felizmente o fato da raiz não poder possuir um objeto representativo não é um problema real, pois a única utilidade do objeto representativo de uma subárvore é definir o raio desta e servir de referência para a poda durante uma busca. Como não faz muito sentido podar toda a árvore em uma busca (há uma probabilidade muito pequena de que isto ocorra), não é necessário ter um representativo para a raiz.

A Figura 3.2 ilustra um exemplo de uma *Slim-Tree* construída para um conjunto de 12 objetos em um espaço métrico definido por coordenadas bidimensionais e a função de distância euclidiana.

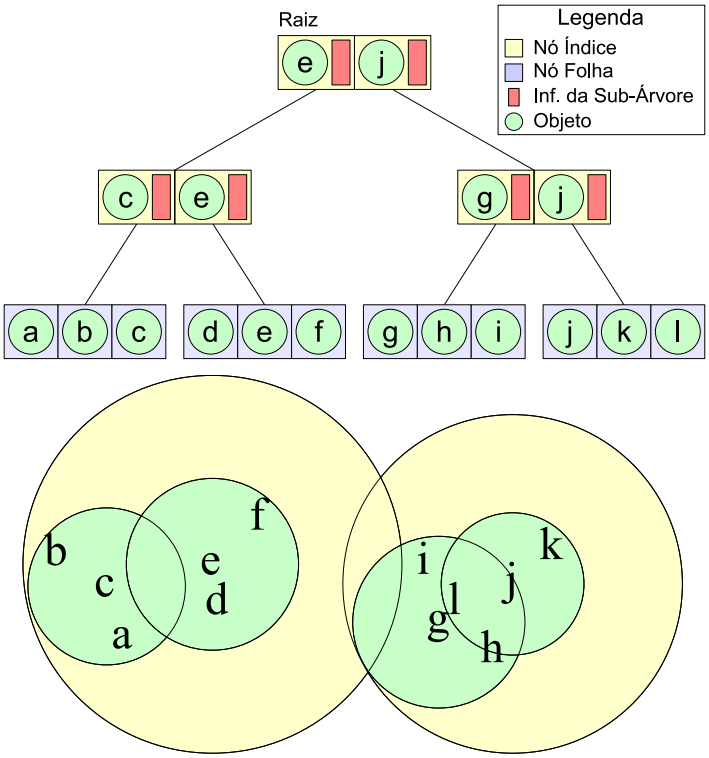

Figura 3.2: Exemplo de uma *Slim-Tree* indexando objetos bidimensionais usando a função euclidiana como distância. (a) mostra a organização da árvore de acordo com a estrutura dos nós. (b) mostra a mesma árvore considerando apenas os raios de cobertura das subárvores. Cada cor indica um nível distinto da árvore, sendo possível identificar a existência de sobreposições entre as subárvores (intersecção entre os círculos).

Para o funcionamento da estrutura, o número de objetos por nó C deve ser pelo menos 3 (o exemplo mostrado na Figura 3.2 é meramente ilustrativo). Em sua versão mais recente, a *Slim-Tree* permite que o número de objetos presentes em cada nó possa variar de acordo com o tamanho dos objetos e da página, flexibilizando a definição dos nós sem alterações no comportamento da árvore.

3.2.2 Construindo uma *Slim-Tree*

A *Slim-Tree* é construída de baixo para cima como uma árvore B. Ao contrário dos métodos estáticos, não é necessário que todos os objetos estejam presentes durante a construção da árvore. Os objetos podem ser inseridos diretamente, a qualquer momento.

Começando pela raiz, o algoritmo tenta localizar uma subárvore cujo raio cubra o objeto (a subárvore se qualifica) ou que possa cobri-lo com o menor custo possível (aumento do raio) caso não haja nenhuma candidata. Se há mais de uma subárvore que se qualifica, um algoritmo de escolha (*ChooseSubTree*) é usado para determinar em qual ramo o objeto será inserido. Se a raíz da subárvore escolhida é um nó índice, o processo se repete recursivamente, caso contrário, tenta-se inserir o objeto neste nó folha.

Se há espaço nesse nó, o objeto é inserido e o algoritmo retorna a recursão atualizando a contagem dos objetos e os raios do ramo afetado pela inserção. Caso não haja espaço, a subárvore definida por esse nó deve ser dividida. Para tanto, um novo nó folha é criado e os objetos são distribuídos entre os dois nós por um algoritmo de divisão. Este algoritmo elege dois representativos, um para cada subárvore, que são promovidos para o nível superior juntamente com as informações das subárvores.

No nó superior (sempre um nó índice), a entrada da subárvore dividida é atualizada (trocando o representativo se necessário e atualizando o raio e a contagem de objetos desta) e a entrada referente à nova subárvore é adicionada (composta pelo representativo, o raio, a contagem de objetos e a ligação para a raiz desta). Se houver espaço, o algoritmo retorna na recursão atualizando as contagens de objetos e os raios do ramo afetado. Se não houver espaço, esta subárvore também é dividida, de forma similar à divisão de um nó folha, porém, criando e distribuindo as entradas entre os nós índice. Os novos representativos e as informações das subárvores são promovidas para o nível superior, repetindo o processo até retornar à raiz. Quando a raiz é dividida, um novo nó índice é criado para ser a nova raiz e acomodar as entradas das subárvores, fazendo com que a árvore ganhe um novo nível.

Entre os critérios utilizados pelo algoritmo *ChooseSubTree* estão:

- **Aleatório**: Escolhe aleatoriamente qualquer um dos ramos que se qualificam. Este é o critério mais simples e de menor custo;
- **Distância Mínima**: Escolhe o ramo cujo objeto representativo esteja mais próximo do objeto inserido;
- **Ocupação Mínima**: Escolhe o ramo que tenha o menor número de objetos. É o critério padrão de escolha da *Slim-Tree*.

A escolha do critério do algoritmo *ChooseSubTree* influencia bastante as características da árvore resultante. O critério **Distância Mínima** tende a criar árvores com uma taxa de ocupação menor e com um grau menor de sobreposição, conseqüentemente as árvores resultantes são mais altas. O critério de **Ocupação Mínima** tende a gerar árvores com taxas de ocupação maior, conseqüentemente mais baixas, porém com um grau de sobreposição maior.

Surpreendentemente, o critério **Aleatório** consegue gerar árvores quase tão boas quanto os outros dois, porém sua eficiência e resultados não são determinísticos. Este fenômeno pode indicar que o verdadeiro responsável pelas características da árvore seja mesmo o algoritmo de divisão (*splitting*) ou quebra de nó.

Entre os algoritmos de quebra de nó disponíveis estão:

- **Aleatório**: Escolhe aleatoriamente 2 objetos do nó e os usa como representativos na divisão dos objetos;
- **minMax**: É o algoritmo padrão de quebra de nós da *M-Tree*. É considerado um algoritmo *optimal*, porém seu custo computacional é muito alto. Este algoritmo toma todos os pares de objetos possíveis e testa para cada par todas as combinações possíveis de objetos para verificar o que acarretará em dois nós com o maior número de objetos e o menor raio em cada nó;
- **MST**: É baseado na árvore do caminho mínimo (*Minimal Spanning Tree* - MST) [Kruskal, 1956], para seleção de dois agrupamentos de objetos, sendo o algoritmo padrão da *Slim-Tree*. Consegue um bom desempenho para consultas, e é similar ao **MinMax** demandando apenas uma pequena fração do seu tempo de processamento.

Tanto o algoritmo **Aleatório** quanto o **minMax** baseiam-se na escolha de dois objetos do nó a ser dividido para serem os representativos dos dois nós resultantes. Os demais objetos são distribuídos entre os dois nós de forma a minimizar os raios de ambos.

De todos os algoritmos possíveis, o **Aleatório** é sem dúvida o mais rápido porém seus resultados são os menos satisfatórios. O **minMax** oferece os melhores resultados em termos de grau de sobreposição, porém, sua complexidade $O(C^3)$, torna o tempo de construção da árvore consideravelmente longo (C é a capacidade do nó, ou número de objetos que podem ser armazenados em um nó da árvore). Isto acontece pois o **minMax** é um algoritmo de comparação exaustiva, ou seja, testa todos os pares de objetos para descobrir quais objetos são os melhores representativos para os novos nós. Cada teste consiste em fazer a distribuição dos objetos em torno dos representativos a serem testados e avaliar os nós resultantes.

O algoritmo baseado no **MST** consiste em construir uma MST com os objetos do nó, eliminando o maior arco, definindo assim dois agrupamentos que formarão os novos nós. Para cada um destes agrupamentos, o algoritmo elege um objeto para ser o representativo deste de forma a minimizar o raio de cobertura. A complexidade do algoritmo é $O(C^2.logC)$, apresentando um desempenho para as buscas equivalente ao do **minMax**, perdendo um pouco quando C é pequeno.

3.2.3 Executando Consultas por Similaridade

A *Slim-Tree* implementa os dois tipos de consultas por similaridade mais comuns citadas no Capítulo 2 e algumas variantes que envolvem a combinação dessas, como uma consulta por abrangência limitada a um certo número de objetos mais próximos.

Os algoritmos de consulta aproveitam a organização da árvore e da propriedade da desigualdade triangular da função de distância para minimizar o número de acessos a disco e de cálculos de distância.

Dado um objeto de consulta e os parâmetros da busca (k para a busca pelos vizinhos mais próximos ou o raio para a busca em abrangência), os algoritmos começam pela raiz da árvore, comparando o objeto de busca com todos os objetos representativos do nível abaixo (que estão associados a cada uma das ligações). Apenas as subárvores que se classificarem, ou seja, cobrirem possíveis respostas, são percorridas recursivamente. Isto ocorre até que o algoritmo atinja um nó folha. Neste caso, todos os objetos deste nó folha são comparados com o objeto de busca a fim de determinar quais deles se qualificam para a resposta, podendo ser adicionados a esta. O algoritmo termina quando não existirem mais ramos que possam conter objetos que pertençam à resposta.

Para as consultas por abrangência, todos os objetos que estiverem a uma distância menor que o raio de busca estão qualificados para a resposta, sendo um algoritmo bastante simples de ser implementado e compreendido.

Nas consultas pelos vizinhos mais próximos, o critério de qualificação de objetos não é tão trivial. Em uma primeira fase, todos os k objetos encontrados, supostamente mais próximos do objeto de busca são inseridos na resposta sem muito critério e sem a possibilidade de poda. A partir deste ponto, é possível começar a percorrer a árvore com um raio de poda definido que é a maior distância entre o objeto de busca e os objetos atualmente na lista de resposta. Um nó só se qualifica quando possuir objetos que estejam a uma distância menor que o atual raio de consulta. Os demais objetos podem ser descartados.

Sempre que um objeto mais próximo que este raio é encontrado, este é inserido na resposta, substituindo o objeto mais distante. Isto diminui o raio dos possíveis resultados, melhorando a capacidade de futuras podas nos demais ramos da árvore.

As buscas pelos vizinhos mais próximos pode levar a situações onde o número de candidatos solicitados para a resposta excede k (ocorre empate). Isto pode acontecer quando mais de um objeto tem a sua distância para o objeto de busca igual ao raio máximo definido pelos objetos de resposta. Para estas situações, a *Slim-Tree* define dois critérios de desempate, um que inclui todos os candidatos na resposta, ou seja, com uma lista de empate, e outra que inclui apenas o primeiro objeto da lista de empate encontrado, ignorando os demais.

Como geralmente a função $d()$ toma valores no conjunto dos reais, a possibilidade de empate para grande parte dos conjuntos de dados é bastante pequena, tornando a questão pouco relevante. Para funções como a distância de [Levenshtein, 1966], utilizada para medir a distância entre palavras através da contagem mínima de inserções, remoções e substituições de caracteres necessárias para transformar uma palavra em outra, cujo domínio de respostas está então limitado a números inteiros entre 0 e o número máximo de caracteres das palavras do conjunto de dados, a probabilidade de ocorrência de empates cresce, aumentando também a importância da lista de empates.

3.2.4 O *Fat-Factor*

O *Fat-Factor* é um conjunto de medidas do grau de sobreposição entre os nós de uma árvore, permitindo tanto avaliações locais quanto comparações entre árvores distintas.

O grau de sobreposição entre duas subárvores é dado pelo número de objetos que se qualificam para pertencer a ambos os nós, dividido pelo total de objetos de ambas as subárvores. Em outras palavras, se não houver pelo menos um objeto nesta condição, o grau de sobreposição entre os ramos é 0, mesmo que os raios de cobertura se sobreponham.

O *Fat-Factor* absoluto é uma medida do grau geral de sobreposição da árvore, definido como:

$$fat(T) = \frac{I_c - H.N}{N} \frac{1}{M - H} \tag{3.1}$$

onde, T é uma *Slim-Tree*, H denota a altura da árvore, M o número de nós, N é o número total de objetos e I_c é o número de acessos necessários para responder a uma consulta pontual (uma consulta por igualdade absoluta para um determinado objeto). O valor do *Fat-Factor* absoluto estará sempre entre 0 e 1, sendo 0 o melhor valor possível (árvores sem nenhuma sobreposição) e 1 o pior (todos os nós se sobrepõem).

Para comparar duas árvores construídas com o mesmo conjunto de dados, foi definido o *Fat-Factor* relativo:

$$rfat(T) = \frac{I_c - H_{min}.N}{N} \frac{1}{M_{min} - H_{min}} \tag{3.2}$$

onde, T é uma *Slim-Tree*, H_{min} denota a altura teórica mínima possível para a árvore, M_{min} o número teórico mínimo de nós, N é o número total de objetos e I_c é o número de acessos necessários para responder a uma busca pontual. O valor do *Fat-Factor* relativo será um valor positivo maior que 0. Quanto maior o valor, pior estará a situação da sobreposição entre os nós na árvore.

3.2.5 O *Slim-Down*

Os algoritmos de escolha de subárvore e de quebra de nós (divisão) tentam minimizar a sobreposição entre os nós da árvore, porém não são capazes de garantir que o grau de sobreposição entre os nós seja o menor possível. A natureza dos conjuntos e a ordem com a qual os objetos são inseridos acabam afetando o desempenho final da árvore. Para minimizar estes efeitos, a *Slim-Tree* possui um algoritmo de otimização denominado *Slim-Down*.

O *Slim-Down* consiste em avaliar o grau de sobreposição entre dois nós folhas. Quando existe sobreposição, os objetos de ambos os nós são comparados com ambos os representativos, a fim de determinar quais objetos podem migrar de um nó para outro de

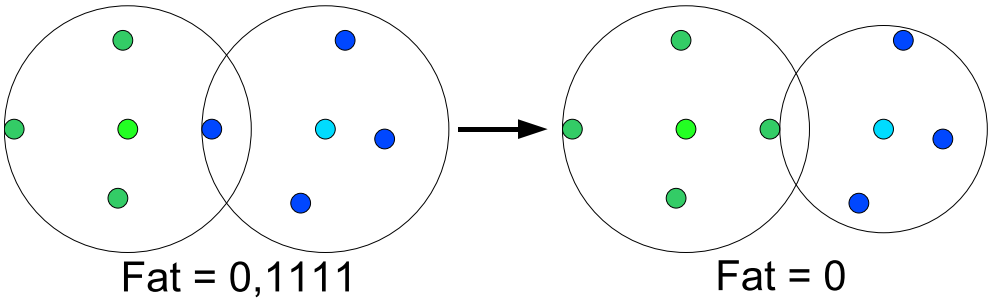

Figura 3.3: A aplicação do *Slim-Down* em dois nós folhas. À esquerda estão os dois nós antes da otimização (Fat-Factor = 0,1111) e à direta os dois nós depois do processo (Fat-Factor = 0). A troca do objeto que estava na sobreposição foi suficiente para diminuir a sobreposição, otimizando assim a estrutura.

maneira a reduzir a sobreposição entre ambos os nós. Este processo é realizado até que não seja mais possível realizar trocas de objetos entre os nós envolvidos. A Figura 3.3 representa visualmente este processo.

A versão atual do algoritmo restringe as comparações apenas aos nós folhas que pertençam a uma mesma subárvore, ou seja, apenas os nós folha que estejam ligados a um mesmo nó índice. Mesmo com esta limitação, o uso do *Slim-Down* aumenta consideravelmente o desempenho das consultas em relação a uma *Slim-Tree* não otimizada. Por ser um processo relativamente caro, é recomendado que este seja executado apenas quando o grau de sobreposição ultrapasse um certo limite para toda a árvore.

3.3 A DBM-Tree

Assim como a *Slim-Tree*, a *DBM-Tree* [Vieira & Traina Jr., 2004] manifesta-se como uma árvore multivias que divide o espaço em regiões e sub-regiões não disjuntas (representadas por cada subárvore) definidas também por um objeto denominado **objeto representativo** (S_{rep}), utilizado como referência para os demais objetos da subárvore, e um raio de cobertura que deve ser grande o suficiente para abranger toda a subárvore.

O grande diferencial desta estrutura em relação às demais é o uso de um balanceamento flexível como meio para diminuir a sobreposição entre subárvores. A *DBM-Tree* permite que subárvores sejam mais profundas em regiões do espaço onde a densidade de objetos é maior. Isto aumenta consideravelmente as possibilidades de construção das estruturas, permitindo que estas assumam configurações que resultam em menores índices de sobreposição que estruturas balanceadas poderiam conseguir.

Aparentemente, o número de acessos a disco necessários para uma busca deveria ser maior na *DBM-Tree* que em outras estruturas balanceadas devido à maior profundidade das subárvores. Porém, a diminuição do grau de sobreposição entre as subárvores tende a minimizar o uso das buscas em largura na estrutura, consideravelmente mais numerosas

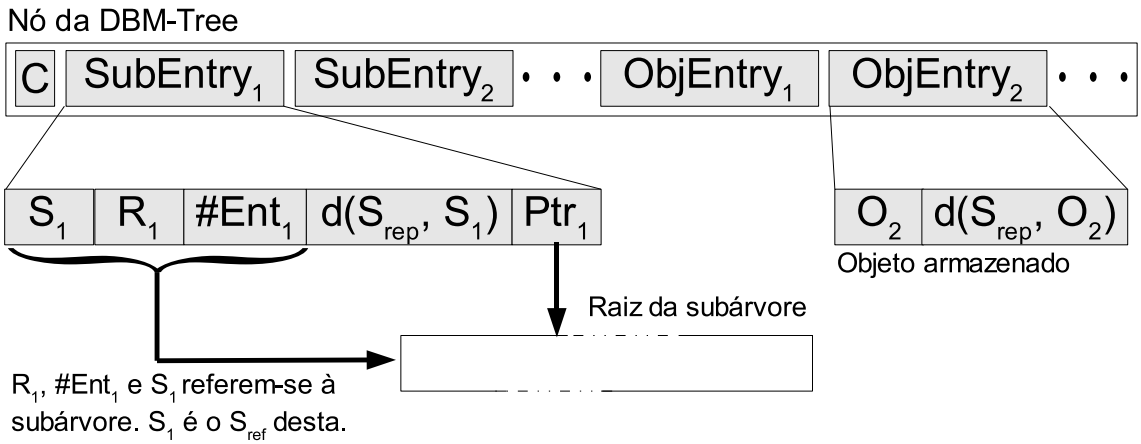

Figura 3.4: Estrutura do nó da *DBM-Tree*. Este nós é capaz de armazenar C entradas dos tipos *SubEntry* ou *ObjEntry*. As entradas do tipo *SubEntry* armazenam as informações referentes às subárvores enquanto as do tipo *ObjEntry* armazenam os objetos. As informações referentes ao objeto representativo, o raio e o número de objetos de uma subárvore estão sempre armazenadas no nível imediatamente superior.

em estruturas balanceadas cujo grau de sobreposição é maior.

3.3.1 Organização da DBM-Tree

A *DBM-Tree* manifesta-se na forma de uma árvore multivias, cujo balanceamento é flexibilizado em regiões com maior densidade de objetos. Cada nó desta estrutura ocupa uma página de disco de tamanho fixo e constitui a raiz de uma subárvore coberta por uma região definida por um objeto representativo e um raio.

Diferentemente da *Slim-Tree*, esta estrutura possui apenas um tipo de nó, capaz de armazenar não só ligações para as subárvores mas também objetos que não são cobertos por nenhuma delas. Em outras palavras, os objetos podem ser armazenados em todos os níveis da estrutura. A Figura 3.4 ilustra a organização do nó da *DBM-Tree*.

3.3.2 Algoritmos de Construção da DBM-Tree

Assim como a *Slim-Tree*, a *DBM-Tree* é uma estrutura dinâmica, capaz de receber novos objetos em sua estrutura ao longo de toda a sua vida. Ao receber um novo objeto para ser adicionado, os algoritmos de construção procuram na estrutura um nó capaz de abrigá-lo.

Um nó só se qualifica para armazenar um objeto quando este cobre um objeto sem que nenhuma de suas subárvores o façam. Além disto, este nó pode estar localizado em qualquer nível da estrutura. Em outras palavras, um nó só se qualifica a armazenar um objeto quando este é coberto pelo raio do nó e nenhuma de suas subárvores cobrem o objeto. Se não existir nenhum nó com estas características, o nó que puder receber este objeto com o menor aumento de raio será escolhido.

Caso o nó escolhido não seja capaz de armazenar o objeto por problemas de espaço físico, o nó é dividido em dois ou mais nós, promovendo objetos e subárvores para os níveis superiores ou inferiores da estrutura quando necessário. Porém, estas políticas ainda eram objeto de estudo até o fechamento deste documento.

A construção da *DBM-Tree* prevê o crescimento da estrutura nos dois sentidos, tanto para os níveis de cima quanto para os níveis de baixo da árvore, portanto, a variação do balanceamento da estrutura deve ser controlada rigorosamente para evitar que os algoritmos de construção gerem árvores degeneradas. Este nível de variação pode ser controlado pelos usuários no momento da criação da estrutura.

3.3.3 Algoritmos de Busca na DBM-Tree

Assim como todas as árvores métricas, as buscas na *DBM-Tree* são simples. Os algoritmos percorrem as subárvores cujas regiões cobertas possuem intersecção não vazia com as regiões definidas pelos critérios da busca.

Todas estas subárvores são percorridas e os objetos armazenados em cada nó destas são testados, primeiramente segundo a desigualdade triangular e, caso sejam candidatos em potencial, tem sua distância com o objeto de exemplo da consulta calculada e avaliada. Se a distância do objeto testado com o objeto de exemplo satisfizer as condições impostas pela consulta, este objeto é adicionado ao conjunto de resposta.

O comportamento das consultas da *DBM-Tree* é muito similar ao comportamento das consultas descritos para a *Slim-Tree* na Seção 3.2.3. Porém, a *DBM-Tree* adiciona os objetos pertencentes à resposta da consulta durante todo o trajeto percorrido na estrutura enquanto a *Slim-Tree* só faz isso quando encontra um nó folha.

3.3.4 Considerações Sobre a Descrição da DBM-Tree

Por se tratar de uma estrutura ainda em desenvolvimento e muitos de seus detalhes ainda não estão totalmente consolidados, o presente texto descreve apenas as características básicas da estrutura, deixando os seus detalhes para serem descritos no Capítulo 7 apenas quando estes forem realmente necessários para o entendimento dos resultados.

O trabalho de desenvolvimento da *DBM-Tree* tem sido o melhor campo de testes para o sistema de visualização proposto nesta dissertação, pois tem possibilitado o estudo do aproveitamento deste sistema em um ambiente real de desenvolvimento de um MAM.

3.4 Visualizando Estruturas Métricas

A visualização de estruturas de dados é uma ferramenta valiosa tanto para o seu desenvolvimento quanto para o seu aprendizado. Representações simples, como as utilizadas

para uma árvore B são bastante satisfatórias para conjuntos cuja relação de ordem total existe. Estas relações podem ser facilmente exibidas através do posicionamento dos nós no diagrama.

Em métodos de acesso métricos e mesmo nos espaciais, diagramas simples são insuficientes para apresentar visualmente todo o estado das estruturas. Utilizando esse modelo, é possível mostrar a organização estrutural das árvores mas não os motivos que levaram a tomar esta forma, pois o posicionamento dos nós no diagrama não consegue representar informações sobre o conjunto de dados.

Para SAM indexando objetos de dimensionalidade baixa (entre 1 e 4 dimensões) e MAM indexando objetos com até 3 dimensões e utilizando a função de distância euclidiana, é possível criar representações geométricas diretas, sem a necessidade de transformações nos dados. Para dados de alta dimensionalidade ou espaços métricos arbitrários, é necessário incorporar técnicas de visualização de informação às técnicas convencionais de visualização de estruturas de dados baseadas em grafos para a obtenção de resultados mais efetivos.

A primeira ferramenta proposta para a visualização de uma árvore métrica foi apresentada em [Traina et al., 2002b] para a *Slim-Tree*. Esta ferramenta cria uma representação geométrica da árvore de forma a exibir as relações de distância entre os objetos, permitindo ao usuário avaliar a distribuição dos objetos, os raios de cobertura de cada nó e o nível de sobreposição entre as regiões.

O sistema utiliza o *FastMap* [Faloutsos & Lin, 1995] (o algoritmo será descrito em detalhes na seção 4.3) para criar um mapeamento dos objetos da árvore, que habitam um espaço métrico arbitrário, para um espaço euclidiano de 2 ou 3 dimensões. Para cada objeto, o algoritmo gera pontos no espaço euclidiano, chamados de imagem do objeto, de modo a preservar o máximo possível as distâncias existentes no espaço métrico original, condensando assim a informação relativa às distâncias entre os objetos na distribuição espacial das imagens. Além da distribuição dos objetos, o sistema extrai informações sobre a organização da árvore, como os raios de cobertura e outras informações estruturais.

Em sua primeira versão, o sistema era capaz de mostrar ao usuário os objetos contidos na estrutura e sua "disposição" no espaço métrico e os raios de cobertura dos nós índice. Uma parte do sistema, embutido na implementação da *Slim-Tree*, percorria a árvore extraindo as informações desta que eram então armazenadas em um arquivo que poderia ser exibido pelo *GNUPlot*[1] Este recurso de visualização mostrou-se um aliado indispensável durante a fase de desenvolvimento da *Slim-Tree*, permitindo ao seus autores acompanhar, mesmo que de modo limitado, o comportamento da árvore, identificar características adicionais da estrutura e problemas de implementação com o uso de inspeção visual.

[1]Aplicação GNU para impressão de funções (http://www.gnuplot.info/).

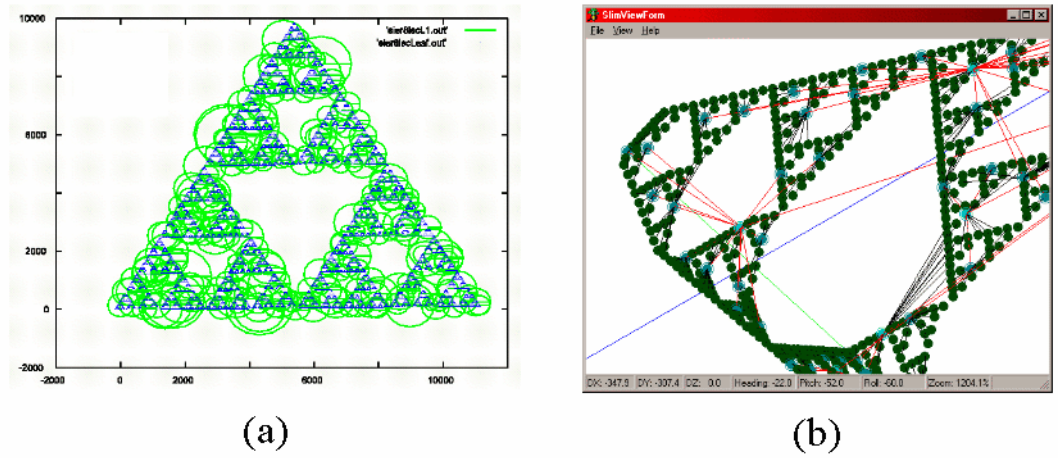

Figura 3.5: Telas das ferramentas de visualização da *Slim-Tree*. (a) é uma tela da primeira versão baseada no *GNUPlot* e (b) é uma tela da segunda versão baseada em um visualizador dedicado.

Em sua última versão, a ferramenta ganhou um visualizador dedicado que permite maior interatividade (controle de nível de detalhes, uso mais apropriado de cores e um sistema de controle de visões interativo, baseado no *OpenGL*[2]) e a exibição de mais detalhes sobre as árvores (ligações entre os nós e níveis ocupados pelos objetos) [Chino & Traina, 2001].

Ambas as versões permitem somente a visualização estática da estrutura da árvore, criando apenas representações completas da estrutura em um dado instante. A Figura 3.5 ilustra telas das duas versões de visualização sobre uma dada árvore.

O sistema *MAMView*, proposto por este trabalho, visa romper estas limitações, permitido não só a exibição de representações estáticas como também representações dinâmicas das estruturas, ou seja, o funcionamento de seus algoritmos. O *MAMView* será capaz ainda de gerar representações de outras estruturas além da *Slim-Tree*.

3.5 Conclusão

Responder às consultas por similaridade em SGBD exige a elaboração de métodos de indexação capazes de respondê-las de modo eficiente. Surgem então várias abordagens, entre as quais as mais promissoras são os métodos de acesso métricos (MAM). Dentre os diversos MAM propostos, a *Slim-Tree* é um dos mais eficientes, sendo uma estrutura dinâmica que possui um baixo grau de sobreposição.

A *Slim-Tree* é um MAM que se manifesta como uma árvore balanceada, onde cada nó é armazenado em uma página em disco de tamanho fixo. Esta estrutura é uma

[2]O *OpenGL* é uma biblioteca gráfica com padrão aberto desenvolvida para a criação de gráficos tridimensionais interativos. Há implementações para inúmeros sistemas.

evolução da *M-Tree*, possuindo um novo algoritmo de quebra de nós e meios para avaliar e diminuir o grau de sobreposição entre os nós.

A *DBM-Tree* é um outro MAM, ainda em fase de desenvolvimento, que visa criar estruturas com menor nível de sobreposição, permitindo uma flexibilização do balanceamento da estrutura em regiões de maior densidade de objetos. Este método de acesso métrico é especialmente relevante para este projeto por encontrar-se ainda em fase de desenvolvimento, possibilitando a validação do sistema de visualização proposto em um ambiente real de desenvolvimento.

Assim como outras estruturas de dados, os MAM podem se beneficiar consideravelmente de recursos de inspeção visual, como o proposto por este trabalho. A *Slim-Tree* foi a primeira estrutura métrica a se beneficiar deste tipo de recurso, porém, de forma bastante limitada. O sistema *MAMView*, proposto por este trabalho, visa suplantar estas limitações e ainda possibilitar que outras árvores métricas possam se beneficiar das facilidades que a visualização de estruturas e seu comportamento proporcionam.

Capítulo

4

Visualizando Dados Métricos

4.1 Introdução

Embora conceitualmente simples, os espaços métricos são consideravelmente abstratos ao ser humano. A forte natureza visual das pessoas faz com que estas tentem criar uma representação "visível" das idéias que envolvem o seu cotidiano [Ware, 2000]. É difícil para uma pessoa imaginar uma representação visual de um espaço onde existem apenas objetos e distâncias sem a ajuda de alguma ferramenta capaz de sintetizar a complexa teia formada apenas pelas distâncias entre os objetos.

A maneira mais eficiente para sintetizar estas informações é através de uma representação geométrica em 1, 2 ou 3 dimensões, que são as dimensões com que os seres humanos lidam em seu cotidiano. A informação sobre as distâncias é condensada no posicionamento dos objetos no espaço, sendo facilmente assimilada por um usuário. Isto pode ser obtido, de modo bastante satisfatório, por algoritmos denominados de **mapeadores de distância**, cuja função é transformar um espaço métrico arbitrário em um espaço dimensional, geralmente o euclidiano, tentando preservar o máximo possível as relações de distância existentes no espaço original.

4.2 Algoritmos Mapeadores de Distância

Dados dois espaços métricos $M_{src} =< \mathfrak{D}_{src}, d_{src}() >$ e $M_{trg} =< \mathfrak{D}_{trg}, d_{trg}() >$, pode se definir um **algoritmo mapeador de distâncias** como uma função $f(x) : M_{src} \rightarrow M_{trg}$

que mapeia um elemento de M_{src} para um de M_{trg} de modo que $\forall o_i, o_j \in \mathfrak{D}_{src}$, tem-se $d_{src}(o_i, o_j) \approx d_{trg}(f(o_i), f(o_j))$.

Em outras palavras, mapeiam os objetos de um espaço para outro buscando preservar o máximo possível as distâncias entre os objetos nos dois espaços. Algoritmos como a escala multidimensional [Kruskal & Wish, 1978], o *MetricMap* [Wang et al., 1999], o *SparseMap* [Hristescu & Farach-Colton, 1999] e o *FastMap* [Faloutsos & Lin, 1995] podem ser incluídos nesta categoria.

A **Escala Multidimensional** (MDS) visa mapear os objetos do espaço métrico original para um espaço euclidiano com k dimensões, tendo como objetivo a minimização do *stress* (ver a seção 4.3.5) através de um processo de otimização. É capaz de criar mapeamentos muito precisos, porém apresenta um custo computacional alto, por ser de ordem quadrática, o que o torna inviável para grandes conjuntos de dados.

O **SparseMap** foi criado originalmente para mapear cadeias de proteínas em um espaço euclidiano k-dimensional. Seu algoritmo consiste em mapear os objetos de acordo com uma série de subconjuntos do conjunto de objetos original previamente definidos. Infelizmente, os mapeamentos resultantes não garantem um limite mínimo para *stress*, tornando sua aplicação limitada.

O **MetricMap** mapeia os espaços métricos para um espaço chamando de **pseudo-euclidiano** k-dimensional. Cada elemento é associado a um vetor k-dimensional como em um espaço euclidiano, porém a função de distância que rege este espaço métrico não é uma função de distância euclidiana, fazendo com que a visualização dos pontos resultantes seja pouco significativa para uma pessoa.

O **FastMap** é um algoritmo capaz de mapear espaços métricos em espaços euclidianos k-dimensionais, minimizando o *stress*, como a escala multidimensional, porém a um custo linear, isto é, na ordem de $O(N)$ (onde N é o número de objetos do conjunto a ser mapeado).

Como descrito na seção 4.1, aplicações de visualização de espaços métricos necessitam de algoritmos mapeadores de distância capazes de mapear os objetos de um espaço métrico arbitrário M_{src} para um espaço euclidiano de 1, 2 ou 3 dimensões. Com exceção do *MetricMap*, todos os algoritmos citados podem ser aplicados na visualização de espaços métricos, porém as características do *FastMap* colocam-no como o mais indicado para aplicações de visualização de grandes volumes de dados métricos.

4.3 O Algoritmo FastMap

O *Fastmap* foi criado por Faloutsos e Lin [Faloutsos & Lin, 1995] com o intuito de ser utilizado para indexação de dados multimídia, mineração e visualização de dados que habitam em espaços métricos. O objetivo do algoritmo é mapear um conjunto de objetos

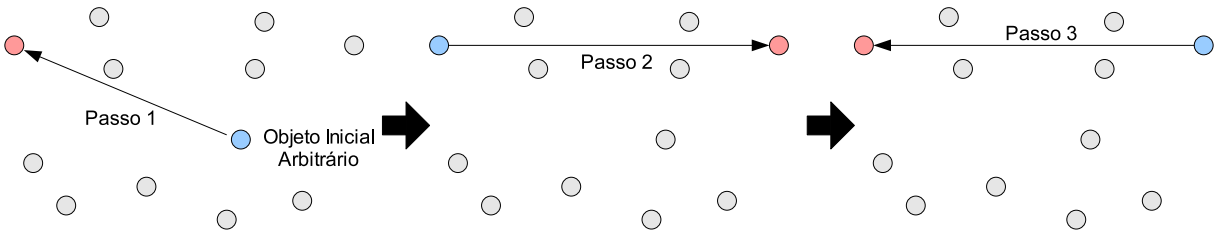

Figura 4.1: Processo de escolha dos pivôs do *FastMap*. O azul indica o objeto a ser comparado com os demais enquanto o vermelho indica o objeto mais distante encontrado em cada passo. Apenas 3 passos foram necessários para encontrar o par de objetos mais distantes neste conjunto. O último passo apenas constatou que o par escolhido no passo 2 é aceitável.

de um espaço métrico em um espaço euclidiano k-dimensional, tentando preservar ao máximo as distâncias originais entre os objetos. Em outras palavras, cada objeto do espaço original é associado a um ponto no espaço euclidiano destino, chamado de imagem do objeto. A distância entre as imagens de dois objetos deve ser a mais próxima possível da imagem dos objetos no espaço de origem.

O mapeamento dos objetos em coordenadas de um espaço euclidiano é dado pela projeção dos objetos em eixos ortogonais definidos por pares de objetos especialmente escolhidos, que são chamados de **pivôs**. Para cada eixo, deve ser eleito um par de pivôs que o definirá.

Este é um algoritmo iterativo que tem como entrada um conjunto de objetos, uma função de distância métrica $d()$ e o número de dimensões para o espaço de destino k. Cada iteração do algoritmo mapeia os objetos sobre um único eixo, escolhendo o par de pivôs (ver seção 4.3.1) que é utilizado para mapear as coordenadas de todos os objetos do conjunto para aquele eixo. A cada iteração, o processo é repetido alterando apenas o modo como as distâncias são calculadas (ver seção 4.3.3). O ciclo é repetido até que os objetos estejam mapeados sobre os k eixos distintos. Os detalhes de cada um dos passos do algoritmo serão apresentados nas próximas seções.

4.3.1 Escolhendo os Pivôs

O primeiro passo na execução de cada iteração do algoritmo é a escolha dos objetos pivôs. Para melhorar a precisão do mapeamento, é necessário que estes objetos sejam os dois objetos que apresentam a maior distância possível entre si. Assim, os eixos envolverão todo o conjunto de dados e a projeção será mais apropriada.

O modo mais simples para a obtenção deste par é computar as distâncias entre todos os objetos, dois a dois, e escolher o par com a maior distância. Entretanto, este algoritmo é de ordem $O(N^2)$, tornando o algoritmo completo de mapeamento muito caro.

O *FastMap* propõe o uso de uma heurística que consiste em escolher um objeto

qualquer do conjunto e procurar pelo objeto mais distante deste. Descarta-se o objeto escolhido previamente e repete-se o processo para o objeto encontrado. Este procedimento é repetido por um certo número de vezes (o artigo original sugere 5) ou até que o objeto inicial seja o mais distante do encontrado e vice-versa. A Figura 4.1 ilustra a execução do do algoritmo para um pequeno conjunto de dados bidimensionais.

4.3.2 Mapeando Objetos no Eixo

Considerando as distâncias entre os pivôs (objetos o_a e o_b) e o objeto a ser mapeado (objeto o_i) como os vértices de um triângulo (Figura 4.2), o valor da coordenada do objeto neste eixo pode ser calculado utilizando a Equação 4.2, que é derivada diretamente da lei dos cossenos (Equação 4.1).

$$d_{bi}^2 = d_{ai}^2 + d_{ab}^2 - 2x_i d_{ab} \tag{4.1}$$

$$x_i = \frac{d_{ai}^2 + d_{ab}^2 - d_{bi}^2}{2d_{ab}} \tag{4.2}$$

Em cada iteração, todos os objetos do conjunto são projetados segundo esta equação.

4.3.3 Mapeando os Demais Eixos

O mapeamento dos demais eixos se dá pela modificação da função de distância utilizada em cada interação e pelos resultados das iterações anteriores. Na primeira iteração, a função de distância utilizada pelo *FastMap* é exatamente a função do espaço original. Nas iterações seguintes, o valor das distâncias entre as imagens dos objetos passa a ser considerada.

A Equação 4.3 mostra como a função de distância de cada iteração, representada

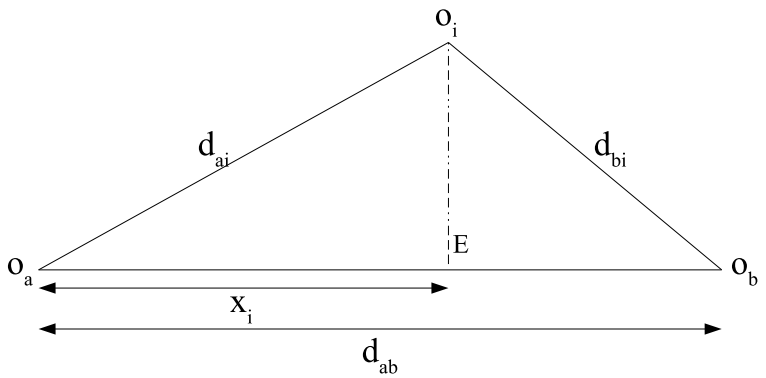

Figura 4.2: Mapeando o objeto o_i no eixo formado pelos pivôs o_a e o_b.

por $d'()$, é modificada. A distância entre os objetos o_i e o_j passa a incorporar a distância entre as imagens x_i e x_j já calculadas nos passos anteriores.

$$d'(o_i, o_j) = \sqrt{d(o_i, o_j)^2 - (x_i - x_j)^2} \tag{4.3}$$

Um dos efeitos deste procedimento é que a precisão do mapeamento aumenta conforme o valor de k se aproxima do valor da dimensão do espaço original.

4.3.4 Mapeando Objetos Externos ao Conjunto Inicial

O *FastMap* utiliza um conjunto inicial de objetos como entrada e seu algoritmo tem como saída o mapeamento destes objetos. Em princípio, isto limita o algoritmo a ter todos os objetos presentes no momento do mapeamento. Entretanto é possível mapear objetos externos ao conjunto inicial se a informação sobre os objetos que compõem cada um dos eixos for armazenada ao final do processo.

As coordenadas da imagem do novo objeto são obtidas projetando-o nos mesmos eixos definidos pelo mapeamento inicial, incorporando-o ao conjunto como se este "estivesse presente" na construção do mapeamento original. Isto preserva a disposição dos objetos mapeados anteriormente, permitindo o uso da técnica para a visualização comparativa do conjunto, mesmo depois da adição de novos objetos.

As conseqüências desta abordagem podem recair sobre a qualidade do resultado visual obtido. Isso acontece se a adição de um novo objeto interferir na escolha dos pivôs utilizados no mapeamento. Porém este detalhe constitui um preço pequeno a ser pago pela manutenção dos objetos em suas posições anteriores.

4.3.5 Estimativas de Erros no Mapeamento

Para avaliar a qualidade do mapeamento, o *FastMap* utiliza a medida de *stress* definida por Kruskal [Kruskal & Wish, 1978]. A Equação 4.4 mostra como o cálculo é feito. d_{ij} denota a distância entre os objetos o_i e o_j enquanto $\hat{d}_{ij}$ denota a distância entre as imagens destes dois objetos. A Figura 4.3 ilustra o significado geométrico do *stress*.

$$stress = \sqrt{\frac{\sum_{ij}(\hat{d}_{ij} - d_{ij})^2}{\sum_{ij} d_{ij}^2}} \tag{4.4}$$

Por constituir um somatório dos quadrados das diferenças entre as distâncias originais e as das imagens, valores pequenos indicam mapeamentos mais próximos ao original (zero constitui o mapeamento perfeito). Portanto, quanto menor o valor do *stress*, menor a distorção causada pelo mapeamento e, conseqüentemente, melhor será a sua qualidade.

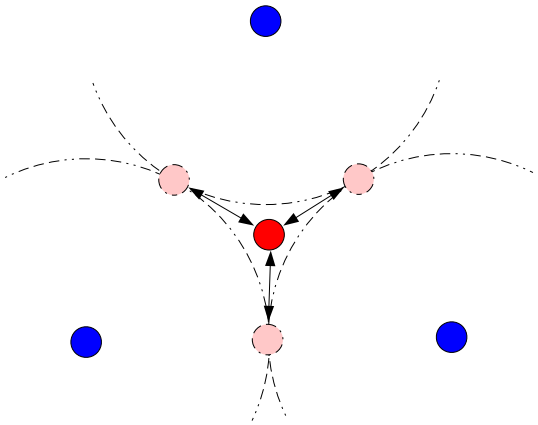

Figura 4.3: Representação gráfica do *stress*. As setas duplas representam a diferença entre a imagem ótima (em vermelho claro) e a imagem mapeada (em vermelho) com relação aos demais objetos (em azul).

4.3.6 Limitações do FastMap

Quando Faloutsos e Lin desenvolveram o *FastMap*, eles assumiram que todos os espaços mapeados poderiam ser interpretados como espaços euclidianos com dimensão desconhecida. No entanto, esta afirmação só é válida quando o espaço original é isométrico a um espaço euclidiano [Hjaltason & Samet, 2000].

Esta limitação cria a possibilidade de que certos conjuntos de dados possam causar a ocorrência de valores negativos na função de distância $d'()$ definida pelo algoritmo (ver 4.3.3) e utilizada para mapear os eixos. Isto faz com que a projeção apresente problemas na escolha dos pivôs, causando grandes distorções no mapeamento.

A conseqüência mais séria deste problema recai quando se utiliza o *FastMap* como ferramenta de indexação. Quando os espaços de origem são isométricos ao espaço euclidiano, as distâncias do espaço mapeado são sempre menores ou iguais às distâncias do espaço original. Se o espaço não for isométrico e a função de distância $d'()$ for negativa para algum dos objetos, a distância projetada possivelmente será maior que a distância original, causando a ocorrência de falsos negativos em buscas por similaridade, ou seja, objetos que devem fazer parte das respostas são descartados. Felizmente seu uso como algoritmo de visualização é pouco comprometido por este problema, pois a percepção visual das pessoas não é afetada por isto (ver seção 4.4).

4.4 Efeitos do Stress na Percepção Humana

O cérebro humano, assim como o de outros seres vivos que possuem a visão como principal meio de captar informações sobre o ambiente, é capaz de assimilar rapidamente informações exibidas visualmente, identificando objetos, padrões e agrupamentos com bastante precisão e eficiência.

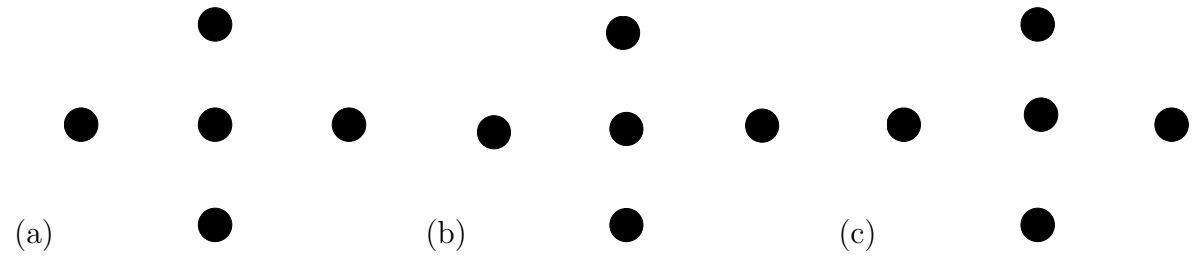

Figura 4.4: Figura com um conjunto de 5 pontos. (a) é a representação ideal; (b) é a representação com um pouco de *stress* uniformemente distribuído; e (c) é a representação com a mesma quantidade de *stress* de (b), porém com distribuição não uniforme.

Aplicações de visualização aproveitam esta enorme capacidade para exibir informações de modo mais natural para as pessoas. Porém, as limitações para a criação de representações visuais precisas aumentam conforme a complexidade dos dados aumenta, causando um certo nível de distorções nas informações exibidas.

O *Stress* (ver Seção 4.3.5 para mais detalhes) associado aos resultados de mapeamentos obtidos com algoritmos como o *FastMap* constitui um dos meios mais simples para avaliar a qualidade visual dos mapeamentos. Pode-se afirmar com segurança que valores muito grandes de *Stress* em um mapeamento significam que a qualidade do mapeamento é bastante baixa, podendo inviabilizá-lo para a visualização do conjunto de dados. Porém, afirmar que a qualidade de um mapeamento é boa apenas porque o valor do *Stress* é baixo pode ser uma atitude arriscada.

O sistema visual humano é pouco sensível a distorções no mapeamento se estas forem uniformemente distribuídas (Figura 4.4(b)), porém, se a distribuição dos erros no mapeamento estiver concentrada em poucos elementos do mapeamento (Figura 4.4(c)), sua qualidade visual será baixa, mesmo que este último mapeamento tenha um valor de *Stress* menor que o do primeiro caso.

Isto se deve ao fato de que o cálculo do *Stress* (Equação 4.4) leva em consideração apenas o valor médio das diferenças das distâncias entre os objetos do espaço original e seus mapeamentos. Uma abordagem melhor para avaliar a qualidade dos mapeamentos constitui em utilizar não só o *stress* mas também a variância e o valor máximo destas diferenças, pois permitiria avaliar não só o valor médio dos erros mas também ter algumas informações sobre sua distribuição. Porém, o estudo desta proposta foge do escopo deste trabalho.

4.5 Conclusão

É difícil para uma pessoa avaliar a distribuição de objetos em um espaço métrico apenas com tabelas e números. Criar uma representação visual do conjunto permite sintetizar estas informações de modo mais facilmente assimilável. Dentre todos os espaços métricos

possíveis, existem 3 em especial que são naturalmente entendidos por qualquer ser humano, os espaços euclidianos de 1, 2 e 3 dimensões. Porém é necessário o uso de ferramentas que permitam "transformar" um espaço métrico arbitrário em um espaço euclidiano, mantendo a informação da distância o mais fiel possível.

Existem vários algoritmos capazes de realizar tal mapeamento, entre eles o *FastMap* mostra-se o mais adequado para a atividade de visualização, pois é capaz de mapear espaços métricos arbitrários em espaços euclidianos k-dimensionais a um custo computacional baixo. Porém, qualquer algoritmo mapeador de distâncias capaz de mapear os objetos para espaços k-dimensionais pode ser utilizado.

É impossível criar um mapeamento absolutamente preciso de um espaço métrico arbitrário para um espaço euclidiano com 1, 2 ou 3 dimensões. Perdas existirão e devem ser sempre consideradas.

A qualidade dos mapeamentos utilizados por uma aplicação de visualização não pode ser inferida apenas pelo valor do *Stress* do mapeamento. A distribuição da distorção deve ser considerada também. O ser humano é capaz de reconhecer visualmente padrões e tendências em conjuntos de dados quando as distorções são uniformemente distribuídas, entretanto, quando as distorções encontram-se concentradas em alguns poucos elementos, esta capacidade é bastante prejudicada. Isto não significa que os valores de distorção inferidos pelo *Stress* possam ser ignorados.

Capítulo

5

Visualizando Estruturas Métricas

5.1 Introdução

Um dos melhores modos de visualizar uma estrutura de dados, bem como seu comportamento, é através da exibição de representações visuais das várias configurações da estrutura durante a evolução de uma operação típica. Este tipo de representação visual pode ser utilizada em salas de aula para ensinar o comportamento de estruturas simples como listas, árvores binárias e até mesmo estruturas mais complexas como árvores B entre outras. Ferramentas como o *AMDB* [Shah et al., 1999, Kornacker et al., 2003] e a *JDSL Visualizer* [Baker et al., 1999] são bons exemplos deste tipo de aplicação.

Baseando-se nesta idéia, pode-se perceber que aplicações similares, voltadas para estruturas métricas, trariam uma enorme contribuição para a compreensão dos MAM, podendo ser utilizadas para atividades de ensino e, principalmente, para auxiliar o desenvolvimento destas estruturas.

A capacidade de explorar e analisar visualmente o comportamento e a organização de estruturas complexas como os MAM reduz consideravelmente o tempo necessário para entender suas propriedades, e pode levar à descoberta de outras propriedades que não seriam percebidas de outras formas, como o uso de "dumps" textuais e acompanhamento de algoritmos entre outras.

Uma ferramenta capaz de exibir o funcionamento de um MAM visualmente seria de vital importância para o estudo destas estruturas, especialmente aquelas que ainda se encontram em estágio de desenvolvimento.

5.2 Construindo um Sistema de Visualização de MAM

Sistemas voltados para a visualização de estruturas de dados possuem dois aspectos fundamentais. Um destes aspectos envolve a criação de um modelo capaz de representar a organização e o comportamento de uma estrutura e ser capaz de gerar representações visuais desta, pois a maioria das estruturas de dados não são apropriadas para este fim. O outro aspecto cobre a forma de apresentação visual destas representações, ou seja, como criar figuras que possam sintetizar as características desta estrutura.

O primeiro aspecto, relativo à criação do modelo para representar a estrutura e seu comportamento, pode ser abordado de dois modos distintos. Em um deles, utiliza-se o conhecimento prévio sobre uma determinada estrutura para criar uma versão modificada desta, que seja capaz de gerar as representações visuais desejadas, ou seja, a estrutura é modificada para acomodar esta função. Esta abordagem é capaz de gerar excelentes representações visuais da estrutura e seu comportamento, porém exige um vasto conhecimento da estrutura e de seu comportamento.

A outra abordagem consiste em criar um modelo genérico da estrutura e extrair informações sobre o seu comportamento diretamente dela mesma. Em outras palavras, este modelo genérico é capaz de utilizar as informações extraídas da estrutura para simular seu comportamento e gerar as representações visuais desejadas. As representações visuais obtidas deste modo não são tão detalhadas como as obtidas pela primeira abordagem. Porém, esta última abordagem pode ser aplicada a estruturas pouco conhecidas, como é o caso dos MAM.

O segundo aspecto, referente à representação visual, consiste em criar maneiras de representar o estado e o comportamento de estruturas utilizando figuras e/ou outros recursos visuais. Estas representações devem ser capazes de sintetizar, se não todas, pelo menos as características principais de uma estrutura, de modo que possam ser facilmente entendidas por um ser humano. Em geral, estas representações utilizam figuras estáticas para representar o estado de uma estrutura em um determinado momento e sintetizam as transformações nela ocorridas durante um procedimento por meio de seqüências de representações dos estados da estrutura antes e depois de cada alteração.

Estruturas de dados convencionais, onde há a relação de ordem entre seus elementos podem ser visualizadas através de representações geométricas simples, como as utilizadas para representar árvores binárias e árvores B. Porém, estruturas métricas, exigem modelos mais elaborados devido à inexistência de representações gráficas intuitivas e diretas dos conjuntos de dados em espaços métricos. Portanto, maneiras de representar visualmente estas estruturas devem ser definidas.

O modelo necessário para representar a estrutura e o comportamento de um MAM,

necessário para a aplicação proposta será detalhada na Seção 5.3, enquanto a definição da representação visual de um MAM será descrita na Seção 5.4. Utilizando estes conceitos, é possível definir uma arquitetura básica para os sistemas de visualização de MAM que será descrita em maiores detalhes na Seção 5.6.

5.3 Modelo para Representar a Organização e o Comportamento de MAM

Como descrito na Seção 5.2, criar representações visuais de estruturas pouco conhecidas, como os MAM, exige a criação de um modelo externo capaz de representar a organização e o comportamento destas estruturas, de modo que ele possa ser utilizado para criar estas representações.

Este modelo precisa ser capaz de abstrair o maior número possível de estruturas métricas e reproduzir todas as possíveis transformações que estes MAM possam promover sobre a organização das estruturas. Estes fatores permitem a reprodução destes MAM e também de vários que ainda estão em desenvolvimento mas que compartilhem as características básicas cobertas por este modelo.

A definição de tal modelo está dividida em dois componentes distintos, um modelo genérico de organização da estrutura e um modelo de comportamento, ou seja, das transformações que podem ser realizadas sobre o modelo de organização.

5.3.1 Modelo Genérico de Organização

Apesar da grande variedade de princípios de funcionamento e modos de organização, a grande maioria dos MAM compartilham características suficientes para que um modelo genérico de organização completo possa ser definido.

Este modelo baseia-se no fato de que a maioria das estruturas métricas tem como objetivo a indexação de dados armazenados em SGBD. Conseqüentemente, estas estruturas tendem a se manifestar como árvores multivias, que dividem o espaço indexado em sub-regiões, que, no caso das estruturas métricas, é definido por pares de objetos representativos e raios de cobertura. Além disto, cada nó destas árvores pode armazenar um certo número de objetos que podem ou não estar concentrados em determinados níveis da estrutura.

Assim, o modelo proposto deve ser capaz de representar estruturas com as seguintes características:

- Estruturas organizadas em nós com identificadores únicos;
- Cada nó pode ou não ter um pai, porém, quando o tiver será único;

- Cada nó pode ter qualquer número de filhos;
- Cada nó pode representar uma divisão do espaço em regiões definidas por um ou mais pares de objetos/raios (representativos);
- Cada objeto pode estar associado a um único nó ou a nenhum (para o caso de inconsistências);
- Podem existir objetos repetidos;
- Nós e objetos estão localizados a uma certa distância da raiz da estrutura (nível);
- O resultado de uma busca é composto por um ou mais objetos de exemplo, um ou mais raios e uma lista de objetos de resposta.

Para garantir que este modelo possa ser aplicado ao maior número de estruturas possível, nenhuma outra restrição pode ser adicionada. Um modelo capaz de representar nós, objetos e regiões com estas características é capaz de assumir todas as configurações que os MAM cobertos podem assumir.

Evidentemente, existem outras propriedades particulares a cada MAM associadas a nós e objetos. Porém, estas propriedades são, em sua maioria, representações de informações adicionais necessárias para operações sobre as estruturas, não influindo em sua organização estrutural.

5.3.2 Modelo Genérico de Comportamento

O comportamento de uma estrutura qualquer pode ser sintetizado pela seqüência de mudanças sofridas na configuração de seus componentes (nós, objetos e ligações) durante a sua operação. O modelo de organização descrito na Seção 5.3.1 é capaz apenas de representar a configuração das estruturas em um determinado momento.

Para representar o comportamento de uma estrutura é necessário definir um conjunto de operações sobre o modelo genérico de organização. Estas operações são:

- Adição, eliminação de um nó e alteração de suas propriedades;
- Adição, eliminação de um objeto e alteração de suas propriedades;
- Atualização do estado de uma consulta.

Embora bastante simples, estas operações representam todas as transformações que o modelo genérico de organização pode sofrer. Em outras palavras, estas operações simples podem ser combinadas de forma a reproduzir mesmo transformações bastante complexas.

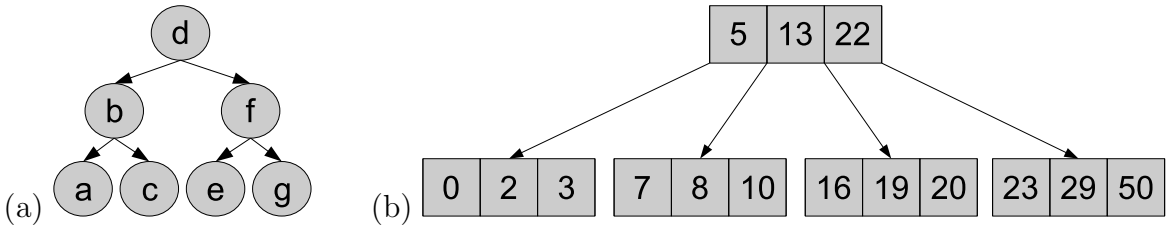

Figura 5.1: Exemplos de representação visual de: (a) uma árvore binária, (b) uma árvore B.

Estas operações devem ser executadas sem nenhum tipo de restrição pois não é possível definir um conjunto genérico de restrições válidos a todos os MAM cobertos pelo modelo de organização. Além disto, espera-se que o modelo de comportamento seja capaz de representar não só as transformações conhecidas, mas as desconhecidas também. Isto é essencial para a detecção de operações inconsistentes e para o estudo de estruturas ainda em desenvolvimento.

A validação do comportamento correto de uma estrutura não deve ser contemplada pelo modelo de comportamento, mas por uma entidade externa, como um usuário ou um módulo de validação. Não cabe a este modelo, questionar as mudanças de estado ocorridas nas estruturas, uma vez que situações não esperadas podem estar realmente ocorrendo.

5.4 Representação Visual de um MAM

A representação visual de uma estrutura consiste em sintetizar as principais informações relevantes a seu funcionamento em uma apresentação visual que possa ser facilmente assimilada por um ser humano.

No caso de estruturas tradicionais, como árvores binárias e árvores B, as representações gráficas são relativamente simples. Os nós podem ser representados por figuras contendo os elementos armazenados, cada uma destas figuras está ligada a outra, seu pai, por linhas, representando as ligações. A disposição geométrica dos elementos armazenados abstrai a relação de ordem existente nos conjuntos de dados (Figura 5.1).

Nos MAM, o primeiro problema enfrentado é a falta de uma representação visual direta para os dados a serem visualizados. Uma abordagem similar à utilizada para árvores B poderia ser utilizada, porém, a mais importante característica destas estruturas, que é a relação de distância entre os objetos indexados não seria exibida por esta representação. É preciso definir um outro modo de representar graficamente os MAM.

Para poder sintetizar as relações de distância é preciso, primeiramente, ser capaz de visualizar estas relações dentro dos conjuntos de dados indexados. Felizmente, é possível criar tais representações visuais utilizando algoritmos mapeadores de distância, como o *FastMap* descrito no Capítulo 4.

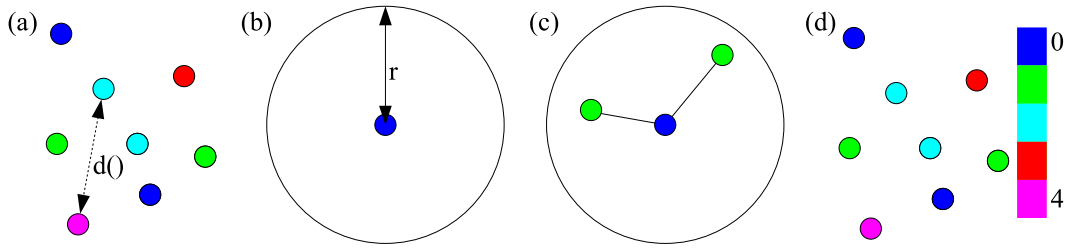

Figura 5.2: Exemplos de representações visuais de um MAM. (a) mostra a abstração das distâncias de acordo com a disposição geométrica; (b) ilustra a representação de uma área definida por um objeto e um raio; (c) ilustra a representação de ligações entre elementos; e (d) ilustra o uso de cores para sintetizar as informações sobre a altura do elemento.

Estes algoritmos são capazes de mapear conjuntos de dados em espaços métricos arbitrários para pontos em espaços euclidianos com 1, 2 e três dimensões (que também são espaços métricos) tentando manter as distâncias entre os objetos. Estes pontos podem ser utilizados para criar representações visuais destes conjuntos que podem ser mais facilmente entendidos por um ser humano (as distâncias são representadas pelo posicionamento dos pontos).

Portanto, pode-se criar figuras para representar os componentes e a organização de uma estrutura métrica baseando-se nos mapeamentos dos conjuntos de dados indexados criados por um algoritmo mapeador de distâncias, desde que este mapeamento habite um dos espaços euclidianos visíveis.

Um objeto é representado por um ponto no espaço, uma região coberta por um par objeto/raio pode ser representada por uma esfera com o objeto no centro, as ligações podem ser representadas por retas ligando os objetos e os resultados podem ser representados pelas regiões e objetos, enquanto os níveis podem ser representados por cores. A Figura 5.2 ilustra alguns exemplos deste modo de representação em um espaço de duas dimensões. Outros recursos como o uso de ícones para representar os diversos tipos de objetos também podem ser empregados.

Representações deste tipo são mais facilmente entendidas pelos usuários, sendo capazes de apresentar para este, uma representação visual de uma estrutura métrica em um dado momento. A execução de um procedimento pode ser apresentada por meio de seqüências de representações visuais das várias configurações da estrutura ao longo deste procedimento. Observando as diferenças em cada uma destas configurações, é possível inferir quais ações cada algoritmo está executando e também determinar as conseqüências destas ações.

5.4.1 Gerenciando o Excesso de Informações

Uma das grandes questões quando se trabalha com ferramentas de visualização reside na quantidade de informações que devem ser exibidas para o usuário. Além de fatores

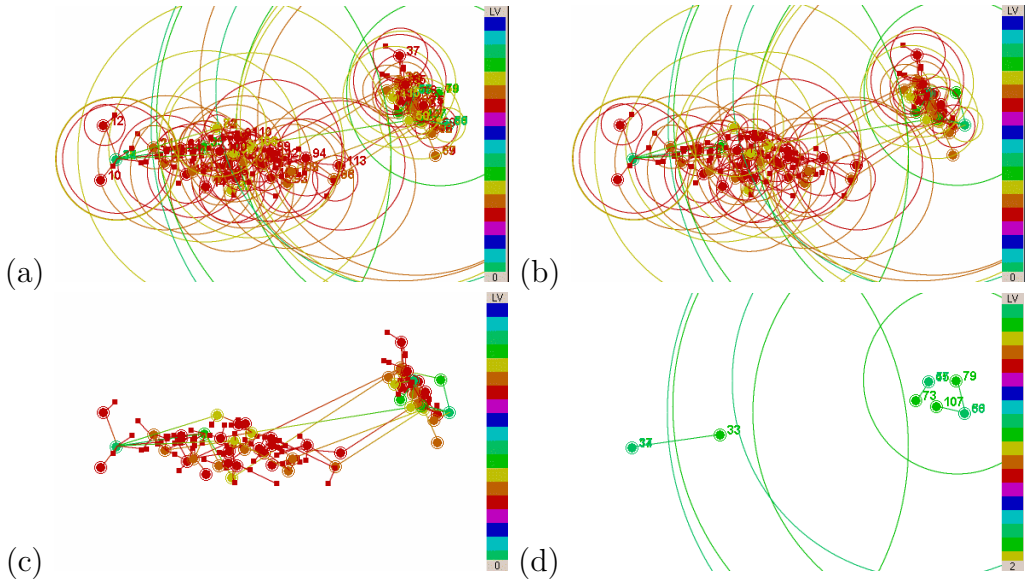

Figura 5.3: Uso de filtragem na apresentação de um MAM. Estas figuras mostram um "dump" de uma *Slim-Tree* carregada com o conjunto IRIS, tendo como função de distância a distância euclidiana e utilizando uma página de disco de 160 bytes. (a) "dump" completo, com todos os elementos. (b) Todos os elementos sem as etiquetas de texto. (c) Todos os elementos, sem o texto e sem os raios de cobertura. (d) Apenas os níveis 2 e 3 da estrutura com todos os elementos.

cognitivos relacionados a este excesso, limitações dos dispositivos de exibição de imagens existentes reduzem consideravelmente o quantidade de informações que podem ser exibidas para o usuário.

Em aplicações onde os dados podem ser classificados por algum critério, como é o caso da visualização de MAM, soluções simples mas bastante satisfatórias, como a aplicação de técnicas de filtragem dos elementos que compõem a representação visual são utilizadas. Ou seja, elimina-se da visualização elementos segundo uma série de critérios baseados em suas propriedades. Esta filtragem, no caso dos MAM, pode basear-se na classificação de elementos estruturais de acordo com seu tipo (nós, objetos, ligações, raios entre outros) ou nível ocupado na estrutura. A Figura 5.3 apresenta alguns exemplos de aplicação esta técnica.

Se o objeto de estudo consiste na visualização da execução de um procedimento, pode-se utilizar fatores temporais para filtrar ainda mais cada cena. Os elementos envolvidos no procedimento podem ser eliminados ou enfatizados de acordo com sua participação no procedimento, ou seja, os elementos "visitados" mais recentemente podem ser enfatizados em detrimento dos demais (Figura 5.4). Este recurso de visualização foi batizado de "Tail" (cauda) por se assemelhar à cauda de uma cobra enquanto esta se desloca em uma superfície qualquer.

Outra estratégia que também pode ser aplicada para eliminar o excesso de informações consiste em exibir apenas os elementos da estrutura potencialmente influentes ao

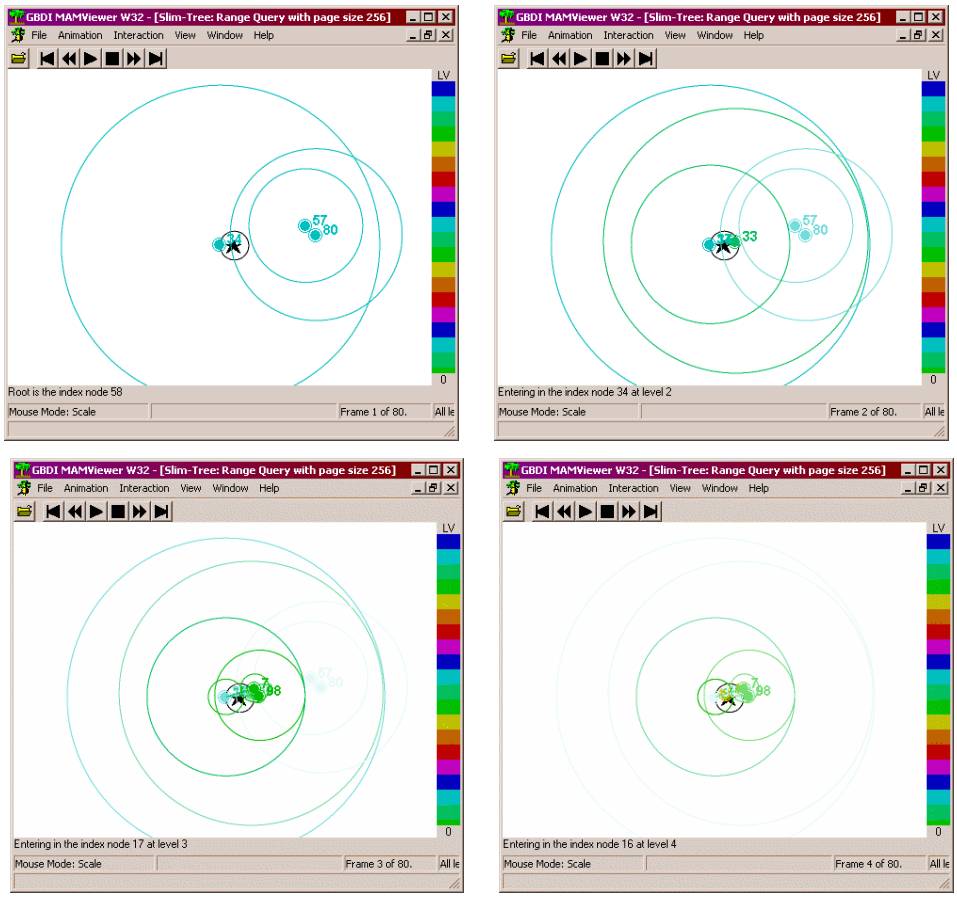

Figura 5.4: Amostra do efeito do "Tail" de comprimento 2 sobre uma animação de uma busca. Os elementos visitados mais recentemente são enfatizados enquanto os demais elementos vão se "apagando" conforme deixam de ser relevantes para a operação.

procedimento observado, ou seja, adicionar à representação visual apenas a vizinhança dos elementos de interesse do procedimento e não todos os elementos indiscriminadamente. Esta estratégia, além de minimizar os elementos visíveis ao mínimo necessário, simplifica consideravelmente a extração de informações da estrutura durante o procedimento.

5.4.2 Efeitos da Precisão dos Algoritmos Mapeadores de Distâncias

Os algoritmos mapeadores de distância descritos no Capítulo 4 que podem ser utilizados para visualizar dados métricos geram distorções que, em alguns casos, podem comprometer o estudo destes conjuntos de dados. Ferramentas como o *FastMapDB* [Traina Jr. et al., 2000b, Traina et al., 2001b, Traina et al., 2001a], destinadas ao estudo de conjuntos de dados, são bastante afetadas por este problema. Porém, tais distorções quantificadas (através da função de *stress*)e quando extrapolarem um determinado limiar de aceitação, têm a visualização produzida descartada.

Nas aplicações de visualização de MAM, porém, os problemas causados por distor-

ções muito grandes nos mapeamentos de certos conjuntos de dados são consideravelmente menores. Isto ocorre porque, diferentemente de aplicações de análise dos conjuntos de dados, as aplicações de visualização de MAM visam explorar as estruturas e suas organizações perante conjuntos de dados, e não os próprios dados.

Com exceção da quantidade de bytes necessária para armazená-los, a natureza dos objetos indexados tem pouco efeito direto sobre a organização dos MAM. O fator que mais influencia o comportamento destas estruturas é a distribuição das distâncias entre os elementos do conjunto de dados, ou seja, os valores das distâncias e os agrupamentos existentes entre estes elementos. Isto permite que os conjuntos de dados sejam substituídos por outros com distribuições de distância similares mas com mapeamentos de melhor qualidade.

5.5 Simulando um MAM

Baseando-se no modelo de organização e no modelo de comportamento de um MAM genérico, é possível criar um simulador de estruturas capaz de reproduzir as operações realizadas por uma estrutura e gerar as representações visuais que serão exibidas para um usuário.

Este simulador utiliza informações sobre o funcionamento das estruturas, extraídas diretamente destas, para determinar quais operações associadas ao comportamento genérico das estruturas (descritas na Seção 5.3.2) devem ser realizadas em seu modelo de organização interno, reproduzindo em detalhes todas as transformações sofridas pela estrutura.

Além disto, a simulação deve ser controlada de modo a criar, em momentos determinados pela própria estrutura, representações visuais da organização da estrutura. Estas representações, denominadas de *frames* (quadros de animação), são adicionadas a uma seqüência de representações que irá constituir uma animação referente ao procedimento estudado.

O simulador proposto é composto por duas partes distintas, uma interna às estruturas, denominado de extrator e o simulador em si. A função do simulador é reproduzir o comportamento dos MAM de acordo com as informações fornecidas pelo extrator. Seu funcionamento baseia-se no modelo genérico de MAM descrito na Seção 5.3.1.

O extrator é um dos componentes mais importantes deste sistema, sendo o responsável por fornecer todas as informações necessárias para que o simulador possa reproduzir fielmente a operação da estrutura. Este componente deve ser instalado nas estruturas de modo que possa capturar as transformações nelas ocorridas. A correta extração de informações das estruturas é fundamental para o funcionamento do simulador, e portanto, de todo o sistema de visualização.

5.5.1 Extração de Informações da Estrutura

As informações extraídas da estrutura constituem um dos elos fundamentais da simulação de MAM. Informações extraídas incorretamente levarão a simulações inconsistentes e, conseqüentemente, a diagnósticos errados sobre o comportamento das estruturas. Portanto, é essencial que as chances de ocorrência de erros na obtenção destas informações seja mínima.

Deste modo, optou-se por limitar a quantidade de informações extraídas para o mínimo possível, simplificando consideravelmente o processo de obtenção destes dados, reduzindo consideravelmente a probabilidade de ocorrência de erros, permitindo ainda que estas informações sejam extraídas facilmente, mesmo de MAM pouco conhecidos.

Estas informações consistem de eventos simples que são apenas declarações da existência de elementos da estrutura (nós, objetos e respostas), alterações de suas propriedades, eliminação de elementos e algumas transformações globais que afetam toda a estrutura. Estes eventos podem ser facilmente identificados e localizados dentro de um procedimento, pois nenhum conhecimento sobre as conseqüências destes eventos é necessário.

Além das informações estruturais, a simulação pode incluir um elemento temporal, visto que os algoritmos sempre possuem um certo conjunto de elementos de interesse a cada passo. Esta informação pode ser usada para criar um rastro do caminho percorrido pelo algoritmo, enfatizando as áreas de interesse mais recentemente visitadas em detrimento das demais.

Os eventos extraídos devem ainda ser agrupados em *frames* (quadros) para que o simulador possa determinar quando uma representação visual da estrutura deve ser gerada e adicionada à animação que mostra o comportamento do procedimento. Em outras palavras, cada agrupamento de eventos resulta em um quadro na animação do procedimento.

O modo como estes eventos podem ser agrupados é dependente não só do comportamento da estrutura em questão, mas também da necessidade de detalhes que se espera obter na simulação. O modo com que os eventos são agrupados não afeta a simulação, mas define o nível de detalhes das animações que serão exibidas para os usuários.

5.6 Arquitetura Básica de uma Aplicação de Visualização de MAM

Utilizando os conceitos definidos neste capítulo, é possível definir uma arquitetura básica na qual qualquer aplicação de visualização de estruturas métricas deve possuir. A Figura 5.5 ilustra os componentes básicos desta arquitetura.

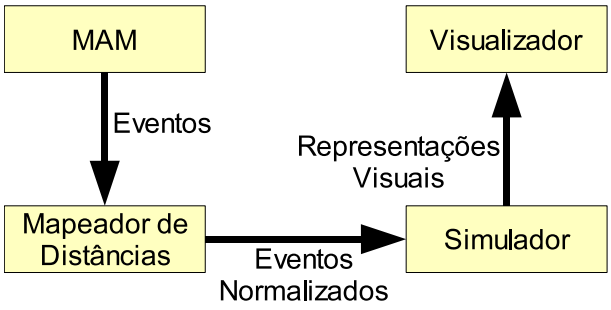

Figura 5.5: Arquitetura básica de uma aplicação de visualização de MAM.

No início do processo, encontra-se a própria estrutura métrica, da qual as informações sobre o funcionamento serão extraídas durante a execução de algum procedimento. Estas informações são, na realidade, eventos que descrevem as transformações sofridas pela estrutura durante uma determinada operação e podem conter informações sobre os objetos indexados pela estrutura. Além disto, estes eventos devem estar agrupados em *frames*, para definir estados relevantes que devem ser exibidos para o usuário.

Os eventos recém extraídos contêm os objetos ainda no espaço métrico original. Como estes objetos no seu estado original precisam ser convertidos para um espaço euclidiano de uma, duas ou três dimensões em algum ponto do processo, optou-se por fazer esta conversão logo após a extração dos eventos (uso do Mapeador de Distâncias). Isto elimina a necessidade de tratar objetos cujo tipo de dados é desconhecido dentro do simulador. Dá-se a estes eventos a denominação de **Eventos Normalizados**.

Os eventos normalizados são encaminhados para o simulador, onde são utilizados para reproduzir os estados do sistema e gerar instantâneos ("snapshots") de seu estado ao fim da execução de cada agrupamento de eventos. Estes instantâneos manifestam-se na forma de modelos geométricos da estrutura, que são agrupados em seqüência para formar animações que constituem as representações visuais do procedimento estudado.

Estas representações visuais são enviadas para o visualizador que permite que o usuário explore o procedimento através da manipulação das representações visuais geradas pelo simulador.

Esta arquitetura permite que o ônus da visualização da estrutura seja minimizado em seu ponto mais crítico que é a extração de informações da estrutura, repassando grande parte da complexidade do sistema para o simulador de estrutura. Uma outra vantagem associada a este modelo é a possibilidade de visualizar estruturas desconhecidas com bastante facilidade pois, estas não precisam detalhar suas ações, apenas as mudanças ocorridas em sua estrutura, tarefa muito mais simples.

Por outro lado, esta organização não permite o fluxo reverso de informações do visualizador para a estrutura, ou seja, não é possível interferir na execução de um procedimento enquanto este é explorado visualmente. Porém, esta limitação não constitui

uma grande desvantagem para o sistema pois tais interferências seriam de pouca valia no estudo dos procedimentos e das propriedades da estrutura.

5.7 Conclusão

A criação de uma aplicação de visualização de estruturas, sejam elas métricas ou não, dependem de fatores como a extração de informações sobre o seu funcionamento e formas para criar representações visuais destas que sintetizem suas propriedades mais relevantes e possam ser entendidas mais facilmente pelos usuários.

A criação de uma aplicação desta natureza envolve a definição de modelos de organização e comportamento de MAM que sejam capazes de reproduzir todas as operações realizadas por uma estrutura. Estes modelos são utilizados para criar um simulador de estruturas que pode ser utilizado para a geração de representações visuais destas. Este simulador utiliza informações extraídas diretamente das estruturas, sendo capaz de reproduzir fielmente o seu comportamento.

A exibição da organização e do comportamento de estruturas métricas para um usuário em dispositivos gráficos envolve a criação de representações visuais dos dados indexados por estas estruturas. Estas representações podem ser obtidas pelo uso de algoritmos mapeadores de distâncias, como o *FastMap*. As demais informações sobre a organização da estrutura são criadas a partir da representação dos dados.

Estas representações gráficas das estruturas são capazes apenas de exibir o estado dos elementos da estrutura em um dado momento. Por isto, seqüências de estados da estrutura no decorrer de um procedimento são agrupados em animações que são exibidas para os usuários, ilustrando seu comportamento.

Um problema inerente a estes modelos visuais é o excesso de informações que dificulta consideravelmente o entendimento das informações exibidas para o usuário. Os efeitos deste problema podem ser bastante reduzidos pelo uso de técnicas de filtragem que podem ser configuradas para inibir a exibição de determinadas informações presentes nas representações visuais, utilizando critérios como a classificação dos elementos e informações temporais (no caso de animações de procedimentos).

A qualidade dos mapeamentos gerados pelos algoritmos mapeadores de distância são bastante relevantes para a qualidade dos modelos visuais gerados pelo visualizador, porém, seus efeitos podem ser contornados com a troca dos conjuntos de dados estudados. Se um conjunto gera mapeamentos ruins, pode-se tentar trocá-lo por outro que resulte em melhores mapeamentos, mas cujas distribuições de distâncias entre seus elementos seja similar ao anterior, pois o comportamento dos MAM depende mais destes fatores do que da natureza do conjunto de dados em si.

Baseando-se nos conceitos descritos neste capítulo é possível criar uma arquitetura

básica que descreve todos os componentes necessários para a construção de uma aplicação de visualização de MAM. Esta arquitetura serve de base para a criação do sistema *MAMView*, descrito em mais detalhes no Capítulo 6, que foi implementado para avaliar a viabilidade de sistemas de visualização de MAM.

Capítulo

6

O Sistema MAMView

6.1 Introdução

Baseando-se nos conceitos apresentados no Capítulo 5, foi implementado um sistema protótipo, batizado de *MAMView*, cujo propósito é avaliar a viabilidade prática de um sistema de visualização de MAM e prover aos desenvolvedores de MAM, uma poderosa ferramenta de análise para suas estruturas. Este protótipo implementa toda a arquitetura básica descrita na Seção 5.6.

6.2 Arquitetura do Sistema MAMView

O sistema *MAMView* está dividido em dois módulos principais, o *MAMView Extractor* e o *MAMViewer*, cada um responsável por parte das funcionalidades descritas na Seção 5.6. A arquitetura do sistema *MAMView* é ilustrada pela Figura 6.1.

O *MAMView Extractor* é o módulo responsável por extrair os eventos das estruturas, traduzí-los para a forma normalizada e criar os arquivos de dados que serão utilizados pelo *MAMViewer*. Este módulo está implementado diretamente na biblioteca de MAM *GBDI Arboretum* [GBDI-ICMC-USP, 2004], sendo utilizado como plataforma central para o desenvolvimento das estruturas métricas em construção no GBDI.

O *MAMViewer* é o módulo de simulação e visualização do sistema *MAMView*. Ele é responsável por recriar os passos executados pelo MAM (baseando-se nos eventos extraídos pelo *MAMView Extractor*), gerar as representações visuais dos estados da estrutura

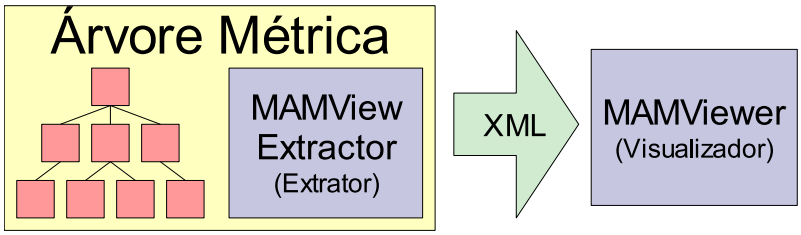

Figura 6.1: Arquitetura do sistema *MAMView*.

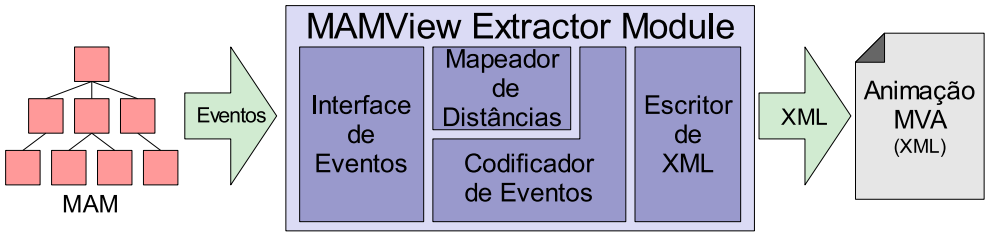

Figura 6.2: Arquitetura interna do *MAMView Extractor*.

durante um procedimento e permitir que os usuários possam explorar estas representações visuais interativamente.

A comunicação entre os dois módulos é feita através de um arquivo de dados, denominado de *MAMView Animation* ou apenas MVA. Este arquivo contém os eventos extraídos pelo *MAMView Extractor* que descrevem o comportamento de uma estrutura.

Os detalhes de cada um destes componentes estão detalhados nas Seções 6.3, 6.4 e 6.5.

6.3 O Módulo de Extração MAMView Extractor

O **MAMView Extractor** é o módulo responsável por extrair os eventos das estruturas, convertê-lo para a forma normalizada, aceita pelo *MAMViewer*, e gerar os arquivos de dados necessários para criar as animações dos procedimentos nas estruturas.

Como este módulo deve extrair as informações sobre as mudanças de estado diretamente do MAM estudado, a implementação do *MAMView Extractor* está integrada à biblioteca de MAM *GBDI Arboretum*, podendo ser utilizado por todas as estruturas implementadas nesta biblioteca.

Como todos os componentes da biblioteca *GBDI Arboretum*, a implementação do *MAMView Extractor* é portável entre vários sistemas operacionais.

6.3.1 Arquitetura Interna do MAMView Extractor

O *MAMView Extractor* implementado por este protótipo está dividido em quatro componentes distintos, a *Interface de Eventos*, o *Mapeador de Distâncias*, o *Codificador de*

Eventos e o *Escritor de XML*. A arquitetura interna do *MAMView Extractor* é ilustrada na Figura 6.2.

O primeiro componente, chamado de **Interface de Eventos** é responsável por fazer a interface externa entre a estrutura e o *MAMView Extractor*, ou seja, recebe os eventos do MAM e os repassa para o *Mapeador de Distâncias* (quando os eventos envolvem objetos) ou diretamente para o *Codificador de Eventos*. Este componente é responsável também pela detecção de possíveis inconsistências na declaração dos eventos decorrentes do uso incorreto do *MAMView Extractor*.

A Interface de Eventos manifesta-se na forma de uma interface de uma classe C++, cuja filosofia de funcionamento se assemelha bastante a outros componentes presentes na biblioteca *GBDI Arboretum*, tornando seu uso bastante familiar.

O **Mapeador de Distâncias** é responsável por traduzir os eventos que envolvam objetos do domínio métrico da estrutura para sua forma canônica, ou seja, para sua imagem no espaço euclidiano com três dimensões. Este componente implementa um algoritmo mapeador de distâncias, no caso, o *FastMap* (ver o Capítulo 4). Os eventos, depois de traduzidos, são repassados para o *Codificador de Eventos*.

O **Codificador de Eventos** é o componente responsável por organizar os eventos de modo que possam ser representados no formado de saída MVA. Este componente utiliza o **Escritor de XML** que se encarrega de criar os arquivos MVA (um XML). Estes arquivos podem então ser utilizados pelo *MAMViewer* para reproduzir os eventos.

6.3.2 Usuários do MAMView Extractor

O *MAMView Extractor* tem como usuários, os desenvolvedores de MAM, sendo visto por estes como um componente que deve ser inicializado e depois utilizado para notificar as mudanças ocorridas nas estruturas. Para estes desenvolvedores, basta saber quais informações devem ser extraídas da estrutura e como agrupá-las para gerar a animação desejada. Não é necessário conhecer nenhum detalhe sobre o funcionamento do sistema *MAMView*.

Os desenvolvedores de estruturas vêem este módulo como um componente único, que deve ser instalado na estrutura para guardar todos os eventos relevantes para a execução de um procedimento. Todo o processo de tradução dos eventos para a forma normalizada e a criação dos arquivos MVA são executados automaticamente, sem a necessidade de interferência dos desenvolvedores.

Além disto, o *MAMView Extractor* procura detectar e alertar os desenvolvedores de MAM sobre possíveis inconsistências ocorridas na instalação do extrator na estrutura, provendo, quando possível, instruções para solucionar os problemas, tornando sua utilização bastante simples.

6.3.3 Eventos Extraídos Pelo MAMView Extractor

O *MAMView Extractor* permite que todos os eventos descritos na Seção 5.5, sejam extraídos. Em outras palavras, é possível declarar e eliminar nós, objetos e respostas, declarar transformações globais (como o aumento ou a diminuição da altura) e enfatizar elementos visitados recentemente pelos algoritmos. Além dos eventos descritos na Seção 5.3.2, o *MAMView Extractor* permite aos desenvolvedores associar comentários à animação e aos *frames*.

É importante lembrar que estes eventos tem efeito acumulativo durante a simulação, ou seja, uma vez que um evento seja declarado, suas conseqüências perdurarão até que um novo evento altere este estado. Por exemplo, se um nó for declarado estando associado a 10 objetos, o estado deste nó só será modificado se algum destes objetos for eliminado ou o nó for declarado novamente com outras características.

Mais detalhes sobre estes eventos podem ser encontrados no Apêndice A.

6.3.4 Instalação do MAMView Extactor em um MAM

A instalação do *MAMView Extractor* em uma estrutura métrica é relativamente simples. O desenvolvedor do MAM deve, primeiramente, criar alguns procedimentos que são necessários para a inicialização do módulo de extração. Estes procedimentos devem ser capazes de coletar um certo número de objetos armazenados na estrutura (utilizados para inicializar o *FastMap*) e configurar o extrator (local dos arquivos de saída entre outras coisas).

Uma vez inicializado, o extrator estará pronto para receber as notificações dos eventos ocorridos na estrutura e gerar os arquivos de saída (arquivos MVA). Para tanto, basta que o extrator esteja devidamente instalado nos procedimentos que se deseja visualizar, como consultas, inserções, remoções e outras operações sobre a estrutura.

A notificação dos eventos para o *MAMView Extractor* é feita por meio de uma *API* (*Application Programming Interface*) que utiliza como parâmetros de entrada os próprios objetos e componentes da implementação da estrutura, não sendo necessário conhecer os detalhes de funcionamento do sistema *MAMView*. Esta interface é descrita com mais detalhes no Apêndice B.

Em linhas gerais, a instalação do extrator em um procedimento resume-se a notificar o início e o fim de uma animação (início e fim do procedimento) e a definição de pelo menos um agrupamento de eventos (*frame*) dentro desta animação. Para cada animação declarada, o *MAMView Extrator* criará um arquivo MVA automaticamente, sem a necessidade de intervenção do desenvolvedor.

Cada *Frame* pode conter um ou mais eventos que descrevem as ações dos algoritmos na estrutura. Estes eventos consistem em declarações de existência, alteração ou

eliminação de elementos da estrutura (a existência ou atualização de propriedades de um nó ou objeto), declarações de mudanças de estado globais da estrutura (aumento ou diminuição de altura entre outras operações) e a notificação da passagem do algoritmo por um determinado elemento (marcação dos elementos de interesse do algoritmo).

Os eventos são sempre reportados de forma declarativa, não sendo possível controlar o modo que o simulador deve agir. Além disto, alguns dos eventos permitem a inclusão de comentários, que poderão ser utilizados pelo usuário como fonte de informação adicional.

Como descrito no Capítulo 5, estes eventos devem ser agrupados em *frames* para definir quais transformações devem ser aplicadas na estrutura antes que uma nova representação de seu estado seja criada. A organização destes eventos em quadros é dependente do procedimento estudado e da necessidade de detalhes exigida pelo desenvolvedor de MAM. É importante salientar que o modo como os eventos são organizados em *frames* define como o procedimento será apresentado, mas não interfere na simulação final das estruturas.

Para visualizar uma busca, por exemplo, os eventos podem ser agrupados em *frames* da seguinte maneira:

- No início do procedimento declarar o início de uma animação (opcional pois uma animação pode conter mais de um procedimento);
- declarar um *frame* contendo a especificação da busca (seu estado inicial);
- ao entrar em uma interação recursiva, definir um *frame* contendo a declaração de todos os elementos diretamente relacionados, ou seja, todos os elementos que podem ser declarados diretamente;
- ao retornar de uma iteração, definir um *frame* que marca o nó atual como "ativo";
- a cada alteração ou grupo de alterações no resultado, definir um "frame" para atualizar o resultado;
- notificar o final da animação (opcional).

Estas ações, quando aplicadas, não irão declarar ao simulador a existência de elementos não visitados. Porém, alterando-se a execução do procedimento para gerar um "frame" contendo todo o "dump" da estrutura antes de iniciar a busca garantirá que todos os elementos da estrutura sejam exibidos. Entretanto, esta é uma atitude pouco recomendada, pois gera um excesso de informações que pode prejudicar o entendimento da animação.

O "dump" de toda a estrutura pode ser obtido através da instalação de declarações dos componentes visitados por um algoritmo que percorra toda a árvore recursivamente.

(a)

```
// Percorre a estrutura intera.
procedimento Dump()

   DumpRecursive(root)
fim Dump




// Percorre a estrutura recursivamente
// em Pre-Ordem.
procedimento DumpRecursive(root)

   node = GetNode(root)
   para cada entry de node faça
      se entry é referência
         // Entrada de subárvore
         DumpRecursive(entry.root)
      senão
         // É um objeto. Faça algo !
         ...
      fim se
   fim para
fim DumpRecursive
```

(b)

```
// Percorre a estrutura intera.
procedimento Dump()

   // Inicialização da animação
   MAMView.BeginAnimation()
   MAMView.SetLevel(0)

   MAMView.BeginFrame() // Início do Frame
   DumpRecursive(root)
   MAMView.EndFrame() // Fim do Frame

   MAMView.EndAnimation() // Fim da animação
fim Dump

// Percorre a estrutura recursivamente
// em Pre-Ordem.
procedimento DumpRecursive(root)

   MAMView.LevelUp() // Próximo nível
   node = GetNode(root)
   MAMView.SetNode(node) // Declarar dados do nó
   para cada entry de node faça
      se entry é referência
         // Entrada de subárvore
         DumpRecursive(entry.root)
      senão
         // É um objeto. Faça algo !
         ...
         // Declarar dados do nó
         MAMView.SetObject(entry.object)
      fim se
   fim para
   MAMView.LevelDown() // Voltando um nível
fim DumpRecursive
```

Figura 6.3: Pseudo-código da criação de um "dump" de uma estrutura fictícia. O código (a) ilustra um algoritmo para percorrer a árvore recursivamente enquanto o código (b) apresenta, em vermelho, a instalação do *MAMView Extractor* para gerar o arquivo MVA do "dump". A interface do extrator foi simplificada para facilitar a leitura do código.

Estas declarações podem estar todas agrupadas em um único "frame" ou então acompanhar a execução do procedimento recursivo, tendo como critério de agrupamento, os elementos encontrados pela primeira vez em cada interação da recursão. A Figura 6.3 mostra o pseudo-código utilizado para criar um "dump" de uma estrutura genérica.

Procedimentos mais complexos, como adições e remoções, também podem ser visualizados pelo *MAMView*, bastando para isto, instalar o extrator nestes procedimentos, notificando para o extrator todas as transformações ocorridas na estrutura em cada "passo" dos algoritmos. A divisão dos eventos em "passos" é determinada pelo agrupamento dos eventos em *Frames*. A Figura 6.4 exemplifica a instalação do extrator em um procedimento de consulta por abrangência de uma árvore genérica.

O nível de detalhes obtido com os dados extraídos pelo *MAMView Extractor* é dependente do modo como os eventos foram agrupados durante o processo de extração. Como a divisão dos eventos em *Frames* não altera o resultado final da simulação, é possível refinar experimentalmente os agrupamentos de eventos, ajustando-os às necessidades dos usuários do visualizador.

(a)

```
// Busca por abrangência.
função RangeQuery(sample, radius)

   result = CreateResult()
   RangeQueryR(root, sample, radius,
      result)
   retornar result
fim RangeQuery

// Busca por abrangência recursiva
procedimento RangeQueryR(root, sample,
      radius, result)

   node = GetNode(root)
   para cada entry de node faça
      se entry é referência
         se entry.rep se qualifica
            RangeQueryR(entry.root,
               sample, radius, result)
         fim se
      senão
         // É um objeto.
         se entry faz parte da resposta
            result.add(entry.object)
         fim se
      fim se
   fim para
fim RangeQueryR
```

(b)

```
// Busca por abrangência.
função RangeQuery(sample, radius)

   MAMView.BeginAnimation() // Início da
                            // animação
   result = CreateResult()
   RangeQueryR(root, sample, radius,
      result)

   // Resultado final
   MAMView.BeginFrame()
   MAMView.SetResult(result)
   MAMView.EndFrame()

   MAMView.EndAnimation() // Fim da animação
   retornar result
fim RangeQuery

// Busca por abrangência recursiva
procedimento RangeQueryR(root, sample,
      radius, result)

   MAMView.LevelUp() // Próximo nível
   node = GetNode(root)
   // Entrada no nó
   MAMView.BeginFrame()
   MAMView.SetNode(node)
   MAMView.EndFrame()
   para cada entry de node faça
      se entry é referência
         se entry.rep se qualifica
            RangeQueryR(entry.root,
               sample, radius, result)
            // Voltando para o nó
            MAMView.BeginFrame()
            MAMView.ActivateNode(node)
            MAMView.EndFrame()
         fim se
      senão
         // Declarar o objeto
         MAMView.BeginFrame()
         MAMView.SetObject(entry.object)
         MAMView.EndFrame()
         se entry faz parte da resposta
            result.add(entry.object)
            // Atualizar a resposta
            MAMView.BeginFrame()
            MAMView.SetResult(result)
            MAMView.EndFrame()
         fim se
      fim se
   fim para
   MAMView.LevelUp() // Próximo nível
fim RangeQueryR
```

Figura 6.4: Pseudo-código de uma busca por abrangência em uma estrutura fictícia. O código (a) ilustra um algoritmo original enquanto o código (b) apresenta, em vermelho, a instalação do *MAMView Extractor* para gerar o arquivo MVA relativo a cada execução do procedimento. A divisão dos eventos em *Frames* é bastante simples gerando quadros a medida que as alterações ocorrem. A interface do extrator foi simplificada para facilitar a leitura do código.

Em linhas gerais, a instalação do extrator nas estruturas é bastante simples, mesmo que o comportamento destas seja pouco conhecido, pois este processo resume-se a localizar os pontos do algoritmo onde ocorrem modificações na estrutura e reportá-los exatamente como ocorrem, sem a necessidade de prever as conseqüências destas alterações na estrutura.

6.3.5 Localização do Mapeador de Distâncias

O único fator realmente relevante na implementação do *MAMView Extractor* reside na localização do *Mapeador de Distâncias* pois a posição deste componente no sistema *MAMView* é bastante flexível. Em teoria, este componente poderia estar implementado em qualquer ponto entre a *Interface de Eventos* do *MAMView Extractor* e o sub-sistema de visualização do *MAMViewer*. Porém, sua colocação dentro do *MAMView Extractor* traz grandes vantagens para o sistema como um todo.

A primeira vantagem reside no aproveitamento da implementação do objeto indexado e da função de distância presentes na estrutura. Se este componente não compartilhasse estas implementações, o *MAMViewer* seria obrigado a implementar todos os objetos e todas as funções de distância possíveis, o que tornaria sua implementação bastante complexa, além de abrir margem para inconsistências entre as implementações presentes na estrutura e no visualizador.

Outra vantagem associada a esta decisão de projeto reside na forma de representação dos objetos associados aos eventos nos arquivos MVA. A enorme variedade de objetos que podem ser indexados por um MAM tornaria a definição do XML consideravelmente mais complexa e limitada pois apenas objetos previstos de antemão poderiam ser armazenados.

Há também vantagens na implementação do simulador, que pode ter sua implementação simplificada, pois pode trabalhar com uma representação unificada dos objetos (apenas objetos ocupando espaços euclidianos com 3 dimensões).

6.4 O Módulo de Visualização MAMViewer

O *MAMViewer* implementa o módulo de simulação e visualização das estruturas. Este módulo utiliza as informações extraídas da estrutura durante um determinado procedimento, recria o comportamento da estrutura por meio de simulações e cria as representações visuais que podem ser exploradas interativamente pelo usuário.

Este módulo manifesta-se na forma de uma aplicação externa à biblioteca *GBDI Arboretum*, desenvolvida para a plataforma Microsoft Win32. Embora quase a totalidade de seus componentes funcionais seja portável entre vários sistemas operacionais, os componentes relacionados à interação com o usuário e de interface com dispositivos gráficos

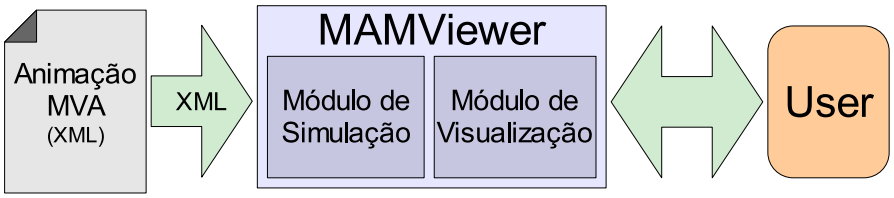

Figura 6.5: Arquitetura interna do *MAMViewer*.

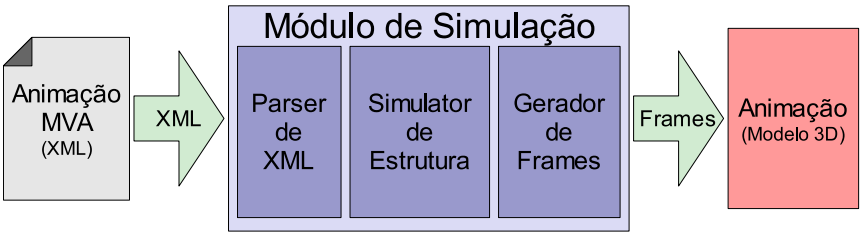

Figura 6.6: Arquitetura interna do Sistema de Simulação do *MAMViewer*.

tornam a implementação deste módulo bastante dependente de sistema, independentemente de qual ele seja.

6.4.1 Arquitetura do MAMViewer

O *MAMViewer* está dividido em 2 subsistemas, um responsável pela simulação da estrutura (Subsistema de Simulação) e o outro pela visualização das representações gráficas das estruturas (Subsistema de Visualização). Cada um destes subsistemas será melhor descrito nas próximas subseções.

6.4.1.1 Subsistema de Simulação

O **Subsistema de Simulação** é responsável por interpretar as informações extraídas pelo *MAMView Extractor*, recriar o comportamento das estruturas e gerar as representações visuais que serão repassadas para o *Subsistema de Visualização*.

Este subsistema está dividido em três componentes, o *Parser de XML*, o *Simulador de Estruturas* e o *Gerador de Frames*. A organização destes componentes dentro do *Subsistema de Simulação* é ilustrado pela Figura 6.6.

O **Parser de XML** é o componente responsável por interpretar os arquivos de dados (MVA) e comandar as ações do *Simulador de Estruturas* e do *Gerador de Frames*, de acordo com os eventos descritos nestes arquivos. A validação do formato do arquivo também é realizada por este componente.

O **Simulador de Estruturas** implementa o modelo genérico de organização e comportamento descritos na Seção 5.3. É o responsável por reproduzir o comportamento das estruturas de acordo com os eventos extraídos do MAM estudado. Este simulador

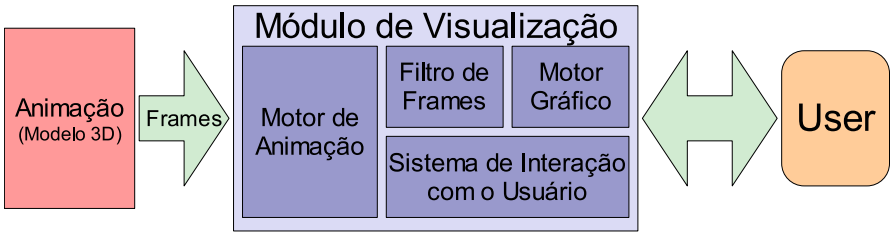

Figura 6.7: Arquitetura interna do Sistema de Visualização do *MAMViewer*.

trabalha apenas com eventos normalizados e não aplica nenhum tipo de restrição às operações dos MAM descritas pelas informações deles extraídas.

Finalmente, o **Gerador de Frames** é responsável por criar as representações visuais da estrutura, baseando-se no estado do simulador em momentos determinados pelas informações extraídas da estrutura.

6.4.1.2 Subsistema de Visualização

O **Subsistema de Visualização** é responsável por exibir os modelos gerados pelo Subsistema de Simulação ao usuário. Todos os recursos que permitem a exploração interativa das animações pelo usuário são implementadas por este subsistema.

Sua arquitetura interna divide-se em quatro componentes principais, o *Motor de Animação*, o *Filtro de Frames*, o *Motor Gráfico* e o *Sistema de Interação com o Usuário*. A organização destes módulos dentro do Subsistema de Visualização é ilustrada na Figura 6.5.

O **Motor de Animação** nada mais é que um sistema de gerenciamento de animações 3D. Ele é responsável por exibir e controlar a ordem de exibição dos quadros(*frames*) que compõem uma animação. O *Motor de Animação* utiliza o **Filtro de Frames**, que implementa as técnicas de visualização utilizadas para minimizar o excesso de informações que podem estar associadas a cada quadro.

Para desenhar cada quadro em sua janela de exibição, o *Motor de Animação* utiliza os serviços do **Motor Gráfico** que é responsável por gerenciar a porta de visão e prover acesso para os dispositivos gráficos, utilizando o *OpenGL*[1] ou outras bibliotecas gráficas. O uso destas bibliotecas gráficas associado a hardwares especiais de aceleração gráfica (GPUs) permitem a visualização de quadros em tempo real, mesmo quando o número de elementos destes ultrapassa a casa das dezenas de milhares.

O controle dos demais módulos é exercido pelo **Sistema de Interação com o Usuário**. Este componente recebe os comandos dos usuários e controla as ações dos demais componentes do módulo de visualização.

[1]Para mais informações, visite http://www.opengl.org/

6.4.1.3 Técnicas de Visualização Utilizadas pelo MAMViewer

Como descrito no Capítulo 5, aplicações de visualização como esta podem gerar modelos visuais cuja quantidade de informações a ser exibida pode ser excessiva. Nestas situações, a habilidade de gerar representações visuais torna-se pouco útil, pois pouca ou nenhuma informação relevante pode ser obtida pela observação destes modelos. É necessário aplicar algumas técnicas especiais de visualização que permitam selecionar as informações que serão exibidas, dependendo das necessidades dos usuários.

A primeira e mais simples das técnicas utilizadas pelo *MAMViewer* consiste na interação direta com os modelos visuais. Os usuários podem modificar as portas de visão interativamente e em tempo real, observando os efeitos destas alterações instantaneamente. Há também a possibilidade de criar múltiplas visões das mesmas animações ou de outras que podem ser exibidas lado a lado, de forma que possam ser comparadas mais facilmente.

Uma outra técnica utilizada é a filtragem de elementos por categoria e nível. Os diversos elementos das estruturas, como nós, objetos, conexões, respostas de buscas e etiquetas podem ser adicionados ou eliminados de qualquer quadro das animações, sem a necessidade de alterá-las definitivamente. Além desta classificação, é possível também filtrar os elementos de acordo com o nível ocupado na estrutura. Ambos os critérios podem ser combinados para aumentar a gama de opções disponíveis aos usuários.

O *MAMViewer* implementa também o princípio da "Tail" (cauda), descrita na Seção 5.4. O comprimento da cauda pode ser modificado a qualquer momento, permitindo que o nível de detalhes exibidos seja controlado conforme a necessidade.

São utilizadas também técnicas associadas à exibição dos modelos tridimensionais como o uso de cores para indicar a direção das ligações entre os níveis (a cor inicial da ligação é interpolada com a cor final para indicar a mudança) e o uso de projeções em perspectiva (pouco relevante) e de nevoeiro (*Fog*) (para aumentar a noção de profundidade entre os elementos da figura). Estas técnicas são capazes de enfatizar certas características representadas pelos modelos tridimensionais exibidos, não sendo utilizadas para gerenciar o excesso de informações destes modelos.

6.5 Comunicação entre os Módulos

A comunicação entre os dois módulos principais do sistema *MAMView* se dá através de um arquivo denominado "MAMView Animation" (MVA). O uso deste formato permite que as implementações do *MAMView Extractor* e do *MAMViewer* possam ser criadas independentemente, desde que ambos possam criar e/ou interpretar os arquivos MVA corretamente.

Entre as vantagens associadas a este modelo de comunicação encontra-se a pos-

```
<?xml version="1.0"?>
<!DOCTYPE mamview SYSTEM "mamview.dtd">
<animation>
   <description>
      <title>
         MVA Example.
      </title>
      <comment>
         This is not a real animation.
      </comment>
   </description>
   <frame id="0">
      <comment>
         Just a node.
      </comment>
      <node id="0" active="true" parent="-1"
         type="0" level="0">
         <repobj x="0" y="0" z="0" radius="1" />
      </node>
    </frame>
</animation>
```

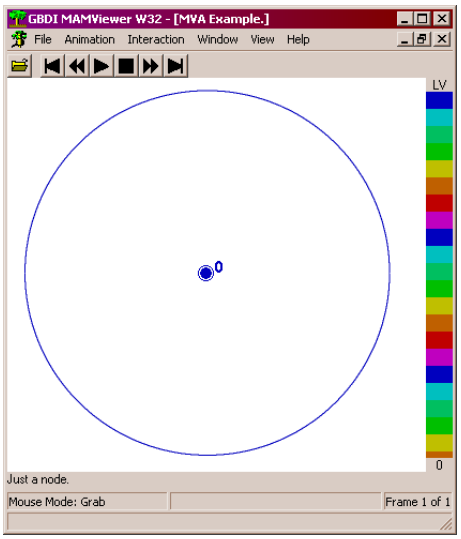

Figura 6.8: Exemplo de um arquivo MVA simples. Este arquivo declara um nó com identificador 0 e um representativo nas coordenadas $(0, 0, 0)$ e raio de cobertura 1.

sibilidade de analisar as animações geradas sem a necessidade de executar novamente o procedimento que a gerou. Ou seja, é possível armazenar estas animações para comparações futuras, sem a necessidade de manter versões executáveis dos procedimentos que as geraram.

Por outro lado, este meio de comunicação não permite que o usuário interfira na execução do procedimento por meio de sua animação. Porém, para implementar este tipo de interação nas estruturas é preciso promover grandes alterações nestas. Isto demanda muito tempo e conhecimento sobre o funcionamento das estruturas, o que nem sempre está disponível.

O formato MVA é, na realidade, um arquivo XML, cujo DTD e interpretação estão descritas com mais detalhes no Apêndice A. A escolha da XML como formato para a troca de informações entre os módulos deve-se, primeiramente, à natureza destes dados. Os eventos e outras informações associadas consistem em construções declarativas que devem estar organizadas em estruturas hierárquicas (os "frames") e manter sua relação de ordem.

Além disto, o XML é um padrão bem definido e bastante difundido. Sua implementação é simples e seu uso é bastante facilitado pela existência de inúmeras ferramentas e bibliotecas de software capazes de manipular arquivos XML.

Outra vantagem associada ao XML está associada à sua forma de apresentação. Embora seja um formato bastante rígido e próprio para a interpretação automatizada, possui uma estrutura que pode ser facilmente compreendida por uma pessoa, sem a necessidade de softwares especiais de visualização. Isto permite que arquivos MVA contendo informações para teste do sistema possam ser compostos facilmente, sem a necessidade de softwares especiais.

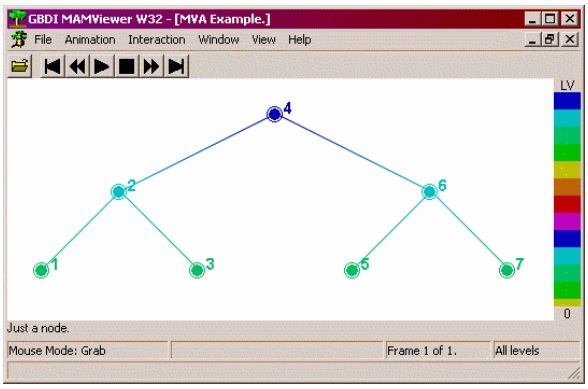

Figura 6.9: Exemplo do uso do *MAMViewer* para visualizar uma árvore binária.

6.6 Outras aplicações do MAMView

Embora o *MAMViewer* tenha sido projetado para representar MAM e seus comportamentos, é possível utilizar o simulador do *MAMViewer* para criar representações visuais de outros tipos de estruturas e dados.

Para tanto, basta conhecer o funcionamento do simulador e a estrutura do arquivo de entrada MVA para criar eventos capazes de "enganar" o simulador, criando uma estrutura métrica falsa, cujo comportamento e representação sejam compatíveis com o resultado desejado.

A Figura 6.9 ilustra a representação de uma árvore binária que foi criada através de um arquivo MVA especialmente modificado. Os nós da árvore binária são representados por nós de uma árvore métrica, que possuem um único representativo com raio zero. Cada um dos representativos foi posicionado de forma a terem suas posições compatíveis com sua localização na árvore binária (as distâncias passam a ser irrelevantes) e tiveram seus valores convertidos para os nomes dos nós. As ligações são representadas da mesma forma que as ligações são representadas nas árvores métricas.

Esta aplicação do *MAMView* não tem conseqüências práticas diretas, porém, serve para reforçar a capacidade de generalização do modelo desenvolvido para representar os MAM apresentados por este trabalho. Outras aplicações do sistema podem ainda ser exploradas, porém, estas aplicações fogem do escopo deste projeto e não serão aqui abordadas.

6.7 Conclusão

Para demonstrar a validade do sistema proposto por este trabalho, foi desenvolvido um protótipo totalmente funcional de um visualizador de MAM, chamado de *MAMView*. Este sistema está implementado junto à biblioteca de MAM *GBDI Arboretum*, o que permite sua aplicação em todas as estruturas implementadas por esta biblioteca.

O *MAMView* está dividido em dois módulos, o *MAMView Extractor*, responsável por extrair as informações das estruturas e o *MAMViewer*, responsável por simular e apresentar as representações visuais dos MAM e de seus procedimentos.

As informações extraídas pelo *MAMView Extractor* são repassadas para o *MAMViewer* pelo uso de um arquivo de dados chamado de MVA. Este arquivo contém todas as informações necessárias para o sistema recriar as operações ocorridas na estrutura e criar as representações visuais destas ações.

A grande capacidade de generalização de estruturas proporcionada pelo modelo genérico de organização e comportamento dos MAM e a arquitetura utilizada para implementar o *MAMView* permitem que estes sejam utilizados para outras atividades não relacionadas com estruturas métricas.

Este protótipo do sistema de visualização de MAM foi utilizado para avaliar sua corretude e seu desempenho em um ambiente real de desenvolvimento de MAM. Alguns dos resultados destes experimentos estão descritos no Capítulo 7.

Capítulo

7

Resultados

7.1 Introdução

Este capítulo apresenta alguns dos resultados obtidos durante os experimentos destinados a validar a viabilidade de uma ferramenta de visualização de MAM, como o sistema *MAMView* proposto por este trabalho.

Os experimentos descritos neste texto visam demonstrar a corretude do sistema com a visualização de um MAM conhecido e já estabelecido (*Slim-Tree*) e os benefícios que podem ser obtidos com o uso deste sistema no processo de desenvolvimento de um novo MAM (*DBM-Tree*).

Foram também realizadas comparações entre os conhecimentos que podem ser obtidos com o uso de um sistema de visualização interativo, como o *MAMView*, e de "dumps" textuais.

7.2 MAMView na Slim-Tree

O objetivo principal dos testes do *MAMView* realizados sobre a *Slim-Tree* é avaliar a corretude do sistema. Para efeitos meramente ilustrativos, esta seção apresenta os resultados da visualização de uma *Slim-Tree* carregada com o conjunto de dados IRIS, utilizando a função de distância euclidiana.

Este conjunto foi escolhido por ser pequeno (composto por apenas 150 objetos), bastante conhecido e ser composto por objetos que, combinados com uma função de dis-

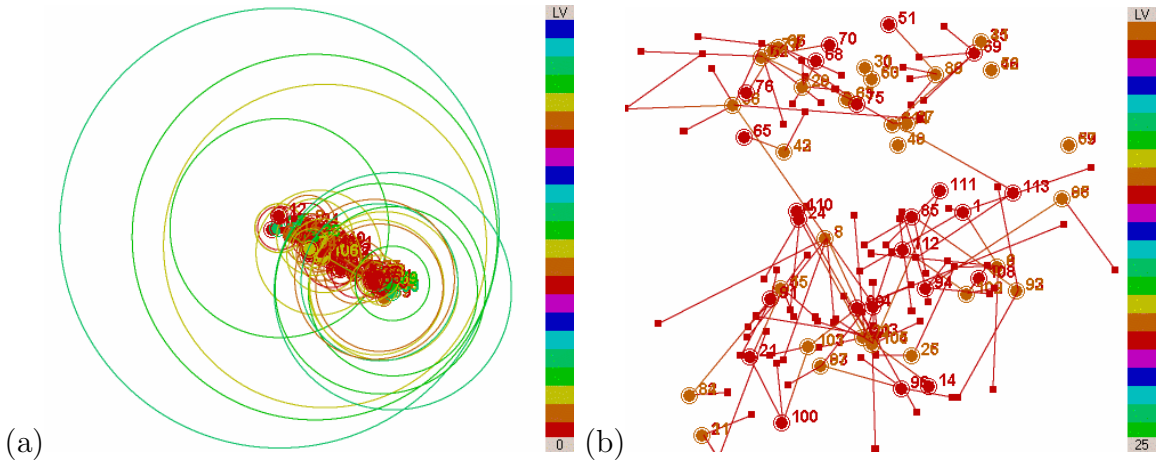

Figura 7.1: Duas imagens geradas pelo *MAMView* apresentando uma *Slim-Tree* com páginas de 256 bytes carregada com o conjunto IRIS (150 objetos com 4 dimensões).

tância, representam um espaço métrico que não pode ser visualizado facilmente. As figuras foram criadas utilizando a implementação da *Slim-Tree* presente na biblioteca *GBDI Arboretum*, utilizando páginas de disco com 256 bytes para simular estruturas com objetos grandes (baixa cardinalidade) e com 1Kb para simular estruturas com cardinalidades mais próximas das normalmente encontradas em aplicações reais.

A Figura 7.1 mostra o "dump" visual da *Slim-Tree* carregada com o conjunto de dados IRIS (páginas de 256 bytes). A Figura 7.1(a) mostra uma visão geral da estrutura, exibindo todos os seus elementos. É possível verificar que a estrutura possui 7 níveis, sendo que os níveis mais próximos da raiz têm raios de cobertura maiores que os níveis mais próximos das folhas. Em 7.1(b) apenas os últimos 2 níveis da estrutura, sem os raios de cobertura, são desenhados.

A Figura 7.2 ilustra os seis quadros gerados por uma busca por abrangência realizada em uma *Slim-Tree* carregada com o conjunto IRIS, mas com páginas de disco de 1 Kb. O recurso "Tail" está ativado para enfatizar o trajeto do algoritmo. Os quadros mostram todos os passos do algoritmo enquanto este construía o conjunto de resposta. Apenas os 2 nós folha que poderiam conter respostas para a consulta foram visitados. Um terceiro nó, que aparentemente poderia fazer parte da resposta pode ser descartado com o uso de rotações nas cenas (não há intersecção entre a região de resposta e este nó). O último quadro mostra a resposta final.

A Figura 7.3 ilustra os seis quadros gerados por uma busca pelos vizinhos mais próximos realizada na mesma estrutura da Figura 7.2. Inicialmente, o raio coberto é infinito, sendo definido pela primeira vez no quadro 3 onde os primeiros 5 objetos são incluídos. No quadro 4, objetos mais próximos são encontrados, eliminando alguns objetos incluídos anteriormente e reduzindo o raio. Assim como na animação da consulta por abrangência, o último quadro mostra a resposta final.

A Figura 7.4 mostra o "dump" visual de uma *Slim-Tree* com páginas de disco de

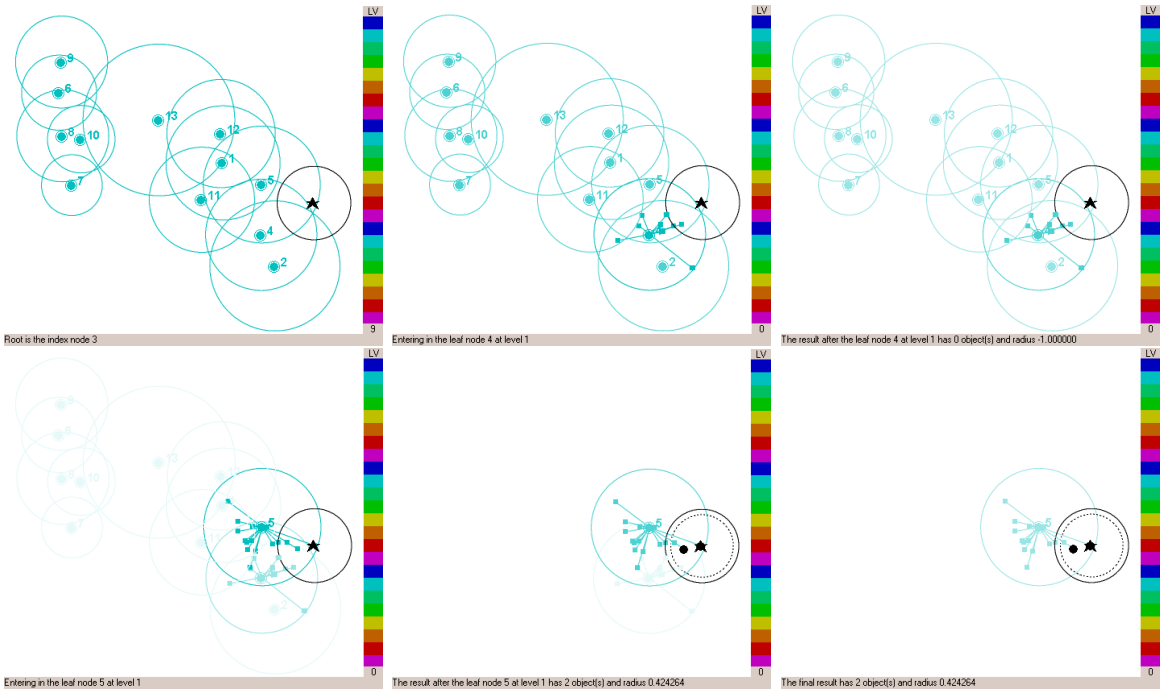

Figura 7.2: Animação completa de uma consulta por abrangência com raio 1 realizada na *Slim-Tree* carregada com o conjunto IRIS. A estrela preta representa o objeto de consulta e o círculo preto, seu raio de cobertura. Os elementos visitados em cada quadro estão com maior ênfase devido ao uso do recurso "Tail".

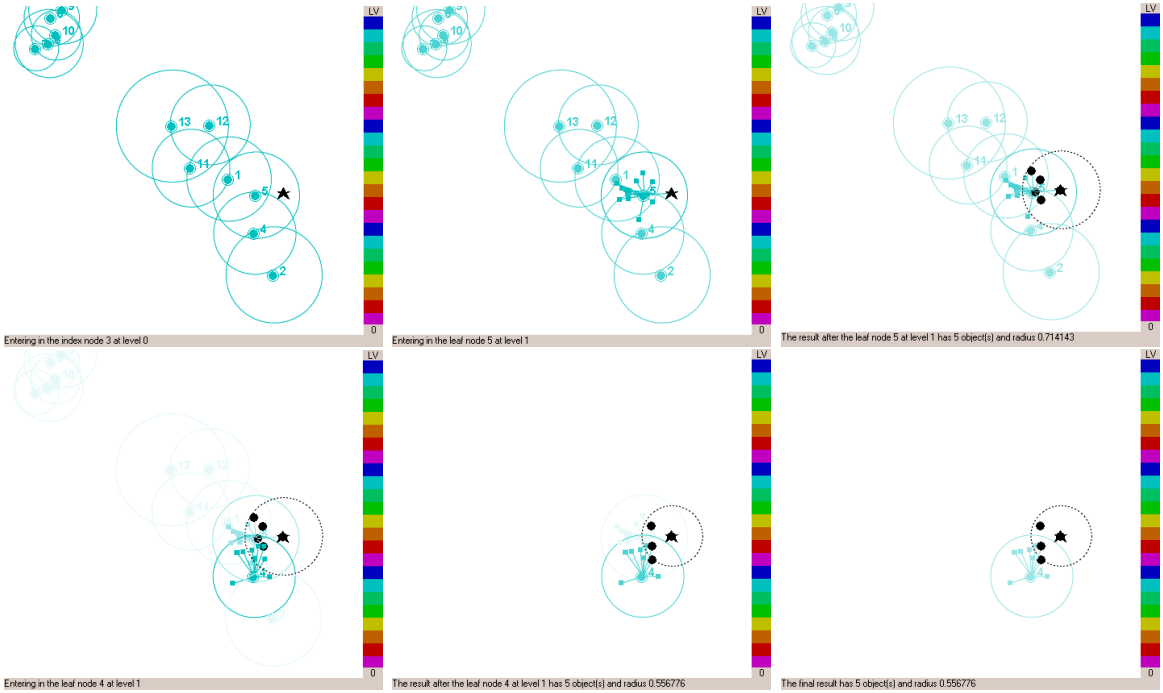

Figura 7.3: Animação completa de uma consulta pelos 4 vizinhos mais próximos realizada na mesma estrutura da Figura 7.2. É possível perceber a inclusão e remoção de objetos do conjunto de resposta conforme objetos mais próximos são encontrados.

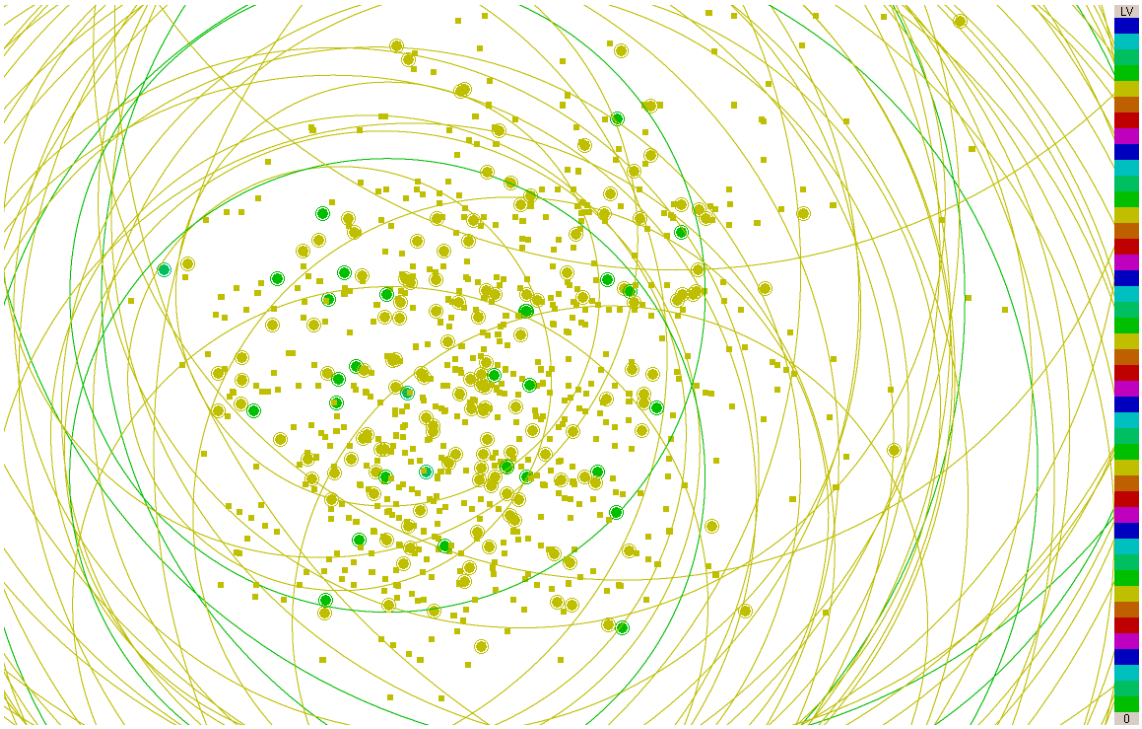

Figura 7.4: Dump visual de uma *Slim-Tree* com páginas de 512 bytes carregada com 1000 palavras inglesas usando a distância de Levenshtein.

512 bytes carregada com 1000 palavras inglesas. Esta árvore utilizou a função de distância de Levenshtein [Levenshtein, 1966]. A combinação do conjunto de palavras com a função de distância de Levenshtein constitui um conjunto de dados adimensional[1], demonstrando a capacidade do sistema *MAMView* de exibir quaisquer tipos de dados métricos.

7.2.1 Escalabilidade do Sistema

O *MAMView* foi testado também com conjuntos de dados grandes, alguns com dezenas de milhares de objetos. Com estes conjuntos de dados, foram realizadas consultas e outras operações que visavam criar animações grandes, composta por milhares de quadros de animação.

O *MAMView Extractor* foi capaz de gerar os arquivos de saída sem problemas, um deles quase 1,2GB. O consumo de memória e processamento deste módulo independe do número de objetos armazenados na estrutura, portanto este é capaz de manipular quaisquer quantidades de objetos exceto por limitações de tamanho do arquivo de saída MVA. Este tamanho é limitado pelos sistemas de arquivo utilizados.

Por outro lado, o *MAMViewer* é consideravelmente mais sensível a este aumento, pois seu consumo de memória e processamento é dependente do número de objetos armazenados e do número de quadros em que se divide cada animação. A escalabilidade do

[1]Conjuntos de dados adimensionais são conjuntos de dados compostos por elementos que não podem ser identificados por meio de coordenadas em eixos ortogonais.

MAMViewer é dependente, principalmente da quantidade de memória disponível no computador utilizado. Os testes mostraram que a grande maioria das animações e "dumps" gerados em condições normais de funcionamento de um MAM podem ser exibidos sem problemas em computadores com 256MB de memória.

Excluindo a capacidade do sistema em lidar com grandes quantidades de informações, deve-se considerar os usuários do visualizador. No caso de "dumps" de estruturas inteiras, grandes quantidades de objetos e elementos nas estruturas limitam consideravelmente a capacidade do usuário em entender as representações visuais geradas, mesmo com o uso dos recursos de filtragem.

Já as representações visuais de algoritmos podem ser entendidas mais facilmente, pois a informação temporal presente nestas animações permite o uso do recurso "Tail" para filtrar as informações pouco relevantes para o algoritmo em cada passo.

7.3 MAMView em um Ambiente de Desenvolvimento Real: DBM-Tree

Durante o desenvolvimento do *MAMView*, foi possível avaliar sua utilidade e aplicação em um processo de desenvolvimento de MAM real. A *DBM-Tree*, descrita na Seção 3.3, é uma estrutura ainda em desenvolvimento que está utilizando o *MAMView* como ferramenta de depuração e análise.

Esta seção traz alguns casos reais, utilizados durante o processo de desenvolvimento da *DBM-Tree*, que ajudaram a esclarecer algumas das características da estrutura, levando a melhorias em seus algoritmos. Estas figuras foram geradas utilizando um conjunto de coordenadas das cidades brasileiras e a função de distância euclidiana. Este conjunto foi escolhido por ter suas características de distribuição de distâncias bastante conhecidas e não por constituir um conjunto facilmente representado (a representação visual do conjunto foi obtida com o uso do *FastMap*, não sendo utilizadas suas coordenadas reais).

A Figura 7.5 ilustra a primeira representação visual de uma *DBM-Tree* gerada pelo *MAMView*. Esta representação mostrou que a profundidade é maior em regiões com maior densidade de objetos e que estes estão armazenados em todos os níveis da estrutura, como o esperado.

Porém, observou-se também que o nível de sobreposição entre as subárvores estava excessivamente grande, existindo até mesmo subárvores completamente cobertas por outras em um mesmo nível. Isto explicava os resultados pouco satisfatórios obtidos nos testes de desempenho realizados anteriormente.

Concluiu-se que os algoritmos de construção utilizados não estavam sendo capazes de obter vantagens significativas com a flexibilização do balanceamento da estrutura. A

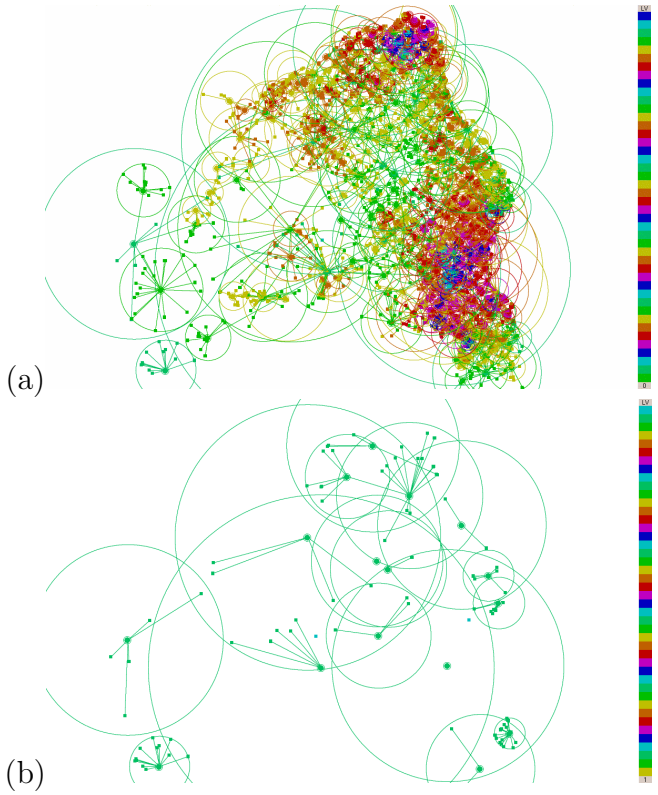

Figura 7.5: Primeira representação visual da *DBM-Tree* criada pelo *MAMView* para o conjunto de cidades brasileiras. Esta figura mostra a variação de profundidade da estrutura de acordo com a densidade de objetos (a). Constatou-se também a existência de uma grande taxa de sobreposição entre os nós de um mesmo nível (b).

análise deste "dump" visual levou ainda a identificação de outras estratégias que poderiam ser aplicadas na estrutura, como a reinserção de subárvores em outras quando estas fossem totalmente cobertas por outras.

A Figura 7.6 já apresenta um segundo "dump" feito após algumas modificações nos algoritmos de construção da estrutura baseadas nos conhecimentos obtidos com o primeiro "dump" visual. É possível verificar uma mudança considerável na organização dos nós, com um maior aumento na profundidade da estrutura em regiões mais densas e uma taxa de sobreposição um pouco menor com regiões melhor distribuídas. Os resultados dos testes de desempenho apresentaram uma pequena melhora mas a taxa de sobreposição ainda é um pouco mais alta que a esperada. Assim como no primeiro caso, novas possíveis modificações foram identificadas, dando continuidade ao processo de desenvolvimento.

A Figura 7.7 ilustra alguns dos quadros da animação de uma consulta por abrangência realizada pela *DBM-Tree*. Esta animação mostrou claramente as diferenças entre o comportamento das buscas na *DBM-Tree* e na *Slim-Tree*. Ela também mostrou que o número de buscas em largura realizadas na estrutura era maior que o esperado inicialmente, comprovando os efeitos da sobreposição verificados na análise da representação da Figura 7.5.

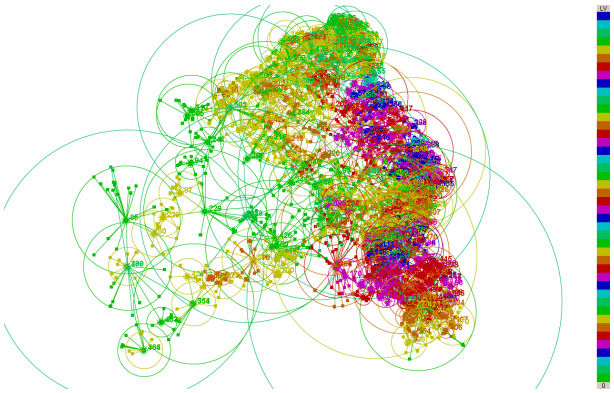

Figura 7.6: Representação visual da *DBM-Tree*, também para o conjunto de cidades brasileiras, após algumas modificações promovidas nos algoritmos de construção da estrutura com base nas informações obtidas com a primeira visualização (Figura 7.5).

Curiosamente, esta animação também promoveu uma pequena melhoria no algoritmo de consultas pelos vizinhos mais próximos da *DBM-Tree*. A inserção de objetos de um nó nas respostas de uma busca por abrangência dependia da ordem com que as entradas eram armazenadas nos nós. Como esta implementação da *DBM-Tree* permite que entradas de subárvores e de objetos sejam intercaladas, o algoritmo tendia a iniciar a busca em uma subárvore assim que sua entrada era encontrada, antes que todos os objetos do nó fossem processados. As consultas pelos vizinhos mais próximos apresentavam o mesmo comportamento.

Nas buscas por abrangência, a ordem de inserção dos objetos na resposta não modifica o caminho percorrido pelo algoritmo na estrutura, pois todos os objetos encontrados que sejam cobertos pelo raio da consulta são adicionados. Já nas consultas pelos vizinhos mais próximos, testar todos os objetos de um nó antes de continuar a busca em uma subárvore pode diminuir mais rapidamente o raio da consulta, aumentando a capacidade de poda de ramos da árvore, aumentando também sua eficiência. Como resultado desta observação, a implementação das consultas pelos vizinhos mais próximos foi modificada, aumentando um pouco o desempenho deste tipo de consulta.

A *MAMView* também foi utilizado para exibir o comportamento localizado dos algoritmos de quebra de nós da *DBM-Tree* e da *Slim-Tree* para que seus resultados pudessem ser comparados. As representações destes algoritmos permitiram que os resultados de pequenas modificações nestes algoritmos pudessem ser avaliadas e explicadas prontamente, sem a necessidade de testes de desempenho e de análises teóricas mais aprofundadas das regras de divisão experimentadas. Ou seja, a possibilidade de visualizar imediatamente o mecanismo de funcionamento de um MAM auxilia bastante o processo de seu desenvolvimento e aprimoramento.

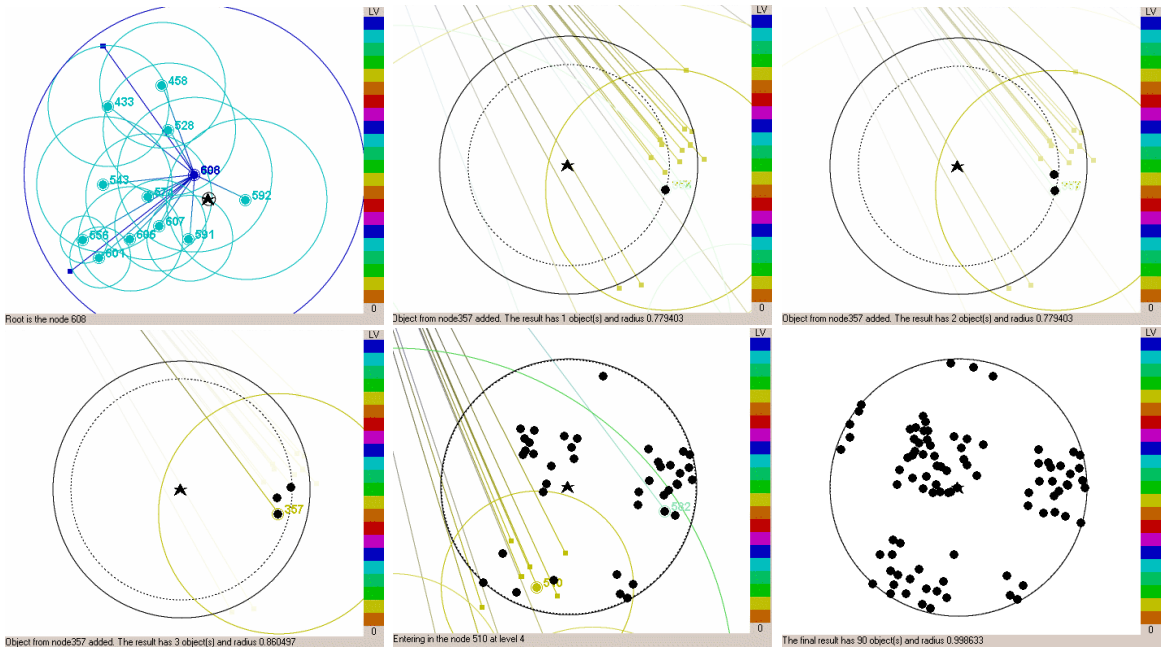

Figura 7.7: Alguns quadros da animação de uma consulta por abrangência de raio 1 realizada na *DBM-Tree* carregada com o conjunto de cidades brasileiras. Esta animação mostrou as diferenças no comportamento das buscas nesta estrutura e na *Slim-Tree*.

7.4 Comparação do MAMView com "Dumps" Textuais

Durante os estágios iniciais da implementação da biblioteca *GBDI Arboretum*, foi necessário criar meios que permitissem a depuração das estruturas enquanto eram implementadas. A depuração do código feita pelo uso de técnicas de inspeção e exploração de "dumps" textuais simples era bastante precária.

Como a implementação do sistema *MAMView* era dependente da implementação da própria biblioteca (devido à implementação do *MAMView Extractor*) e ainda encontrava-se em seus estágios iniciais de projeto, foi preciso criar uma evolução dos "dumps" textuais que pudessem ser explorados mais facilmente.

O resultado desta necessidade foi a construção de uma ferramenta de criação de "dumps" chamada de *SlimDump*[2]. Esta ferramenta foi inicialmente implementada para a *Slim-Tree* mas teve seus princípios de funcionamento adaptados a outras estruturas que também estavam sendo implementadas na *GBDI Arboretum*, como a *DBMDump* da *DBM-Tree*.

Esta ferramenta incorpora elementos de navegação de hipertexto aos "dumps" e utiliza cores para codificar informações sobre os tipos dos elementos das estruturas. Os

[2]Esta ferramenta foi criada originalmente pelo autor deste documento antes que o protótipo do sistema de visualização estivesse operacional. Alguns dos princípios planejados para este sistema de visualização, como o uso de cores e de interatividade, foram aplicados na construção desta ferramenta de "dump".

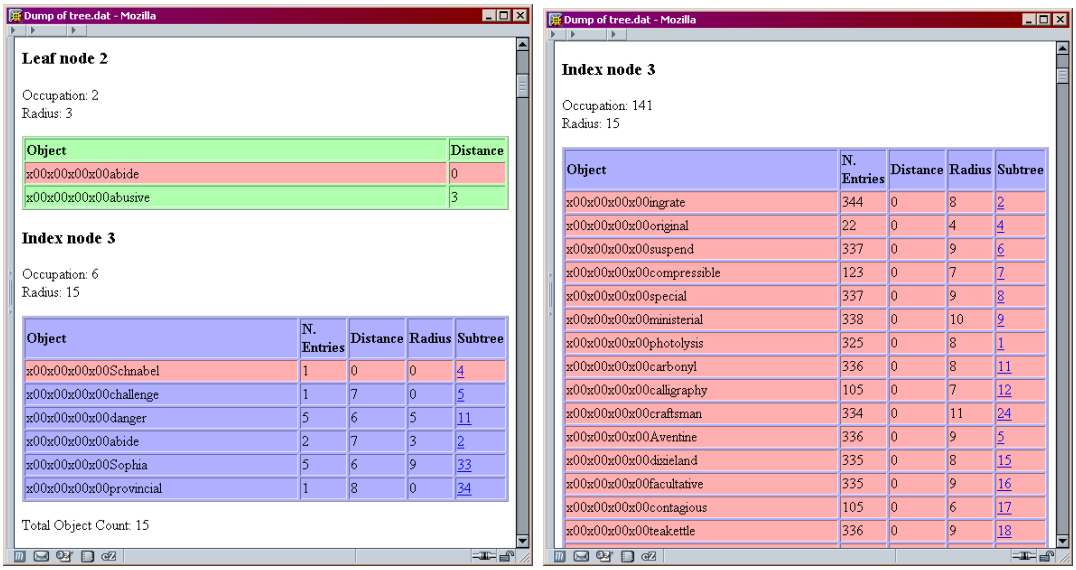

Figura 7.8: Exibições de Dumps textuais gerados pelo *SlimDump* para o conjunto de palavras inglesas. A imagem à esquerda representa uma estrutura com poucos objetos por nó enquanto a da direita ilustra uma árvore com ocupação alta. Todas as informações são apresentadas textualmente, incluindo o conteúdo dos objetos, que é uma representação do valor binário do objeto. Esta ferramenta de "dump" utiliza ainda recursos de hipertexto para permitir uma rápida navegação pela estrutura e cores para enfatizar os tipos de nós e de objetos.

"dumps" gerados são arquivos HTML que podem ser exibidos por um navegador de Internet e permitem navegar pelas estruturas. A Figura 7.8 ilustra exemplos de "dumps" gerados pelo *SlimDump* e pelo *DBMDump*.

A comparação entre estes "dumps" e as representações gráficas geradas pelo *MAMView* apresentam características bastante distintas. A Tabela 7.1 sintetiza estas diferenças.

Característica Representada	**MAMView**	**SlimDump e similares**
Organização Estrutural	X	X
Nós e objetos	X	X
Ligações	X	X
Nível dos elementos	X	X
Regiões de cobertura	X	-
Sobreposição de regiões	X	-
"Dump" da estrutura (estático)	X	X
Comportamento de algoritmos	X	-
Estado geral da estrutura	X	-

Tabela 7.1: Quadro comparativo entre as capacidades do *MAMView* e dos "dumps" textuais gerados pelo *SlimDump* e seus similares.

Além disto, a forma de apresentação visual empregada pelo *MAMView* mostra-se mais facilmente assimilável pelo sistema cognitivo humano que sua contra partida baseada

em “dumps” textuais mais abstratos.

Porém, a experiência obtida no processo de desenvolvimento da *DBM-Tree* mostrou que o uso combinado das duas ferramentas, o *MAMView* e a família do *SlimDump* aumentam consideravelmente a capacidade dos desenvolvedores em extrair informações sobre o comportamento das estruturas.

O *MAMView* é capaz de exibir muito mais informações que os “dumps” gerados pelo *SlimDump* e seus similares. As representações visuais do *MAMView* enfatizam o estado global da estruturas em detrimento de uma apresentação mais detalhada de sua organização física, enquanto os “dumps” representam a organização estrutural de forma muito simples compensando a falta de capacidade de ilustrar a dinâmica da estrutura e seu estado global.

Em outras palavras, estes “dumps” podem ser utilizados como mapas capazes de complementar as representações gráficas do *MAMView*, que cobrem com mais ênfase os aspectos lógicos das estruturas, apresentando-as em uma forma mais próxima à sua organização física nos arquivos de dados.

7.5 Conclusão

Os testes realizados com a *Slim-Tree* foram utilizados para validar o sistema *MAMView*. As representações visuais e as animações resultantes do uso do sistema de visualização em instâncias desta estrutura são perfeitamente coerentes e refletem exatamente o comportamento conhecido da *Slim-Tree*.

Já os testes realizados com a *DBM-Tree* visam demonstrar a utilidade do sistema no processo de desenvolvimento de novas estruturas. As contribuições do sistema *MAM-View* para o desenvolvimento desta estrutura têm sido bastante grandes, permitindo que as propriedades resultantes de cada modificação na estrutura sejam levantadas mais rapidamente, acelerando o processo de desenvolvimento.

A comparação entre as informações apresentadas pelo *MAMView* e por um sistema de “dump” textual melhorado deixa claro que o tipo de informação que pode ser obtida por cada uma destas ferramentas é bastante diferente. Enquanto os “dumps” textuais são bastante eficientes para exibir a organização destes MAM em nós e subárvores, é muito árduo avaliar o estado global das estruturas e seu comportamento. Já o *MAMView* é capaz de exibir não só a organização das estruturas em nós e subárvores como também é capaz de exibir as relações de distância existentes entre os objetos armazenados, as regiões de cobertura das subárvores e o comportamento das estruturas, levando a um entendimento maior das propriedades de um MAM.

Estes resultados ilustram apenas algumas das possíveis vantagens do uso de um sistema de visualização de MAM como o *MAMView* no estudo e desenvolvimento de

MAM. Com ferramentas deste tipo, é possível reduzir o tempo necessário para compreender o funcionamento destas estruturas e ainda levar a descoberta de propriedades que certamente passariam despercebidas sem o uso deste tipo de recurso.

Capítulo

8

Conclusões

8.1 Considerações Finais

O desenvolvimento de métodos de acesso métricos é essencial para o uso prático de consultas por similaridade em SGBD que gerenciam informações não convencionais, como dados multimídia e cadeias de DNA entre outras. Infelizmente a grande complexidade destas estruturas e a própria natureza dos dados indexados e suas relações de distância dificultam consideravelmente a descoberta de novas propriedades e a compreensão do funcionamento dos MAM, tornando árdua a tarefa de desenvolvê-los.

Ferramentas como o sistema *MAMView*, proposto por este trabalho, permitem que os desenvolvedores de MAM possam realizar inspeções visuais em suas estruturas, aumentando sua capacidade de compreender não só a organização, como também o comportamento destes MAM, podendo levar à descoberta de novas propriedades que, de outro modo, passariam despercebidas.

De posse de uma ferramenta destas, é possível ainda avaliar prontamente os efeitos de alterações na estrutura permitindo a comparação com resultados anteriores e com outras estruturas, diminuindo o tempo necessário para avaliar cada alteração. Esta ferramenta pode ainda ser utilizada por desenvolvedores de aplicações que utilizam MAM para encontrar os parâmetros de configuração de uma determinada estrutura que podem levar a consultas por similaridade mais eficientes.

A existência de uma ferramenta como o *MAMView* é de grande valia não só para os desenvolvedores de MAM, que terão uma poderosa ferramenta de apoio ao desenvolvimento de novas estruturas mais eficientes, como também para os usuários de MAM, que

poderão se beneficiar com a disponibilidade destas estruturas.

8.2 Sumário dos Resultados Obtidos

Entre os resultados obtidos por este projeto, deve-se enfatizar:

- O levantamento dos requisitos mínimos necessários para a criação de uma aplicação de visualização de estruturas métricas, sem os quais seria impossível criar a base teórica necessária para a implementação do sistema *MAMView*.
- Criação de um modelo genérico para representação da organização e do comportamento de MAM. Este modelo é capaz de representar vários MAM.
- Definição de um modelo de representação visual de estruturas métricas capaz de representar os principais aspectos da organização e do comportamento destas estruturas.
- Criação do recurso de visualização "tail", utilizado para enfatizar apenas os elementos de interesse dos algoritmos realizados nas estruturas métricas.
- Elaboração de uma arquitetura básica para sistemas de visualização de MAM. Esta arquitetura poderá ser utilizada como base para a criação de outras ferramentas de visualização de estruturas métricas ou ainda adaptado para outros tipos de estrutura.
- Criação e disponibilização do sistema *MAMView*, um visualizador de estruturas métricas totalmente funcional e que pode ser utilizado para visualizar muito dos MAM existentes e/ou em desenvolvimento.

8.3 Propostas Para Trabalhos Futuros

O trabalho de desenvolvimento do sistema *MAMView* abriu ainda a possibilidade de novas frentes de pesquisa, entre as quais encontram-se:

- O estudo da aplicação de outras técnicas de visualização de informação no visualizador de estruturas. É possível que outras técnicas utilizadas para visualização de informações possam aumentar a capacidade dos usuários de entender as representações exibidas.
- A avaliação da viabilidade de outros algoritmos mapeadores de distância. Diferentes conjuntos de dados podem ter mapeamentos de melhor qualidade com a aplicação de algoritmos distintos.

- A criação de módulos personalizados de análise automatizada ou semi-automatizada capazes de extrair informações sobre o comportamento dos MAM usando informações da simulação das estruturas. Estes módulos seriam sistemas especialistas, criados para detectar anomalias no comportamento de um determinado MAM, adicionando um pouco de semântica à simulação da estrutura.

- O aperfeiçoamento do modelo de representação de organização e comportamento para que possam englobar mais estruturas. Espera-se que o modelo de organização de comportamento de MAM possa ser modificado para representar mais estruturas métricas e até mesmo outros tipos de estruturas, permitindo que sejam também visualizadas.

- O estudo mais detalhado do impacto do uso do sistema de visualização de MAM no desenvolvimento de novas estruturas. Embora o uso do *MAMView* no desenvolvimento da *DBM-Tree* tenha servido para testar a viabilidade do sistema, não foi feito um estudo muito profundo sobre o impacto da ferramenta no processo de desenvolvimento da estrutura. Sabe-se apenas que o uso do *MAMView* permitiu a descoberta mais rápida das propriedades da estrutura mas não se sabe quanto tempo foi efetivamente economizado e se estas descobertas poderiam ter sido obtidas de outras formas.

- A criação de um simulador de MAM capaz de assumir o comportamento das estruturas por meio de regras de comportamento. Tal simulador poderia ser utilizado tanto para a visualização quanto para a criação de protótipos rápidos de estruturas. Estes protótipos poderiam ser utilizados para uma primeira avaliação de novos conceitos de construção de estruturas antes que estes sejam implementados.

Referências Bibliográficas

[Baker et al., 1999] Baker, R., Boilen, M., Goodrich, M. T., Tamassia, R., e Stibel, B. A. (1999). Testers and visualizers for teaching data structures. In *SIGCSE's Technical Symposium 1999*, New Orleans, Louisiana, USA.

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

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

[Bray et al., 2000] Bray, T., Paoli, J., Sperberg-McQueen, C. M., e Maler, E. (2000). Extensible markup language (xml) 1.0 - second edition. W3C Reccomendation.

[Brin, 1995] Brin, S. (1995). Near neighbor search in large metric spaces. In Dayal, U., Gray, P. M. D., e Nishio, S., editors, *VLDB '95: proceedings of the 21st International Conference on Very Large Data Bases, Zurich, Switzerland, Sept. 11–15, 1995*, p. 574–584, Los Altos, CA 94022, USA. Morgan Kaufmann Publishers.

[Bueno et al., 2002] Bueno, J. M., Traina, A. J. M., Traina Jr., C., e de Azevedo-Marques, P. M. (2002). cbPACS: Pacs com suporte à recuperação de imagens médicas baseada em conteúdo. In *VIII Congresso Brasileiro de Informática em Saúde- CBIS2002*, page 6 pags., Natal, RN-Brazil.

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

[Cao & Huang, 2000] Cao, X. e Huang, H. (2000). Current status and future advances of digital radiography and pacs. *IEEE Engineering in Medicine and Biology Magazine*, 9(5):80–88.

[Chino & Traina, 2001] Chino, F. J. T. e Traina, A. J. M. (2001). Slimview: Uma ferramenta para visualização de informações para a slim-tree. Monografia de Projeto de Graduação, ICMC-USP.

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

[Ciaccia et al., 1997] Ciaccia, P., Patella, M., e Zezula, P. (1997). M-tree: An efficient access method for similarity search in metric spaces. In Jarke, M., Carey, M. J., Dittrich, K. R., Lochovsky, F. H., Loucopoulos, P., e Jeusfeld, M. A., editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, p. 426–435. Morgan Kaufmann.

[Faloutsos & Lin, 1995] Faloutsos, C. e Lin, K.-I. (1995). FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In Carey, M. J. e Schneider, D. A., editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, p. 163–174, San Jose, California.

[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/.

[Guttman, 1984] Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In Yormark, B., editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, p. 47–57. ACM Press.

[Hjaltason & Samet, 2000] Hjaltason, G. e Samet, H. (2000). Contractive embedding methods for similarity searching in metric spaces. University of Maryland Computer Science TR 4102.

[Hristescu & Farach-Colton, 1999] Hristescu, G. e Farach-Colton, M. (1999). Cluster-preserving embedding of proteins. Technical Report 99-50, DIMACS.

[Kornacker et al., 2003] Kornacker, M., Shah, M. A., e Hellerstein, J. M. (2003). amdb: A design tool for access methods. *Data Engineering Bulletin*, 26(2).

[Kruskal, 1956] Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50.

[Kruskal & Wish, 1978] Kruskal, J. B. e Wish, M. (1978). *Multidimensional Scaling*. Sage Piblications, Beverly Hills, CA.

[Levenshtein, 1966] Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710.

[Marsh, 1997] Marsh, A. (1997). Euromed - the creation of a telemedical information society. *10' IEEE Symposium on Computer Based Medical Systems, Maribor, Slovenia*.

[Rosa et al., 2002] Rosa, N. A., Santos Filho, R. F., Bueno, J. M., Traina, A. J. M., e Traina Jr., C. (2002). Sistema de recuperação de imagens similares em um hospital universitário. In *VIII Congresso Brasileiro de Informática em Saúde- CBIS'2002*, Natal, RN-Brazil.

[Shah et al., 1999] Shah, M., Kornacker, M., e Hellerstein, J. M. (1999). amdb: A visual access method development tool. In *Int'l Wksp. on User Interfaces to Data Intensive Systems*, Edinburgh, Scotland.

[Siegel & Kolodner, 1999] Siegel, E. L. e Kolodner, R. M. (1999). *Filmless Radiology*. Springer Verlag, New York City, NY.

[Traina et al., 2001a] Traina, A., Traina Jr., C., Papadimitriou, S., e Faloutsos, C. (2001a). Tri-plots: Scalable tools for multidimensional data mining. In Provost, F. e Srikant, R., editors, *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-01)*, p. 184–193, New York. ACM Press.

[Traina et al., 2001b] Traina, A. J. M., Traina Jr., C., Barioni, M. C. N., Botelho, E., e Bueno, R. (2001b). Visualização de dados em sistemas de bancos de dados relacionais. In *XVI Simpósio Brasileiro de Banco de Dados, Rio de Janeiro, Brazil*.

[Traina et al., 2002a] Traina, A. J. M., Traina Jr., C., Bueno, J. M., e Azevedo-Marques, P. M. (2002a). The metric histogram: A new and efficient approach for content-based image retrieval. *VDB 2002*, p. 297–311.

[Traina et al., 2002b] Traina, Caetano, J., Traina, A. J. M., Faloutsos, C., e Seeger, B. (2002b). Fast indexing and visualization of metric datasets using slim-trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):244–260.

[Traina Jr. et al., 2000a] Traina Jr., C., Traina, A., Seeger, B., e Faloutsos, C. (2000a). Slim-Trees: High performance metric trees minimizing overlap between nodes. In Zaniolo, C., Lockemann, P. C., Scholl, M. H., e Grust, T., editors, *Intl. Conf. on Extending Database Technology*, v. 1777 of *Lecture Notes in Computer Science*, p. 51–65, Konstanz, Germany. Springer.

[Traina Jr. et al., 2000b] Traina Jr., C., Traina, A. J. M., e Faloutsos, C. (2000b). *Fast-MapDB User's Manual*. Carnegie Mellon University - School of Computer Science.

[Uhlmann, 1991] Uhlmann, J. K. (1991). Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179.

[Vieira & Traina Jr., 2004] Vieira, M. R. e Traina Jr., C. (2004). Dbm-tree: Método de acesso métrico sensível a densidade local. Projeto de Mestrado no ICMC-USP em andamento.

[Wang et al., 1999] Wang, J. T.-L., Wang, X., Lin, K.-I., Shasha, D., Shapiro, B. A., e Zhang, K. (1999). Evaluating a class of distance-mapping algorithms for data mining and clustering. In *Knowledge Discovery and Data Mining*, p. 307–311.

[Ware, 2000] Ware, C. (2000). *Information Visualization: Perception for design*. Morgan Kaufmann Publishers.

[Yianilos, 1993] Yianilos (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*.

Apêndice

A

MAMView Animation File (MVA)

A.1 Descrição do MAMView Animation File

O formato utilizado pelo sistema *MAMView* para a comunicação entre os módulos, chamado de *MAMView Animation File* ou *MVA*, é, na realidade, um arquivo XML [Bray et al., 2000], cujo DTD é apresentado em A.3 e sua interpretação é detalhada em A.4. Arquivos com este formato são gerados pelo *MAMView Extractor* (o módulo de extração) e são interpretados pelo *MAMViewer* (o módulo de visualização).

A.2 Exemplo Simples

Este é um exemplo bem simples de um arquivo MVA contendo pelo menos uma declaração possível para cada entidade definida pelo DTD. Este exemplo não representa nenhuma atividade em particular, terá apenas um quadro e todos os objetos posicionados na origem $(0, 0, 0)$. O comando declarado não existe, porem, a implementação do *MAMViewer* está preparado para ignorá-lo pois este tipo de extensão é permitido em versões personalizadas do par extrator/visualizador

```
<?xml version="1.0"?>
<!DOCTYPE mamview SYSTEM "mamview.dtd">
<animation>
   <description>
      <title>
       Animation's title.
      </title>
      <comment>
         Animation's comment.
```

```
      </comment>
   </description>
   <frame id="0">
      <comment>
         Frame comments.
      </comment>
      <node id="0" active="true" parent="0" type="0" level="0">
         <repobj x="0" y="0" z="0" radius="0" />
      </node>
      <object active="true" x="0" y="0" z="0" level="0" parent="0"/>
      <query>
         <sample x="0" y="0" z="0" k="0" innerRadius="0" outerRadius="0" radius="0" />
         <answer x="0" y="0" z="0" />
      </query>
      <command name="commandName">
         <param name="parmName" value="0" />
         <xyzparam name="parmName" x="0" y="0" z="0" />
      </command>
</frame>
</animation>
```

A.3 DTD do Formato MVA

```
<!ELEMENT animation (description, frame+)>

<!ELEMENT description (title, comment)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT comment (#PCDATA)>

<!ELEMENT frame ((comment|node|object|query|command)+) >
<!ATTLIST frame
id ID #REQUIRED>

<!ELEMENT object EMPTY>
<!ATTLIST object
active (true|false) #REQUIRED
x CDATA #REQUIRED
y CDATA #REQUIRED
z CDATA #REQUIRED
level CDATA #REQUIRED
parent CDATA #REQUIRED>

<!ELEMENT node (repobj+)>
<!ATTLIST node
id CDATA #REQUIRED
active (true|false) #REQUIRED
parent CDATA #REQUIRED
level CDATA #REQUIRED
type CDATA #REQUIRED>

<!ELEMENT repobj EMPTY>
<!ATTLIST repobj
x CDATA #REQUIRED
y CDATA #REQUIRED
z CDATA #REQUIRED
radius CDATA #REQUIRED>

<!ELEMENT query (sample+, answer*)>

<!ELEMENT answer EMPTY>
<!ATTLIST answer
x CDATA #REQUIRED
y CDATA #REQUIRED
z CDATA #REQUIRED>

<!ELEMENT sample EMPTY>
<!ATTLIST sample
x CDATA #REQUIRED
```

```
y CDATA #REQUIRED
z CDATA #REQUIRED
innerRadius CDATA #REQUIRED
outerRadius CDATA #REQUIRED
radius CDATA #REQUIRED>

<!ELEMENT command ((param|xyzparam)*)>
<!ATTLIST command
name CDATA #REQUIRED>

<!ELEMENT param EMPTY>
<!ATTLIST param
name CDATA #REQUIRED
value CDATA #REQUIRED>

<!ELEMENT xyzparam EMPTY>
<!ATTLIST xyzparam
name CDATA #REQUIRED
x CDATA #REQUIRED
y CDATA #REQUIRED
z CDATA #REQUIRED>
```

A.4 Interpretação das Entidades do MVA

A.4.1 /animation

Define uma animação. Deve conter apenas uma entidade *description* e pelo menos uma entidade *frame*.

Um arquivo MVA só pode conter uma única entidade deste tipo.

A.4.2 /animation/description

Define o bloco de descrição da animação. Cada arquivo deve conter apenas uma entidade deste tipo, sendo também a primeira a ser declarada dentro de uma animação.

A.4.3 /animation/description/title

Define o título da animação.

A.4.4 /animation/description/comment

Define o comentário da animação.

A.4.5 /animation/frame

Define um frame da animação. Possui um atributo chamado *id* que determina o índice do quadro na animação. Deve conter pelo menos um evento.

A.4.6 /animation/frame/comment

Comentário atribuído a um frame. Tecnicamente, um *frame* pode conter várias entidades deste tipo, porém, apenas a última é considerada.

A.4.7 /animation/frame/object

Declara um objeto. Possui os atributos *active* (valor booleano que indica o valor da ênfase); x, y e z (coordenadas do objeto); *level* (nível do objeto na árvore); e *parent* (nó pai do objeto).

Cada declaração de um objeto irá gerar um novo objeto mesmo que todos os atributos possuam os mesmos valores.

A.4.8 /animation/frame/node

Declara um nó. Possui os atributos *id* (nome do nó), *active* (valor booleano que indica o valor da ênfase), *parent* (nó pai do objeto), *type* (tipo do nó, atualmente apenas o valor 0 é possível) e *level* (nível do objeto na árvore).

Quando dois nós são declarados com um mesmo identificador (*id*), a última declaração substituirá o nó anteriormente declarado. Os nós e objetos que possuem este identificador como pai não são afetados.

A.4.9 /animation/frame/node/repobj

Declara o objeto representativo de um nó. Possui os atributos x, y e z para as coordenadas e *radius* para determinar a área de cobertura.

A.4.10 /animation/frame/query

Representa a resposta de uma busca. Cada vez que for declarada, substituirá totalmente o resultado anterior.

A.4.11 /animation/frame/query/sample

Representa um objeto de consulta ou exemplo. Possui os atributos x, y e z para representar as coordenadas do objeto; k para o número de objetos de resposta em uma *kNN* (0 para ser ignorado); *innerRadius* para o raio interno da busca (0 para ser ignorado); *outerRadius* para o raio externo da busca; e *radius* para o raio atual da resposta.

A.4.12 /animation/frame/query/answer

Declara um objeto que faz parte da resposta. Possui os atributos x, y e z para representar as coordenadas do objeto.

A.4.13 /animation/frame/command

Consiste na declaração de uma ação da estrutura. Possui apenas o atributo *name* que define o nome do comando.

É a mais flexível das entidades do MVA, permitindo a inclusão de vários tipos de ações. Os comandos válidos estão descritos em A.5.

A.4.14 /animation/frame/command/param

Parâmetro de um comando. Possui os atributos *name* e *value*.

A.4.15 /animation/frame/command/xyzparam

Parâmetro de um comando composto por 3 coordenadas. Possui os atributos *name*, x, y e z.

A.5 Comandos

A.5.1 ADDLEVEL()

Faz com que todas as entidades da estrutura tenham o nível aumentado em 1. É usado para simular o aumento de nível da árvore durante a inserção de um objeto.

A.5.2 SUBLEVEL()

Faz com que todas as entidades da estrutura tenham o nível diminuindo em 1. É usado para simular a redução de nível da árvore durante a remoção de um objeto;

A.5.3 ACTIVATEOBJ(parent: param, coords: xyzparam)

Enfatiza um determinado objeto (controle do "Tail Mode"). Pode ser usado para indicar que um objeto previamente declarado foi novamente visitado pelo algoritmo em questão. O primeiro parâmetro é o identificador do pai e o segundo as coordenadas do objeto. Apenas a primeira entrada não ativa que possuir este pai e estas coordenadas será enfatizada.

A.5.4 ACTIVATENODE(nodeid: param)

Enfatiza um determinado nó (controle do "Tail Mode"). Pode ser usado para indicar que um nó previamente declarado foi novamente visitado pelo algoritmo em questão.

A.5.5 RMOBJ(parent: param, coords: xyzparam)

Elimina um determinado objeto da simulação. O primeiro parâmetro é o identificador do pai e o segundo as coordenadas do objeto. Apenas a primeira entrada que possuir este pai e estas coordenadas será eliminada.

A.5.6 RMOBJS(parent: param)

Elimina todos os objetos do pai em questão.

A.5.7 RMNODE(nodeid: param)

Elimina um determinado nó da simulação e todos os objetos (não nós) diretamente ligados a este. A eliminação de uma árvore deve ser declarada através de comandos individuais de remoção de nós.

A.6 Ordem das Entidades

As entidades do MVA podem ser declaradas em qualquer ordem, porém, serão interpretadas pelo simulador de MAM do *MAMViewer* na ordem em que são declaradas no arquivo. Em outras palavras, se existem duas ou mais entidades que representam eventos que se sobrepõem em um mesmo quadro, apenas o resultado final será exibido na simulação do quadro.

A.7 Estendendo o formato MVA

Os *parsers* MVA permitem que certos tipos de extensão sejam feitas no formato do arquivo sem que haja a necessidade de alterar os *parsers* mais antigos. Isto é possível desde que as seguintes regras sejam cumpridas:

- Nenhuma nova entidade pode ser adicionada;
- A interpretação de entidades e atributos existentes não podem ser alterados;
- Novos atributos podem ser adicionados às entidades existentes, porém, serão ignorados por *parsers* mais antigos;

- Novos comandos podem ser adicionados livremente, porém, serão ignorados por *parsers* mais antigos;
- Novos parâmetros podem ser adicionados aos comandos já definidos, porém, serão ignorados por *parsers* mais antigos.

A.8 Invariâncias dos Parsers MVA

Todo *parser* de arquivos MVA deve tomar as seguintes atitudes diante a situações anormais no arquivo MVA:

- **Falta de atributo obrigatório**: Deve levar a situação de erro irrecuperável;
- **Entidade não definida**: Deve levar a situação de erro irrecuperável;
- **Atributos não definidos**: Devem ser ignorados;
- **Comandos não suportados**: Devem ser ignorados;
- **Elementos com pai inexistente ou inválido**: São processados normalmente, porém não estarão ligados a nenhum nó.

Apêndice

B

Implementação do Sistema MAMView

B.1 Introdução

Como descrito no Capítulo 6, a implementação do sistema *MAMView* está dividida em dois módulos funcionais distintos. O *MAMView Extractor* está implementado dentro da biblioteca de MAM *GBDI Arboretum* [GBDI-ICMC-USP, 2004] enquanto o *MAMViewer* é um software independente, criado para a plataforma *MS Win32*. Tanto a biblioteca *GBDI Arboretum* como o *MAMViewer* podem ser encontrados no seguinte endereço:

http://gbdi.icmc.usp.br/arboretum

B.2 Implementação do MAMViewer

O *MAMViewer* é um software desenvolvido para a plataforma *MS Win32*, utilizando o compilador *Borland C++ Builder*. Seu código é composto por um núcleo portável, que inclui os subsistemas de simulação e parte do sistema de visualização, e uma parte não portável, que inclui apenas o *Sistema de Interação com o Usuário* e parte do *Motor Gráfico*.

O *Motor Gráfico* utilizado pelo *MAMViewer* foi projetado para ser o mais genérico e portável possível, permitindo o uso de "drivers" específicos para vários sistemas gráficos como o *OpenGL*[1] e o *DirectX* entre outros. Por ser bastante flexível, o *Motor Gráfico*

[1] http://www.opengl.org

desenvolvido para o *MAMViewer* é utilizado também por outras aplicações desenvolvidas pelo GBDI-USP.

A interpretação dos arquivos MVA (XML) foi implementada utilizando a biblioteca *LIBXML*, também conhecida como *The XML C parser and toolkit of Gnome*[2]. Esta biblioteca permitiu que a implementação do *Parser de XML* pudesse ser feita sem que fosse necessário tratar os detalhes do formato XML.

B.3 Implementação do MAMView Extractor

O *MAMView Extractor* implementado para validar o sistema *MAMView* é parte integrante da biblioteca *GBDI Arboretum* e encontra-se disponível junto com a distribuição desta biblioteca. Esta implementação é composta por um pequeno conjunto de classes que pode ser utilizada por todos os componentes da biblioteca.

Por se tratar de um recurso oneroso, o extrator é considerado um recurso opcional (pode ser ativado e desativado durante a compilação da biblioteca/aplicação conforme a necessidade), tendo sido instalado apenas em alguns procedimentos das estruturas que foram selecionados para validar o sistema *MAMView* ou estão sendo estudados por este (caso de alguns procedimentos experimentais da *Slim-Tree* e da *DBM-Tree*).

B.3.1 Classes do MAMView Extractor

O *MAMView Extractor* da *GBDI Arboretum* é composta pelas seguintes classes:

- **stFastMapImage**: Implementa uma imagem do *FastMap*. Esta classe é utilizada internamente pela classe *stFastMapper*;
- **stFastMapper**: Implementa o algoritmo *FastMap*. É utilizado como o *Mapeador de Distâncias* do *MAMView Extractor* mas pode ser utilizada por outras classes da biblioteca ou mesmo por outras aplicações;
- **stMAMViewExtractor**: A principal classe do *MAMView Extractor*. Implementa todos os sistemas do extrator, exceto o *Mapeador de Distâncias*;
- **stMAMViewObjectSample**: Classe utilitária utilizada para criar as amostras de objetos que são necessárias para inicializar o *FastMap*.

Este Apêndice apresenta apenas parte da interface da classe *stMAMViewExtactor*, por ser a única destas classes que é utilizada diretamente pelos desenvolvedores de MAM. A documentação completa desta classe e das demais pode ser encontrada na documentação da biblioteca *GBDI Arboretum*.

[2]http://www.xmlsoft.org/

B.3.2 Tipos da Básicos da Biblioteca GBDI Arboretum

Para compreender a interface da classe *stMAMViewExtactor*, descrita neste Apêndice, é necessário conhecer o significado de alguns tipos de dados definidos pela biblioteca *GBDI Arboretum*. Estes tipos são:

- **ObjectType**: Qualquer classe que respeite a interface *stObject* definida pela *GBDI Arboretum*. Esta classe representa um objeto armazenado nas estruturas;
- **EvaluatorType**: Qualquer classe que respeite a interface *stMetricEvaluator* definida pela *GBDI Arboretum*. Esta classe representa a função de distância entre dois objetos do tipo *ObjectType*;
- **stDistance**: Representa as distâncias entre objetos do conjunto de dados;
- **stPageID**: Identificador único de um nó. Representa o número da página que o nó ocupa;
- **stResult**: Esta classe implementa o resultado de uma consulta.

Mais informações sobre estes tipos de dados podem ser encontradas na documentação da biblioteca *GBDI Arboretum*.

B.3.3 Classe stMAMViewExtactor

B.3.3.1 Descrição

Esta classe implementa o *MAMView Extractor* descrito no Capítulo *cha:mamview*. Sua interface obedece aos padrões estabelecidos pela biblioteca *GBDI Arboretum*, sendo bastante familiar a esta.

Além das funcionalidades descritas pelo *MAMView Extractor*, esta implementação possui ainda facilidades para controle de níveis (útil para evitar modificações nos procedimentos das estruturas) e recursos simples para diagnóstico e correção de erros de instalação (os desenvolvedores de MAM são avisados e orientados sobre erros cometidos durante a instalação do extrator).

B.3.3.2 Construtores

```
tMAMViewExtractor (EvaluatorType *eval)
```

Função:

Cria uma nova instância desta classe.

Parâmetros:

eval: Instância do *MetricEvaluator* utilizada para calcular as distâncias entre os objetos. Esta instância é emprestada da estrutura.

B.3.3.3 Métodos de Inicialização

```
void Init(stMAMViewObjectSample *sample)
```

Função:

Prepara o extrator para ser utilizado. Nenhum outro método desta classe funcionará antes da execução deste método.

O parâmetro *sample* contém os objetos que serão utilizados para inicializar a instância do *stFastMapper*. Este conjunto deve possuir pelo menos 6 objetos, porém, quanto maior o número de objetos, maior a probabilidade do *FastMap* atingir sua qualidade ótima.

Parâmetros:

sample: Conjunto de objetos extraídos da estrutura ou do conjunto de dados que será utilizado para inicializar o algoritmo mapeador de distâncias (*FastMap*).

```
void SetOutputDir (const char *outputDir)
```

Função:

Determina o diretório base onde os arquivos MVA resultantes serão escritos. Se nenhum diretório for especificado, o extrator criará os arquivos no diretório atual.

Parâmetros:

outputDir: Caminho do diretório de saída.

B.3.3.4 Métodos de Controle de Animação

```
void BeginAnimation (const char *title, const char *comment)
```

Função:

Notifica para o extrator o início de uma animação. Sempre que este método for executado, um novo arquivo MVA será criado automaticamente.

Parâmetros:

title: Título da animação.
comment: Comentário sobre a animação.

`void EndAnimation()`

Função:

Notifica para o extrator o final de uma animação.

`void BeginFrame(const char *comment)`

Função:

Notifica para o extrator o início de uma *frame*.

Parâmetros:

comment: Comentário sobre o *frame*.

`void EndFrame()`

Função:

Notifica para o extrator o final de uma *frame*.

B.3.3.5 Métodos de Declaração de Eventos

`void SetNode(stPageID nodeID, ObjectType **reps, stDistance *radii, int n, stPageID parent, int type, bool active)`

Função:

Declara a existência de um nó ou a substituição de seu conteúdo. Este método espera nós com mais de um objeto representativo.

Parâmetros:

nodeID: Nome do nó.

reps: Objetos representativos do nó.

radii: Raios associados a cada representativo.

n: Número de representativos.

parent: Nome do nó pai. Valores inválidos resultarão em nós sem pai.

type: Tipo de nó. Este parâmetro ainda não é utilizado pelo *MAMViewer* mas está reservado para uso futuro.

active: Determina se o elemento deve é relevante para o procedimento (controle do "Tail").

`void SetNode(stPageID nodeID, ObjectType *rep, stDistance radius, stPageID parent, int type, bool active)`

Função:

Declara a existência de um nó ou a substituição de seu conteúdo. Este método espera nós com apenas um objeto representativo.

Este método existe apenas por conveniência, executando o método para múltiplos representativos com os parâmetros certos.

Parâmetros:

nodeID: Nome do nó.

rep: Objeto representativo do nó.

radius: Raio do nó.

parent: Nome do nó pai. Valores inválidos resultarão em nós sem pai.

type: Tipo de nó. Este parâmetro ainda não é utilizado pelo *MAMViewer* mas está reservado para uso futuro.

active: Determina se o elemento deve é relevante para o procedimento (controle do "Tail").

```
void EnableNode(stPageID nodeID)
```

Função:

Ativa um nó (comando ACTIVATENODE).

Parâmetros:

nodeID: Nome do nó a ser ativado.

```
void SetObject(ObjectType *obj, stPageID parent, bool active=false)
```

Função:

Declara a existência de um objeto.

Parâmetros:

obj: Objeto.

parent: Nome do nó pai. Valores inválidos resultarão em objetos sem pai.

active: Determina se o elemento deve é relevante para o procedimento (controle do "Tail").

```
void SetResult(ObjectType *queryObj, stResult *result, stCount k, stDistance radius)
```

Função:

Declara o estado do resultado de uma consulta.

Parâmetros:

queryObj: Objeto de consulta.
result: Resultado da consulta.
k: Número de objetos para uma consulta por abrangência.
radius: Raio de uma consulta por abrangência.

```
void SetResult(ObjectType *queryObj, stResult *result)
```

Função:

Declara o estado do resultado de uma consulta. Este método utiliza valores para *k* e *radius* previamente definidos no início da consulta (método de conveniência).

Parâmetros:

queryObj: Objeto de consulta.
result: Resultado da consulta.

```
void SetSample(ObjectType *queryObj)
```

Função:

Declara um resultado composto apenas pelo objeto de consulta.

Este método pode ser utilizado por procedimentos que não possuem respostas de buscas, como inserções (para representar o objeto que está sendo inserido).

Parâmetros:

queryObj: Objeto de consulta.

```
void SetQueryInfo(int k, stDistance radius)
```

Função:

Define os valores padrão para *k* e *radius* utilizados por alguns métodos de conveniência.

Pode ser executado a qualquer momento, porém, é aconselhável executá-lo apenas no inicio de animações de consultas quando estes parâmetros forem úteis.

Parâmetros:

k: Número de objetos para uma consulta por abrangência. Valores iguais ou menores que 0 (zero) fazem com que este parâmetro seja ignorado.

radius: Raio de uma consulta por abrangência. Valores negativos fazem com que este parâmetro seja ignorado.

`void SetLevel(int level)`

Função:

Define o nível no qual todos os eventos declarados ocorrem. Seu valor afeta todas as declarações de eventos.

Parâmetros:

level: Nível dos eventos na estrutura. Valores negativos são "arredondados" para zero.

`int GetLevel()`

Função:

Retorna o nível no qual os eventos estão sendo declarados.

`void LevelUp()`

Função:

Soma 1 no valor do nível atual. Pode ser utilizado para controlar o nível dos eventos em procedimentos recursivos quando as estruturas não o fazem.

`void LevelDown()`

Função:

Subtrai 1 do valor do nível atual. Pode ser utilizado para controlar o nível dos eventos em procedimentos recursivos quando as estruturas não o fazem.

B.3.4 Limitações da Classe stMAMViewExtactor

Todos os comandos definidos pela entidade <*command*> do formato MVA (ver Apêndice A), estão sendo implementados conforme a necessidade dos usuários do extrator. Portanto, alguns dos comandos definidos não podem ser gerados pela implementação do extrator da *GBDI Arboretum*. Entretanto, estes comandos podem ser facilmente implementados e deverão estar disponíveis em versões futuras do extrator.

Outra limitação encontrada nesta implementação do extrator é a ausência de resultados com múltiplos centros prevista pelo formato MVA. Isto se deve a limitações da própria biblioteca *GBDI Arboretum* que ainda não é capaz de executar consultas com múltiplos objetos de consulta. Assim que esta limitação da biblioteca for solucionada, este recurso será incorporado ao extrator.

Estas limitações não afetam o *MAMViewer* (visualizador), uma vez que constituem aplicações distintas.

Apêndice

C

Utilizando os MAM para Conjuntos de Dados Multimídia

Teoricamente, o uso das MAM para indexar conjuntos de dados multimídia depende apenas do conjunto de objetos e de uma função de distância métrica apropriada. Porém, o tamanho dos objetos indexados costuma ser grande o suficiente para degradar sua indexação direta pelas MAM.

Por exemplo, uma imagem com 512 por 512 com 16 bits por pixel, um tamanho comum para imagens médicas, ocupa 512KB de espaço de armazenamento. Se uma estrutura como a *Slim-Tree* for utilizada para indexar as imagens diretamente, sua página precisaria ter pouco mais de 1536KB (ver a Seção 3.2.1 para mais detalhes). Este valor excede em muitas vezes o tamanho de página de disco utilizado pela maioria dos sistemas de bases de dados.

Uma maneira de solucionar o problema dos tamanhos dos objetos é o uso de **vetores de características** extraídas dos objetos por algum algoritmo **extrator de características** ao invés dos próprios objetos. Este vetor deve ser capaz de sintetizar as informações mais relevantes do objeto original. Além disto, é necessário definir funções de distância métricas capazes de operar sobre estes vetores de características de forma a permitir efetuar comparações que sejam significativas para os dados originais.

Portanto, ao indexar um conjunto de imagens, deve-se definir um ou mais extratores de características e as respectivas funções de distância métricas para cada um dos conjuntos de vetores. Um bom exemplo disto são os **Histogramas Métricos** e sua função de distância [Traina et al., 2002a]. O mesmo princípio pode ser utilizado para outros

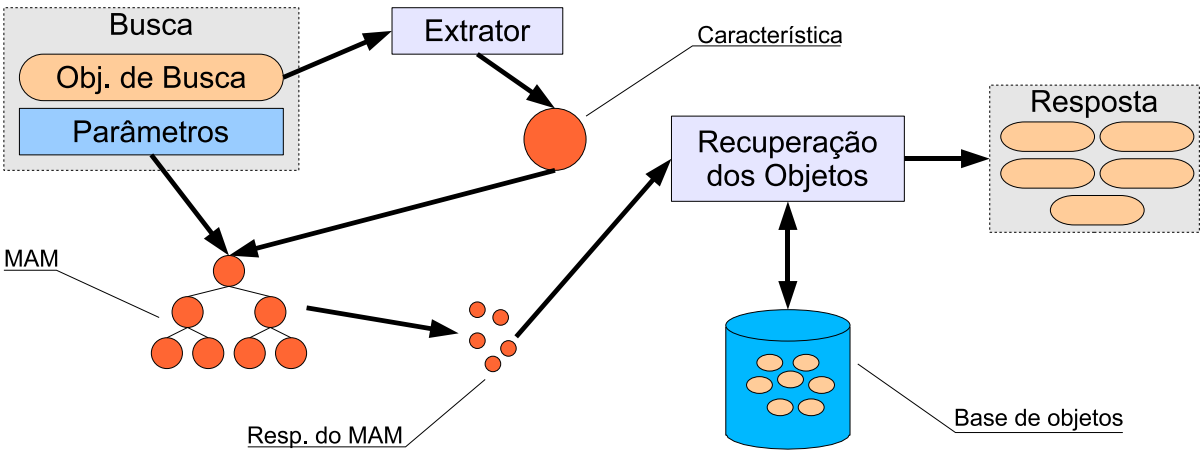

Figura C.1: Esquema de uma consulta por similaridade utilizando MAM. Os objetos (em bege) são indexados indiretamente, através de vetores de características (laranja). O objeto de consulta passa por um extrator para a obtenção de um vetor de características. A consulta é feita no MAM utilizando este vetor de características e os parâmetros especificados pelo usuário. O MAM retorna como resposta um conjunto de vetores de características, cada um associados a um objetos real armazenados na base de dados. Estes objetos são recuperados e enviados para o usuário como resposta.

domínios de objetos além de imagens.

Como os espaços métricos definidos pelos vetores de características e pelas suas funções de distância devem ser aproximações do espaço de objetos original, é possível utilizar MAM para indexar os vetores de características e obter resultados similares a consultas realizadas diretamente sobre o conjunto de dados original, mas a um custo computacional muito menor. Usando esta abordagem, uma busca por similaridade ocorre como o exemplificado na Figura C.1.

Observe que a qualidade das respostas depende tanto do extrator de características escolhido, que deve representar bem os objetos, quanto da função de distância definida para as características escolhidas.

Apêndice

D

Usando a Desigualdade Triangular

Dado um domínio de objetos $\mathfrak{D}$, uma função de distância $d : \mathfrak{D} \times \mathfrak{D} \rightarrow \Re$ e objetos $a, b, c \in \mathfrak{D}$, temos a inequação da desigualdade triangular:

$$d(a, b) \leq d(a, c) + d(c, b) \tag{D.1}$$

Esta equação permite inferir os limites superior e inferior de uma das distâncias conhecendo apenas os valores das outras duas distâncias.

Suponha que as distâncias entre a e c ($d(a, c)$) e entre b e c ($d(b, c)$) são conhecidas e deseja-se inferir os limites para o valor da distância entre a e b ($d(a, b)$), que será denotada por x para facilitar a compreensão. Usando a Equação D.1 temos:

$$x \leq d(a, c) + d(c, b) \tag{D.2}$$

$$d(a, c) \leq x + d(c, b) \Rightarrow x \geq d(c, b) - d(a, c) \tag{D.3}$$

$$d(c, b) \leq d(a, c) + x \Rightarrow x \geq d(a, c) - d(c, b) \tag{D.4}$$

Sabendo que $x \geq 0$, as Equações D.3 e D.4 podem ser combinadas, formando:

$$x \geq |d(a, c) - d(c, b)| \tag{D.5}$$

pois se $d(a, c) > d(c, b)$, o limite inferior será a Equação D.4, caso contrário, será a Equação D.3.

Combinando as Equações D.2 e D.5, temos:

$$|d(a,c) - d(c,b)| \leq x \leq d(a,c) + d(c,b) \tag{D.6}$$

ou seja:

$$|d(a,c) - d(c,b)| \leq d(a,b) \leq d(a,c) + d(c,b) \tag{D.7}$$

A Equação D.7 pode ser utilizada para descartar objetos pertencentes a consultas realizadas em MAM. Considerando uma consulta por abrangência com raio r, objeto de exemplo a, objeto candidato b e representativo de uma região c, temos:

$$d(a,b) \leq r, b \in resposta \tag{D.8}$$

$$d(a,b) > r, b \notin resposta \tag{D.9}$$

Sabendo apenas $d(a,c)$ e $d(c,b)$, é possível afirmar que b só pode ser candidato a resposta desta consulta se e somente se:

$$|d(a,c) - r| \leq d(c,b) \leq d(a,c) + r \tag{D.10}$$

partindo disto, sabe-se que $d(c,b)$ não se encontrar dentro deste intervalo, $d(a,b)$ certamente será maior que r e portanto pode ser descartado sem a necessidade de calcular seu valor. Caso contrário, o limite superior de $d(a,b)$ será $2d(a,c) + r$. Neste caso, $d(a,b)$ deve ser calculado para que possa ser comparado com r, descartando ou não o objeto da resposta dependendo de seu valor.

Este documento foi preparado com o formatador de textos LaTeX .