Estrutura de indexação em memória para dados métricos

Andréia Dal Ponte Novelli

Orientador: ***Prof. Dr. Caetano Traina Junior***

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

USP – São Carlos
Junho/2003

A Comissão Julgadora:

Prof. Dr. Caetano Traina Junior

Profa. Dra. Josiane Maria Bueno

Prof. Dr. Jander Moreira

A meus pais Geraldo e Cecília.

RESUMO

A recuperação de dados armazenados em Bancos de Dado em geral é feita utilizando estruturas de indexação, que permitem fazer a recuperação dos dados muito mais rapidamente do que se a busca fosse feita seqüencialmente. No entanto, as estruturas de indexação que podem ser utilizadas dependem das propriedades dos domínios de dados indexados e do tipo de consultas que devem ser respondidas. Tradicionalmente, os gerenciadores de bancos de dados suportam bem dados de domínios que possuem a propriedade de relação de ordem total, tais como números e textos com a relação de ordem lexicográfica permitindo consultas por igualdade e consultas envolvendo relações de ordem tais como $>, \leq, >, \lneq$ etc. Além disso, as estruturas de indexação comumente utilizadas em sistemas gerenciadores de bases de dados são construídas para serem armazenadas em disco, particionando o conjunto de dados em registros de tamanho fixo. O exemplo mais comum desse arranjo é o das árvores de indexação, quando os registros são então chamados "nós".

Aplicações mais sofisticadas freqüentemente apresentam dados em outros domínios, com outros tipos de consulta. Quando as aplicações lidam com dados em domínio métrico, além dos próprios elementos de dados, é definida uma função de similaridade (ou de distância) entre pares de elementos, e essa função é a única maneira de comparação entre dois elementos de dados do conjunto.

Existem diversas estruturas de indexação criadas para dados em domínios métricos . Entretanto essas estruturas ou são estáticas (impedindo que novos elementos sejam acrescentados depois que a estrutura foi criada), ou são para armazenamento em disco. Neste trabalho foi desenvolvida uma nova estrutura métrica dinâmica para dados métricos totalmente armazenada em memória principal. Uma outra propriedade interessante dessa estrutura é que a execução de uma consulta por existência (*point query*) percorre um único caminho de busca. Essa característica é muito interessante, pois todas as outras árvores dinâmicas existentes requerem que a navegação seja feita não apenas em profundidade, mas também em largura. A estrutura proposta permite a navegação apenas em profundidade para a consulta por existência.

ABSTRACT

The stored data recovery in Data bases, in general is made using indexation structures, that allow to much more quickly make the data recovery of that if the search was made sequentially. However, the indexation structures that can be used depend on the domains properties of the of indexing data and the type of queries that must be answered. Traditionally, the data bases management systems support well domains dataset that possess the property of relation of total order, such as numbers and texts with the lexicographical relation order allowing to queries for equality and queries involving order relations such as $>$, $\leq$, $>$ and $\leq$ Moreover, the commonly indexing structures used in data bases management systems are constructed to be stored in record, partitioning the data set in registers of so great fixture. The example most common of this arrangement is of the indexing trees, when the registers then are called "nodes".

Sophisticated applications more frequently present datasets in other domains, with other types of quey. When the applications deal with data in metric domain, beyond the proper data elements, a similarity function (or distance) between pairs of elements is defined, and this function is the only way of comparison enters two elements of data of the set.

There are diverse metric domain data indexing structures created. However these structures are static (hindering that new elements are added later that the structure was created), or are for record storage. In this work a new dynamic metric structure for metric data total stored in principal memory e was developed. One another interesting property of this structure is that the execution of a quey for existence (point query) covers an only way of search. This characteristic is very interesting, therefore all the other existing dynamic trees require that the navigation is made not only in depth, but also in width. The structure proposal allows the navigation only in depth for the quey for existence.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

A área de banco de dados pode ser considerada como uma área básica que dá suporte para as atividades da maioria das áreas da computação. Sua participação em qualquer projeto consiste em fornecer mecanismos de suporte à armazenagem, recuperação e atualização de dados. Várias propriedades desses mecanismos são fundamentais para o papel que se espera de um gerenciador de banco de dados, como, por exemplo, garantia de integridade (física e conceitual) nos dados armazenados, eficiência e corretitude na recuperação, controle de concorrência nas transações de atualização de dados, etc. O atendimento a essas propriedades é o que sempre se espera dos gerenciadores de bancos de dados, nos projetos em que ele participa.

Esse suporte vem sendo oferecido, desde o início da própria ciência da computação, concentrado principalmente na disponibilização de gerenciadores de dados de propósito geral, que são usados para volumes de dados grandes, persistentes, bem estruturados, com tipos de dados relativamente simples, que exigem respostas precisas, exaustivas e repetitivas.

Atualmente, algumas propriedades dos gerenciadores de banco de dados têm sido desconsideradas em aplicações que podem ser chamadas de não convencionais[1], mas para a maioria das aplicações estas propriedades se mantêm. O resultado disso é que as estruturas mais internas dos gerenciadores, como as estruturas de indexação e de controle de concorrência, do ponto de vista conceitual, se mantêm essencialmente as mesmas, uma vez que os requisitos que mudam para o gerenciador como um todo, em geral afetam apenas as camadas de mais alto nível do gerenciador.

[1] Aplicações que armazenam no banco de dados, tipos de dados que não são apenas numéricos ou textuais. Entre os tipos de dados armazenados encontram-se, por exemplo: áudio digitalizado, hipertextos e imagens em geral.

1.1- Motivação

Um exemplo de classe de aplicações que vem crescendo, tanto em importância quanto em variedade de requisitos que impõe aos gerenciadores, é a das aplicações para a *Web* que dependem da integração de dados. Existem várias maneiras de implementar uma ferramenta de integração de dados. Uma das maneiras é criar uma ferramenta denominada mediador. Dentre os principais requisitos de uma estrutura de indexação para um mediador de consultas na *Web* está a eficiência na resposta às consultas. Não há, a princípio, a restrição de que essas estruturas sejam armazenadas em disco, desde que possam ser criadas rapidamente.

Baseando-se nas características apresentadas por essas estruturas de indexação, verificou-se que, se os dados indexados são dados métricos, não existe na literatura uma estrutura que possa atender adequadamente os as necessidades impostas pelas estruturas de indexação para mediadores. Por exemplo, um domínio métrico surge na *Web*, se forem consideradas buscas por padrões que ocorrem nos textos das páginas dos *sites*. Desta forma, quando as buscas devem atender não apenas uma comparação exata, mas também buscas por palavras semelhantes com erros de digitação ou flexões, então se pode ter associadas às buscas um "fator de semelhança" com as palavras centrais da busca, originando um domínio de consulta métrico.

Portanto, a existência de aplicações, que necessitem de uma estrutura métrica em memória, e a carência de trabalhos na área de indexação de dados enfocando estruturas de indexação que tenham essa característica motivaram o desenvolvimento desse trabalho de mestrado.

1.2- Objetivos

Para domínios de dados que possuam a relação de ordem total, as árvores binárias, em particular quando balanceadas, podem ser úteis para a indexação em memória. O mesmo ocorre para domínios de dados espaciais pontuais, com a disponibilidade, por exemplo, da estrutura *k-d-tree*. No entanto, não existe nenhum método adequado para indexação em memória de dados métricos que seja voltado a responder consultas por similaridade. Desta forma, o primeiro objetivo deste projeto é o desenvolvimento de uma estrutura em memória para dados métricos voltada para responder às consultas por similaridade.

Tanto as estruturas para domínios métricos quanto as estruturas para domínios espaciais com armazenagem em disco existentes atualmente apresentam uma característica que prejudica seu desempenho para consultas, característica esta que não existe nas estruturas tradicionais que operam sobre dados em domínios ordenáveis. Essa característica é explicada a seguir, usando-se como exemplo a estrutura *Slim-tree*. Um nó da árvore *Slim-tree* define uma região de cobertura, que garante que todos os objetos armazenados em todas as sub-árvores que partem desse nó estão dentro da região de cobertura. No entanto, não existe garantia que todos os objetos que estejam dentro dessa região de cobertura estejam armazenados em uma sub-árvore particular que parte desse nó, pois outros nós que fazem parte de outras sub-árvores que partem desse nó podem também cobrir parte ou toda essa região. Essa intersecção não nula de regiões cobertas por nós que podem estar em um mesmo nível da árvore obriga que as operações de busca pesquisem cada uma das sub-árvores em profundidade. Isto é, para localizar se um determinado objeto está armazenado na árvore, sempre que em um dado nível, mais do que uma sub-árvore cobrir a região onde o objeto se encontra, todas essas sub-árvores terão que ser pesquisadas, avaliando diversos caminhos de busca até que o objeto seja encontrado, ou até que se decida que ele não está armazenado.

Tanto as árvores binárias para domínios de ordem total, quanto as *k-d-tree* para domínios espaciais pontuais garantem que apenas um único caminho de busca deve ser seguido para encontrar um objeto, ou decidir que ele não está armazenado. Assim, o segundo objetivo deste projeto é que a nova estrutura a ser desenvolvida deve permitir localizar ou decidir pela inexistência do objeto na estrutura percorrendo apenas um caminho de busca.

Um terceiro objetivo é que a estrutura seja dinâmica, ou seja, deve ser capaz de aceitar a inclusão de novos objetos depois que a estrutura esteja "pronta". No entanto, em geral não há a necessidade de remoção de alguns objetos. Por exemplo, no exemplo do mediador para *Web*, sempre que uma página for substituída do *cache*, toda a árvore deve ser invalidada. Uma operação de inclusão de um objeto por vez deve, portanto estar sempre disponível, embora a disponibilidade de incluir uma coleção de objetos simultaneamente, ou mesmo a possibilidade de mistura (*merging*) de duas árvores independentes construídas sobre dados do mesmo domínio sejam operações desejáveis. Assim, o terceiro objetivo deste projeto é que a estrutura desenvolvida seja dinâmica.

Por último, como as consultas por similaridade são as mais comuns sobre dados métricos, deverão ser desenvolvidos também os algoritmos para as operações de busca por vizinhos mais próximos e busca por abrangência para essa estrutura. Pretende-se ainda que as estruturas de dados desenvolvidas

e os algoritmos de inserção e busca na estrutura sejam implementados em um protótipo que permita a avaliação da estrutura.

1.3- Organização da Dissertação

No capítulo 1 encontra-se a introdução geral do trabalho, bem como a motivação para seu desenvolvimento e os objetivos seguidos.

No capitulo 2 apresenta-se o estado da arte das áreas de atividades em Banco de Dados centrados na indexação de dados.

No capitulo 3 apresentam-se alguns conceitos relativos a domínios métricos e apresentam-se mais detalhadamente algumas estruturas de indexação existentes mais relevantes para este trabalho.

No capitulo 4 apresenta-se a estrutura MM-Tree, a árvore que foi desenvolvida neste projeto de mestrado.

No Capitulo 5 são apresentados os resultados obtidos e a análise da estrutura desenvolvida.

No capitulo 6 é apresentada a conclusão da dissertação e as sugestões de trabalhos futuros.

CAPÍTULO 2

ESTUDO DA ARTE EM INDEXAÇÃO

2.1- Introdução

Este capítulo apresenta uma descrição do estado da arte da área de indexação de dados. Inicialmente são apresentados os métodos de indexação mais usados nos gerenciadores que indexam dados convencionais com relação de ordem total. Posteriormente, são apresentadas as estruturas que lidam com outros tipos de dados não convencionais (dados espaciais e métricos).

2.2- Levantamento da Literatura de Indexação de Dados Espaciais e Métricos

A recuperação de dados armazenados em Bancos de Dado em geral é feita utilizando estruturas de indexação, que permitem fazer a recuperação dos dados muito mais rapidamente do que se a busca fosse feita seqüencialmente. No entanto, as estruturas de indexação que podem ser utilizadas dependem das propriedades dos domínios de dados indexados e do tipo de consultas que devem ser respondidas. Tradicionalmente, os gerenciadores de bancos de dados suportam bem dados de domínios que possuem a propriedade de relação de ordem total, tais como números e textos com a relação de ordem lexicográfica permitindo consultas por igualdade e consultas envolvendo relações de ordem tais como $>, \leq, >, \leq$ etc. Além disso, as estruturas de indexação comumente utilizadas em sistemas gerenciadores de bases de dados são construídas para serem armazenadas em disco, particionando o conjunto de dados em registros de tamanho fixo, para fazer uso da estrutura subjacente de memória *cache* para disco. O exemplo mais comum desse arranjo é o das árvores de indexação, quando os registros são então chamados "nós".

As árvores de indexação em geral são balanceadas. Numa árvore de indexação, toda consulta começa pela consulta ao nó raiz, e procede até os nós folhas, definindo um "caminho de busca", ou seja, a coleção dos nós desde a raiz até as folhas. Uma árvore de indexação balanceada tem todos os caminhos de busca de igual comprimento.

Um exemplo de árvore para domínios de dados com a propriedade de ordem total, como números e textos em ordem lexicográfica, é a árvore binária. Esta é uma estrutura não balanceada, mas que pode ser mantida balanceada pela aplicação de algoritmos bem conhecidos, mas complexos, para o balanceamento. Estas árvores são geralmente mantidas somente em memória, pois não existe uma maneira simples de particioná-las em blocos de tamanho fixo. Com base nessa deficiência, foi desenvolvida a família de árvores conhecidas genericamente como B-trees [Comer_1979] [Zisman_1993][Lomet_2001]. Estas árvores são balanceadas por altura e podem ser divididas em páginas armazenáveis em disco.

Embora amplamente utilizadas em SGBDs comerciais como o principal método de indexação de dados, as B-trees restringem-se a domínios de dados que têm a propriedade de relação de ordem total, e são interessantes apenas para consultas que envolvam a relação de ordem. Ou seja, as B-trees são estruturas que usam os operadores ">" e "<" para escolher a posição no nó onde o elemento será inserido na estrutura.

Aplicações mais sofisticadas freqüentemente apresentam dados em outros domínios, com outros tipos de consulta. Dados vetoriais, tais como dados geo-referenciados em duas ou três dimensões, ou mesmo espaços de características em altas dimensões, usualmente não são adequadamente indexados por *B-trees*. Para dados em domínios vetoriais foi desenvolvida uma nova classe de métodos de indexação, que são chamados de "métodos de acesso vetoriais" - MAVs ou de "métodos de acesso multi-dimensionais - MA-kDs. Esses métodos são por sua vez divididos em métodos de acesso para pontos e para regiões. Os métodos de acesso para pontos, em geral chamados de "*Point Access Methods*"- PAM, indexam os pontos do espaço, cada um representado por um valor para cada dimensão do espaço. Esses métodos aceitam consultas por topologias ou envolvendo ângulos. Exemplos bem conhecidos de PAMs são as k-d-B-tree [Robinson_1981] e os BANG-files [Freeston_1987]. Os métodos de acesso para regiões, em geral chamados de "Spatial Access Methods" – SAM, permitem representar regiões, onde consultas como sobreposição ou continência podem ser suportadas. Exemplos bem conhecidos desses métodos são a família de árvores *R-trees* [Guttman_1984] [Sellis_1987] [Beckmann_1990]. Um excelente apanhado dos métodos de acesso

vetoriais, tanto pontuais quanto espaciais, é feito em [Gaede_1998] no qual são discutidos mais de 40 diferentes estruturas de indexação para dados vetoriais.

A grande maioria dos métodos de acesso vetoriais foram desenvolvidos tendo por objetivo a armazenagem em disco. Um dos métodos que foi desenvolvido sem essa restrição é a estrutura k-d-tree [Bentley_1975], que serviu de base para a K-D-B-tree [Robinson_1981] e outras estruturas mais sofisticadas como a hB-tree [Lomet_1989] [Lomet_1990], sendo que todas estas estruturas são projetadas para armazenagem em disco. De uma certa maneira, a k-d-tree pode ser vista como a estrutura para domínios vetoriais pontuais correspondentes às árvores binárias dos domínios com relação de ordem total, enquanto a k-d-B-tree, é uma estrutura para domínios vetoriais pontuais que corresponde à B-tree nos domínios com relação de ordem total.

Uma outra estrutura importante é a *QuadTree* [korth_1999] [Gaede_1998], uma estrutura de indexação vetorial espacial muito próxima da K-D-tree para o espaço de duas dimensões. Como a K-D-Tree esta estrutura decompõe o espaço em partes com base num hiperplano. A *Quadtree* exige que o espaço seja delimitado, ou seja, cada partição divide a dimensão do espaço em duas regiões pela metade exata do espaço disponível. Cada nó da *quadtree* é associado a uma região retangular do espaço. O topo da árvore é associado a todo o espaço considerado.

Um terceiro domínio de dados que tem crescido em importância é o dos domínios métricos. Nesses domínios, além dos próprios elementos de dados, é definida uma função de similaridade (ou de distância) entre pares de elementos, e essa função é a única maneira de comparação entre dois elementos de dados do conjunto. Por exemplo, num domínio métrico não existe a noção de relação de ordem, topologia, ângulo, etc. Nesses domínios, as únicas consultas possíveis são as consultas por similaridade, das quais as consultas por vizinhos mais próximos (*k-nearest neighbors queries*) e consultas por abrangência (*range queries*) são as mais importantes [Faloutsos_1996].

Da mesma maneira que em outros domínios, várias estruturas de indexação têm sido desenvolvidas para consultas por similaridade em domínios métricos.

Os primeiro trabalho desenvolvido em domínios métricos foi feito por Burkhard e Keller [Burkhard_1973] fornecendo duas formas de particionamento de um conjunto de dados métricos. A primeira forma particiona o conjunto de dados escolhendo um representante dentre os elementos do conjunto e agrupando os demais elementos baseados nas distâncias dos mesmos para este representante escolhido. A segunda forma particiona o conjunto dos dados em um número fixo de subgrupos e escolhe-se um representante para cada um desses subgrupos. Para cada um dos subgrupos, a maior

distância do representante para um objeto do subgrupo é armazenada para ajudar nas consultas de vizinhos mais próximos. A partir deste trabalho, vários outros se originaram e são apresentados a seguir.

Em [Uhlmann_1991] e [Yianilos_1993] foi proposta a estrutura métrica conhecida como vp-tree. Esta estrutura usa uma forma similar à apresentada por [Burkhard_1973]. Os objetos são divididos em dois subconjuntos segundo um representante chamado de *vantage point*. No trabalho realizado por [Yianilos_1993] foi proposta ainda uma variação da vp-tree chamada de vps-tree na qual é mantida uma lista dos históricos das inserções feitas partindo do nó raiz, que permite diminuir o tempo para a montagem da estrutura. Uma outra estrutura proposta em [Uhlmann_1991] foi a gh-tree, que particiona os objetos do conjunto em dois grupos escolhendo dois representantes e colocando em cada um dos grupos os elementos mais próximos do representante. Diferentemente das vp-trees, para as gh-trees quando os representantes são bem escolhidos para todos os níveis, a árvore gerada tende a ser bem balanceada.

Para diminuir o número de cálculos de distâncias feitos pelas estruturas criadas até então, Baeza-Yates em [Baeza_Yates_1994] propôs uma variação para a vp-tree que usa o mesmo *vantage point* para todos os objetos que pertencem ao mesmo nível da estrutura. Desta forma, uma árvore binária degeneraria para uma lista simples de *vantage points*.

Ozsoyoglu e Bozkaya em [Bozkaya_1997] e [Bozkaya_1999] propuseram uma extensão da vp-tree chamada de mvp-tree que usa dois ou mais pontos como representantes para cada nó da árvore. Desta forma, cada nó da mvp-tree pode ser visto como dois ou mais níveis da vp-tree onde os nós filhos no nível inferior usam o mesmo *vantage point*. Isto torna possível que os nós da mvp-tree tenham um alto *fanout* e poucos representantes nos nós dos níveis não-folha.

Todas as estruturas métricas apresentadas até aqui são estruturas estáticas, ou seja, têm que ter todo o conjunto dos dados de antemão para montar a árvore de indexação, não suportando inserções posteriores. Em Ciaccia [Ciaccia_1997] foi proposta a primeira estrutura dinâmica conhecida como M-Tree. Esta estrutura é balanceada por altura e os objetos são armazenados nas folhas. Nesta estrutura o particionamento do espaço é feito com base nos objetos representantes escolhidos do conjunto de objetos que serão inseridos.

Uma outra estrutura dinâmica proposta é a Slim-Tree [Traina_2000] [Traina_2002] . Nessa estrutura foi proposta uma nova medida que permite medir qual grau de sobreposição existente entre os nós da árvore. Com esta medida é possível aplicar um algoritmo para diminuir a sobreposição, ou seja,

os objetos são redistribuídos e novos nós representantes são escolhidos.Quando a sobreposição entre os nós é menor, há uma diminuição do número de acessos a disco necessário para responder a uma consulta. Esta estrutura propõe ainda uma nova forma de divisão para o nó que atingiu o limite máximo. Esta nova forma de divisão baseia-se na construção de uma *minimal spanning tree (MST).*

Uma proposta mais recente introduz um novo conceito, o da Família Omni [Santos_2001], com o objetivo de diminuir o número de cálculos de distâncias necessários para responder as consultas. A proposta não se limita em ser uma simples estrutura de indexação, pois o conceito Omni pode ser usado com quase todas as demais estruturas já propostas na literatura e por este motivo, este trabalho originou uma nova família de estruturas de indexação.

Todas as estruturas existentes para domínios métricos apresentadas trabalham utilizando blocos de tamanho fixo, visando sua armazenagem em disco.

2.3- Considerações Finais

Neste capítulo foi apresentado o estado da arte na área de indexação de dados começando da indexação de dados convencionais, passando pela indexação de dados espaciais e por ultimo a indexação de dados métricos.

Com este levantamento da literatura, pode-se concluir que muitas estruturas de indexação vêm sendo desenvolvidas ao longo dos anos. Cada uma delas tem seus pontos fortes e suas deficiências. Desta forma, para se poder escolher uma estrutura adequada para uma certa aplicação, deve-se fazer uma análise dos dados que serão indexados, verificar como estes dados se comportam, se é um conjunto de dados estático ou sofre muitas modificações e quais os tipos de consulta pretende-se fazer sobre os dados.

A tabela 1 mostra exemplos de estruturas de indexação existentes, separando-as segundo o domínio a que se destinam e sua forma de armazenamento. Nesta tabela, vê-se claramente a ausência de uma estrutura dinâmica que seja armazenada em memória que lide com dados métricos. Para certas aplicações, a existência de uma estrutura com essas características é muito útil. Com base nesta necessidade este trabalho propõe uma estrutura que atende a estas características.

Armazenamento / Domínio		Memória	Disco
Relação de Ordem		Binária	B-Tree
Espacial	**Pontuais**	K-D-Tree	K-D-B-Tree X-Trec
		----------	----------
	Por região	QuadTree	R-Tree Sr-Tree
Métricas		**Estrutura Proposta**	M-Tree Slim-Tree

Tabela 1: Exemplo de estruturas baseando-se na forma de armazenamento.

CAPÍTULO 3

MÉTODOS DE INDEXAÇÃO MÉTRICOS

Este capítulo apresenta os conceitos mais importantes para indexação de dados em domínios métricos, e uma descrição das estruturas de indexação para dados em domínios métricos mais relevantes para o entendimento da estrutura proposta.

3.1- Introdução

As estruturas tradicionais são adequadas para dados que apresentam a relação de ordem total, tais como, números e pequenas cadeias de caracteres. Quando um conjunto de dados não apresenta esta propriedade, faz-se necessário o uso de estruturas mais complexas.

Os Métodos de acesso métricos, conhecidos como MAM (*Metric Access Methods*), representam um conjunto de estruturas usadas para indexar dados em domínios métricos. Os métodos de acesso métricos organizam os elementos de um conjunto de dados em uma estrutura de indexação, de várias formas, para que se tenha um acesso rápido e eficiente aos dados. Uma das formas é dividindo o conjunto de objetos em regiões ou nós, escolhendo estrategicamente objetos que serão os representantes para cada uma das regiões. Normalmente, nos nós são armazenados os representantes, os objetos da região considerada e as distâncias dos objetos para o representante. Os nós são organizados hierarquicamente formando estruturas em árvore [Santos_2001].

Por sua relevância para o trabalho desenvolvido, alguns métodos de indexação métricos já citados brevemente no capítulo 2, são apresentados mais detalhadamente.

3.2- Definições

Nesta seção são apresentados os conceitos básicos para a manipulação de dados em domínios métricos.

3.2.1- *Domínio Métrico e Funções Distância*

Em um domínio métrico não existem informações geométricas sobre os objetos. O que existe são somente os objetos e a informação da distância entre eles.

Definindo formalmente, seja X o universo de objetos válidos dentro da aplicação de interesse. Um domínio métrico pode ser definido como o par (X,d), onde $d(x,y) \mid x,y \in X$ é uma função que mede a dissimilaridade(ou distância métrica) entre os objetos de X.

Para ser considerada métrica, a função d deve atender as seguintes propriedades, onde x,y,z são elementos pertencentes a X:

1. Não negatividade - $\forall x,y \in X$, $d(x,y) > 0$ se $x \neq y$ e $d(x,y) = 0$ se $x=y$.
2. Simetria - $\forall x,y \in X$, $d(x,y) = d(y,x)$
3. Desigualdade Triangular - $\forall x,y,z \in X$, $d(x,y) \leq d(x,z) + d(z,y)$

Se as distâncias não são sempre estritamente positivas como em (1), o domínio é pseudométrico. Se o domínio não tem simetria como na propriedade (2), ele é chamado de domínio quasi-métrico [Chavez_2001].

Para calcular a distancia entre objetos no domínio métrico, existem várias funções métricas diferentes. A escolha da função depende da aplicação e do tipo dos objetos considerando, por exemplo imagens ou cadeias de DNA ou ainda textos sem ordem. Quando uma função métrica é definida para dados em domínio vetoriais, este pode ser considerado também um domínio métrico. As funções métricas mais comuns para esses domínios são as da família das L_p ou *Minkovsky,* e dentre elas as mais conhecidas são: L_0 – conhecida como *Infinity* ou *Chebychev*, a L_1 – conhecida como *Manhatan* e L_2 - conhecida como Euclidiana. Já para indexação de palavras a função mais comum é a função L_{edit}, que obtém a medida da distância entre as palavras contando quantas letras têm que ser trocadas, acrescentadas ou suprimidas. Formalmente a L_p é defina como:

Dados:

- d a dimensão do espaço;
- x e y dois objetos de dimensão d

$$L_p\,(x,\,y) = \sqrt[p]{\sum_{i=0}^{d-1}(x_i - y_i)^p}$$

As diferentes funções de distância originam diferentes regiões de cobertura. Por exemplo, para as funções métricas mais conhecidas da família L_p, existem as regiões mostradas na figura 1 como as coberturas possíveis para um dado objeto representante p e um raio r:

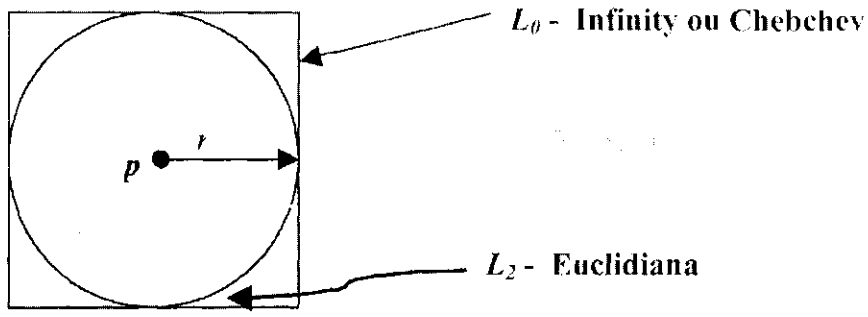


Figura 1: Áreas de cobertura para as diferentes funções de distância.

3.2.2- Consultas por Similaridade

As consultas por similaridade têm por objetivo recuperar objetos que sejam similares ou dissimilares até um tanto, de um dado objeto de consulta q. A similaridade, ou a dissimilaridade, é medida através da função de distância métrica definida entre todos os pares de objetos.

Basicamente, os dois tipos de consultas por similaridade mais usadas para dados em domínios métricos são: as consulta por abrangência (*range query*) e as consultas pelos vizinhos mais próximos *(nearest neighbors query*). A definição formal dessas consultas é apresentada a seguir.

Consultas por Abrangência

Dados:

Um domínio métrico (X,d)

$D \in X$ um conjunto de objetos,

$O = \{O_1, O_2, O_3, \ldots O_n\} \in X$,

$d()$ a função distância métrica,

$Q \in D$ um objeto de consulta e

r à distância de busca máxima $r(Q)$

Algebricamente, pode-se definir a consulta por abrangência como:

Range $(Q, r(Q)) = \{O_i | O_i \in D \text{ e } d(O_i, Q) \leq r(Q)\}$,

Ou seja, a consulta Range(Q,$r(Q)$) retorna todos os objetos Oi do conjunto D que tenham uma distância menor ou igual à distância máxima r.

Por exemplo, para um conjunto de dados que contenha as estrelas conhecidas, indicando suas coordenadas celestes, uma consulta por abrangência possível seria: encontre todas as estrelas que distem até cinco anos luz do Sol. Neste caso, o objeto central da consulta é o Sol e a distância máxima é de cinco anos luz.

Consultas por Vizinhos mais próximos

Dados:

Um domínio métrico (X,d)

$D \in X$ um conjunto de objetos,

$O = \{ O_1, O_2, O_3, \ldots O_n \} \in X$,

$d()$ a função distância métrica,

Q o objeto de consulta $\in D$ e

Um $k \geq 1$ que indica o número de vizinhos a ser retornado como resposta

Algebricamente, pode-se definir a consulta por vizinhos mais próximos como:

K-NN$(Q,k) = \{A_i | A_i \in A, A \subseteq D, |A| = k \text{ e } \forall A_i \in A, O_i \in D - A, \mathbf{d}(Q, A_i) \leq \mathbf{d}(Q,O_i)\}$,

Ou seja, K-NN(Q,k) retorna todos os k objetos mais próximos do objeto de consulta Q.

Por exemplo, ainda usando o conjunto de estrelas, uma consulta por vizinhos mais próximos possível seria: encontre as 20 estrelas mais próximas do Sol. Neste caso, o objeto de consulta ainda seria o Sol e o valor de k é 20.

Um tipo especial de consulta que pode ser realizado sobre dados em domínios métricos é verificar se determinado objeto encontra-se indexado. Essa consulta, denominada consulta de existência ou *point query*, retorna um valor lógico, indicando 'falso' se o objeto pesquisado não estiver indexado, e 'verdade' se o objeto estiver indexado. Essa consulta pode ser representada como uma consulta por abrangência com raio de busca igual a zero. Como a distância de um objeto para ele mesmo é zero, se ele estiver indexado, a busca por abrangência irá retornar um objeto (o que pode ser traduzido como uma consulta que responde 'verdade'), caso contrário nenhum objeto será retornado, (o que pode ser traduzido como uma consulta que responde 'falso').

3.3- Estruturas de Indexação Métricas

Na maioria dos bancos de dados convencionais, muitas consultas fazem referência a apenas uma pequena porção dos registros de um arquivo. O ideal neste caso seria o sistema ser capaz de localizar os registros desejados diretamente. Para permitir acesso rápido aos dados, usam-se estruturas associadas aos arquivos chamadas de índices.

Os índices, para um banco de dados, trabalham como um catálogo para um livro em uma biblioteca. Por exemplo, quando se estiver procurando um livro de um autor em particular, deve-se procurar no catalogo de autores e um cartão informará onde encontrar o livro. Para ajudar na procura, a biblioteca mantém os cartões em ordem alfabética, de forma que não seja necessário percorrer todos os cartões para se encontrar o autor procurado[Korth_1999].

Existem várias formas de organizar estruturalmente um índice para que se tenha um acesso rápido e eficiente aos dados. Uma das formas é dividindo o conjunto de objetos em regiões ou nós, escolhendo estrategicamente objetos que serão os representantes para cada uma das regiões. Normalmente, nos nós são armazenados os representantes, os objetos da região considerada e as distâncias dos objetos para o representante. Os nós são hierarquicamente organizados formando estruturas de árvores [Santos_2001].

Uma dificuldade da indexação de dados métricos é que se tem pouca informação sobre os objetos, já que apenas a função distância pode ser utilizada como um elemento de comparação entre pares de objetos.

Como apresentado no capítulo dois, várias estruturas de indexação para dados métricos foram criadas nos últimos anos com base no trabalho de Burkhard e Keller [Burkhard_1973]. A seguir são apresentadas as estruturas mais relevantes para o entendimento do trabalho desenvolvido.

A primeira estrutura apresentada é a VP-Tree, uma estrutura binária que foi usada nos testes comparativos de performance da estrutura desenvolvida neste projeto. A segunda é a GH-Tree uma estrutura que se apóia num conceito semelhante à da estrutura desenvolvida no projeto, por usar dois representantes por nó. E por ultimo é apresentada a Slim-Tree que é uma estrutura mais atual com algumas inovações importantes e também é uma estrutura dinâmica.

A seguir as estruturas são apresentadas mais detalhadamente.

VP-Tree

A *vantage point tree (VP-tree)* [Uhlmann_1991] [Yianilos_1993] é uma estrutura binária que divide o espaço em cortes esféricos, baseado num objeto, denominado *vantage point,* escolhido dentre os elementos do conjunto.

No nó de uma VP-tree são armazenados os seguintes dados:

- O objeto *vantage point* escolhido;
- O valor da mediana das distâncias calculadas entre o *vantage point* e todos os demais elementos do conjunto armazenado nessa árvore;
- Ponteiro para a sub-árvore esquerda, que contém os objetos com valor de distância menor que a mediana;
- Ponteiro para sub-árvore direita, que contêm os objetos com valor de distância maior ou igual à mediana.

```
Nome: Make_VP
Entrada: Conjunto de objetos a ser inseridos
Saída: novo nó da vp-tree

Inicio
        Se o tamanho do conjunto de objetos a serem inseridos for igual a zero:
                - Retorna um ponteiro nulo;
        Senão
                - Aloca um novo nó na árvore;
                - seleciona o vantage point dentre os objetos do conjunto de objetos a ser inserido;
                - calcula-se as distâncias de vantage point para todos os elementos do conjunto de objetos
                  que ainda tem que ser inseridos;
                        - destas distâncias, obtém-se a mediana;
                        - com base nesta mediana o conjunto de objetos é divido em duas metades:
                            - a parte que possui as distâncias maiores ou iguais à mediana(sub-árvore
                            distante);
                            - a parte que possui as distâncias menores que a mediana(sub-árvore próxima)
                -   Recursivamente executa-se o algoritmo para cada um dos dois novos conjuntos obtidos
                    na partição baseada na mediana.
Fim
```

Figura 2: Algoritmo de Construção da VP-Tree.

A construção da arvore é feita recursivamente, escolhendo-se um dos elementos p do conjunto dos objetos. Os demais elementos serão colocados como filhos da direita ou da esquerda de p baseados na mediana M das distancias com relação à p. Caso um objeto esteja a uma distância maior que M, o objeto será colocado na sub-árvore da direita, caso contrário ele será colocado na sub-árvore da esquerda. O algoritmo de construção é apresentado na figura 2.

O algoritmo para a escolha do *vantage point* é apresentado na figura 3.

```
Nome: Seleciona_vp
Entrada: Conjunto de objetos a ser inserido (S)
Saída: melhor vantage point escolhido

Inicio
  P = selecciona um conjunto de cardinalidade n de objetos escolhidos aleatoriamente
dentre os objetos do conjunto S.
Best_spread = 0
  Para todos os objetos p∈ P
        D = conjunto de cardinalidade n de objetos escolhidos aleatoriamente dentre os
          objetos do conjunto S.
        Calcular as distâncias dos elementos de D e o elemento considerado de P.
        med = Mediana de D d(p,d);
        Spread = Segundo-Momento de D (d(p,d)-med)
        Se Spread >Best_spread
              Best_spread = Spread
              Best_p = p
  Fim do para
Fim
```

Figura 3: Algoritmo de Escolha do Vantage Point

Este algoritmo ainda escolhe o *vantage point* aleatoriamente, entretanto ele faz uma amostragem estatística que escolhe o melhor *vantage point* dentre um subconjunto aleatório do conjunto de objetos.

Para se resolver uma consulta nesta arvore, calcula-se a distância do objeto de *query* q para a raiz da sub-árvore considerada. Se a distância calculada subtraída do raio de consulta for maior que M, percorre-se a sub-árvore da direita, caso contrário deve-se percorrer a sub-árvore da esquerda.

A figura 4 mostra o exemplo de um conjunto de objetos e a respectiva árvore VP-Tree gerada.

Foi proposta uma outra estrutura, semelhante a VP-Tree, a *Multi vantage point tree (MVP-Tree)* em [Bozkaya_1997] [Bozkaya_1999]. Contudo, esta estrutura cria partições considerando mais de um *vantage point* em cada um dos níveis e ainda mantém informações extras para os pontos de dados nos nós folha, para uma filtragem mais eficiente dos pontos que responderiam a uma busca por similaridade na estrutura. A MVP-Tree tem a vantagem de poder trabalhar com nós maiores que a VP-

Tree e como eles são de tamanho fixo, são interessantes para serem usados com registros de memória em disco.

É importante destacar alguns pontos a respeito da VP-Tree, que a torna uma estrutura de aplicação restrita.

Em primeiro lugar, a VP-Tree é uma estrutura balanceada, mas para garantir essa propriedade ela se torna uma estrutura estática, ou seja, ela deve ser construída quando todos os objetos já estão disponíveis. Qualquer operação de inserção posterior á sua construção não é permitida.

Em segundo lugar, sua construção é bastante custosa, uma vez que o cálculo da mediana requer o cálculo de todas as distâncias entre o objeto *vantage point* do nó e os demais objetos da sub-árvore, bem como um algoritmo de ordenação de complexidade θ (n*logn) para obter a distância mediana do conjunto. Note que as distâncias calculadas em um nó não são aproveitadas nos cálculos nas sub-árvores, uma vez que o objeto *vantage point* muda em cada nó.

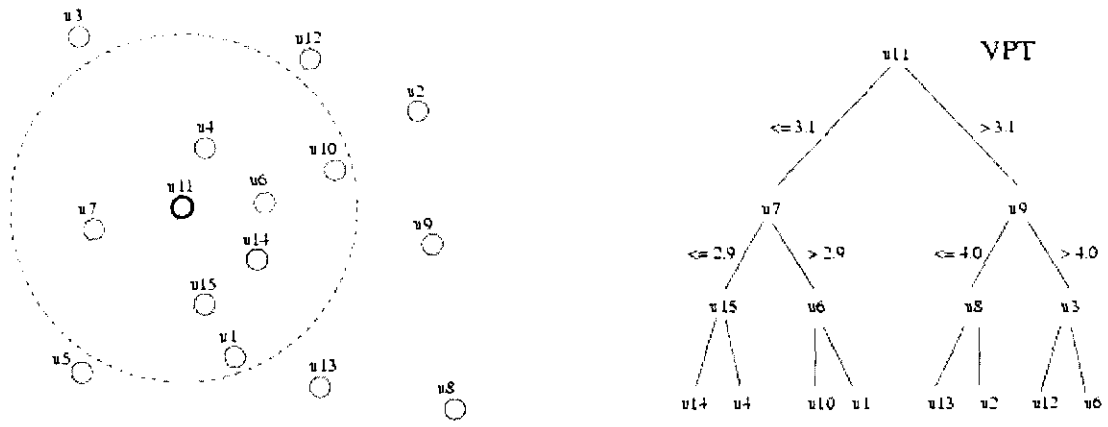


Figura 4: A estrutura da VP-Tree construída com um conjunto de 15 objetos.

GH-Tree

A GH-Tree foi uma estrutura baseada na decomposição GH proposta por Uhmann em [Uhmann_1991].

Esta estrutura tem a mesma forma de construção da *bisector tree*. Na *bisector tree* são escolhidos dois centros para cada nó. Formam-se duas sub-árvores nas quais os objetos são distribuídos. Os objetos mais próximos de um dos centros são colocados na sub-ávore esquerda e os mais próximos do

outro cento para a sub-árvore direita. Entretanto a GH-Tree usa entre os representantes um hiperplano que serve como um critério de poda no momento da execução das buscas [Chavez_2001].

O algoritmo da GH-Tree é descrito na figura 5.

Nome: Make_gh

Entrada: Conjunto de Dados a ser inserido

Saída: -

Inicio

- Em cada nó são escolhidos dois representativos (r_1 e r_2);
- Calcula-se a distância entre os dois representativos escolhidos;
- Calcula-se o ponto médio da distância entre os representativos;
- Traça-se um hiperplano entre os dois representativos passando pelo ponto médio calculado;
- Para cada elemento do conjunto que tem que ser inserido verifica-se a distância que eles estão dos representativos com relação ao hiperplano. Ou seja, os elementos que estiverem mais próximos de r_1 e, portanto a "esquerda" do hiperplano são colocados na sub-árvore esquerda. Já os elementos que estiverem mais próximos de r_2 e a "direita" do hiperplano são colocados na sub-árvore direita.

Fim

Figura 5: Algoritmo de Construção da GH-Tree

A maior limitação desta estrutura é o baixo *fanout* de cada nó, que é limitado para dois e o fato de que a estrutura tende a ser muito desbalanceada.

A figura 6 mostra um exemplo de uma GH-Tree.

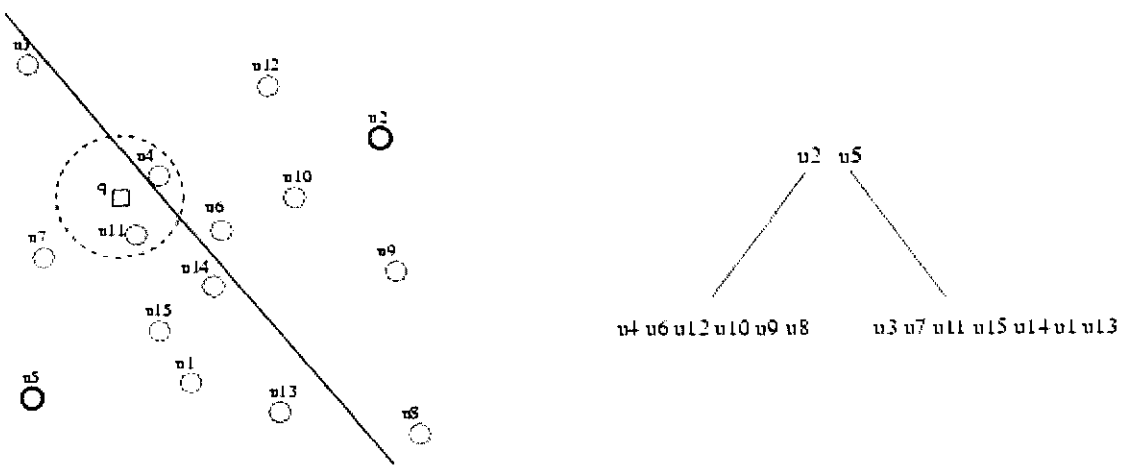


Figura 6: A GH-Tree [Chavez_2001]

Posteriormente, a GH-Tree foi estendida para uma estrutura com mais de dois centros por nó. A estrutura foi chamada de GNAT(*Geometric Near-neighbor Access Tree*), mas ainda mantendo a mesma idéia central apresentada na GH-Tree. Da mesma maneira que a MVP-Tree pode ser usada em disco, a GNAT é uma extensão da GH-Tree para uso em disco. No entanto, a construção da GNAT requer que todos os objetos estejam disponíveis quando a árvore for construída, o que a torna uma árvore também estática.

Slim Tree

A Slim-tree [Traina_2000] [Traina_2001] é uma árvore balanceada e dinâmica que cresce de baixo para cima (*bottom-up*) das folhas para a raiz. Como nas outras árvores métricas, os objetos de um grupo de dados estão agrupados em páginas do disco de tamanho fixo, cada página correspondendo a um nó. Os objetos estão armazenados nas folhas. A intenção principal é organizar os objetos em uma estrutura hicrárquica usando um representante como o centro de cada região que propicie cobertura mínima, isto é, que cobre os objetos em uma sub-árvore minimizando a sobreposição entre os nós [Traina_2001].

Os objetos são inseridos em uma Slim-tree da seguinte maneira. Começando do nó raiz, o algoritmo tenta encontrar um nó que pode cobrir um novo objeto. Se nenhum qualifica, seleciona-se o nó que tem o centro mais próximo do novo objeto. E mais que um nó qualifica, executa-se o algoritmo *ChooseSubtree* para selecionar um deles para alojar o novo objeto. Este processo é aplicado recursivamente para todos os níveis da árvore. Quando um nó *m* estoura, um novo nó *m'* é alocado para o mesmo nível, e os objetos que originalmente estavam no nó *m* mais o novo objeto são re-distribuídos entre esses dois nós. O antigo representante do no *m* é substituído pelo novo representante desse nó, e o representante do novo nó *m'* é também propagado para o nível superior, o que pode causar um estouro também no nó pai. Quando o nó raiz é dividido, uma nova raiz é alocada e a árvore cresce um nível. A Slim-tree provê quatro opções para o algoritmo de *ChooseSubtree*: escolher randomicamente um dos nós que qualificam; escolher o nó que tem a distância mínima do novo objeto e o representante (centro) do nó; ou escolher o nó que tem a mínima ocupação entre os qualificados; ou fazer a divisão do nó usando a *Minimal SpanningTtree* (MST). Neste último método de divisão, é gerada a MST dos objetos, e um dos maiores arcos da árvore MST é apagado, formando dois subconjuntos disjuntos que darão origem a dois nós finais.

Esta estrutura possui também uma forma de otimização da sobreposição das áreas de cobertura de cada representante. O algoritmo que executa essa otimização é o algoritmo de *slim-down.* Este algoritmo é solicitado manualmente na árvore. Desta forma, para se saber quando este algoritmo deve ser aplicado calcula-se o *fat factor*[2].

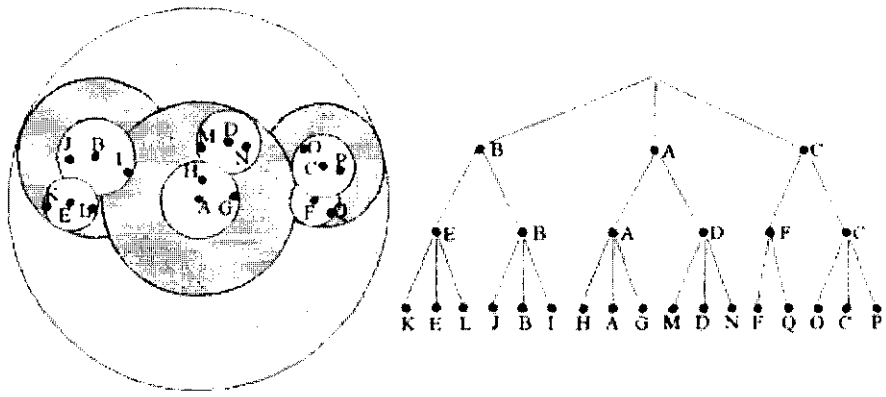


Figura 7: Exemplo de uma Sim-Tree

3.4- Considerações Finais

Neste capitulo, discutiram-se os principais conceitos relativos a domínios métricos e as estruturas de indexação para dados métricos mais importantes para o entendimento da estrutura desenvolvida durante o programa do mestrado.

As estruturas apresentadas foram a VP-Tree, a GH-Tree e a Slim Tree. As estruturas VP-Tree e Slim Tree foram detalhadas neste capítulo, por terem sido usadas para na execução dos testes comparativos da estrutura desenvolvida. Já a GH-Tree foi apresentada mais detalhadamente, por apresentar uma idéia de construção semelhante à estrutura desenvolvida.

No próximo capítulo é apresentada a estrutura MM-Tree, definida e desenvolvida neste projeto de mestrado.

---

[2] Este é um fator que mede o grau de sobreposição dos nós da árvore. Este fator varia de 0 a 1. Quanto mais próximo de 0 ele for, menos sobreposição existe entre os nós e, portanto melhor é a árvore.

CAPÍTULO 4

A MM-TREE

4.1- Introdução

Neste capítulo é apresentada a estrutura de indexação desenvolvida como resultado do trabalho do programa de mestrado. Inicialmente apresenta-se a descrição de um problema-exemplo que serviu como referência para definir as necessidades da estrutura desenvolvida, a qual atende aos quatro requisitos descritos no capítulo 1.

4.2- Descrição do Problema

Os sistemas de integração de dados na *Web* têm por objetivo responder às consultas que requeiram extração e combinação de dados a partir de múltiplas fontes dispersas na *Web*. Um sistema de integração de dados deve ser capaz de adaptar-se às mudanças que ocorrem tanto no ambiente quanto nos requisitos dos usuários, ou seja, um sistema de integração deve ser extensível para adaptar-se às novas fontes de dados, novas interfaces, protocolos e formatos. No caso dos sistemas de integração de dados para *Web,* os sistemas devem ser capazes de lidar com dados de diferentes domínios armazenados em fonte de dados que podem ter um crescimento exponencial.

Para fazer a integração de dados, várias arquiteturas são possíveis. Entretanto, a *Web* necessita da existência de uma arquitetura mais flexível e mais adequada ao tipo de informações que são integradas na *Web*. Várias alternativas estão sendo propostas usando a arquitetura cliente/servidor para troca de informação e uma abordagem *three-tier* para processamento dos dados [Salgado_2001]. Nessa

arquitetura de troca de informação, a camada de nível mais baixo é a fonte dos dados, os servidores [3]. A camada de mais alto nível é a camada do cliente onde estão as interfaces com os usuários ou aplicações que consomem dados[4]. Entre as duas camadas existem camadas intermediárias que oferecem o *middleware* para transformar e integrar os dados.

No que diz respeito à arquitetura da camada intermediária do *middleware* duas arquiteturas são possíveis: uma usando mediadores e outra usando *Data Warehouse* [Salgado_2001].

A arquitetura usando *Data Warehouse* está associada com uma abordagem materializada dos dados integrados, onde os dados são recuperados, integrados e armazenados em um repositório de dados, o Banco de Dados *Data Warehouse*. Nesta arquitetura, o problema de consulta aos dados é resolvido mais facilmente, pois recai na consulta a uma base de dados já integrada. No entanto, a dificuldade em manter essa base atualizada (bem como considerações sobre segurança de dados) tem feito dessa arquitetura uma opção menos utilizada para a integração de dados na *Web*.

Na arquitetura usando mediadores, é implementada uma abordagem virtual de integração de dados: as consultas são submetidas ao sistema de integração através do mediador, que é responsável por decompor as consultas em subconsultas, que são a seguir enviadas às fontes de dados, e integrar os resultados das subconsultas devolvendo os dados integrados para o cliente.

Este trabalho enfoca uma maneira de agilizar a manipulação dos dados que deve ser realizada por um mediador, usando uma estrutura de indexação adequada que atenda aos requisitos impostos pelas aplicações de integração de dados.

Os principais requisitos para uma estrutura de indexação para um mediador de consultas na *Web* são derivados das seguintes considerações [Faloutsos_1997] [Chidlovskii_2000]:

- Os dados têm um tempo de vida bastante curto, pois no máximo têm a duração de cada uma das páginas de dados mantidas no *cache* - quando uma das páginas que têm dados numa estrutura de indexação é removida do *cache*, toda a estrutura deve ser descartada, pois uma página pode contribuir com diversas das chaves indexadas, e a tarefa de procurar e eliminar um conjunto de chaves pode ser mais trabalhoso do que simplesmente refazer a estrutura.

---

[3] Estes servidores podem ser servidores de banco de dados tradicionais, servidores de arquivos ou qualquer outra aplicação que produza dados.

[4] As aplicações que consomem dados podem ser ferramentas para o usuário executar consultas em base de dados, ou mesmo ferramentas para exibição dos dado, como por exemplo, *browsers*.

- Embora o *cache* possa ser mantido em disco, o volume relativamente pequeno de dados, e a rapidez de resposta requerida da estrutura de índice indica que toda a estrutura de indexação deve ser mantida em memória;
- A inserção se dá preferencialmente em blocos e não uma de cada vez, já que os dados são inseridos conforme as páginas (inteiras) são recebidas dos *sites* de consulta.
- Não existe necessidade de remoção, pois quando uma página é removida do *cache*, isso não significa que o dado foi removido, mas que a informação está indisponível até que a página seja lida novamente, quando poderá ter sido modificada.
- Ausência de controle de concorrência;
- Os tipos de dados envolvidos não se restringem a números e a textos ordenáveis, mas incluem também textos comparáveis através de operações de edição de texto (p.ex: USP e USA diferem um do outro pela substituição de um *caracter*, assim como USP e SSP), e outros tipos de dados que permitem consultas por similaridade [Gaede_1995], além de consultas por igualdade, mas não necessariamente por ordem total [Böhm_2000] [Chan_2000].

Portanto, faz-se necessário pesquisar estruturas de indexação adequadas para uso em mediadores para integração de dados na *Web*, que atendam aos requisitos expressos acima, e sejam ao mesmo tempo muito eficientes para consultas por similaridade e por igualdade, para uso em operações de seleção e junção de dados armazenados em estruturas de *cache* e/ou páginas de duração efêmeras (páginas específicas de uma consulta, não colocadas no *cache*).

4.3- A MM-Tree: Memory Metric Tree

Para resolver o problema da ausência de uma estrutura de indexação dinâmica que lide com dados métricos e que seja armazenada em memória, foi desenvolvida uma estrutura de dados para domínios de dados métricos, que tem uma estrutura equivalente às propriedades que a árvore binária e a *k-d-tree* têm respectivamente para os domínios que apresentam a propriedade de ordem total e os domínios vetoriais pontuais.

A estrutura proposta é capaz de executar uma consulta de existência (*point query),* ou seja, localizar ou decidir pela inexistência do objeto na estrutura, percorrendo apenas um caminho de busca. Ela é armazenada totalmente em memória, o que dispensa a necessidade de ser particionada em blocos

de tamanho fixo. Além disso, é dinâmica, apresentando para um conjunto de N objetos inseridos, uma complexidade de construção tendendo a ordem de $\Omega(N*\log N)$ embora apresente ordem de $O(N^2)$ no pior caso, mas que, como foi constatado em experimentos práticos, raramente ocorre.

A estrutura dispõe das operações:

- Inserção individual de objetos, o que torna desnecessário que todo o conjunto de objetos seja conhecido inicialmente.
- Consulta por existência de objeto (*point-query*)
- Consultas por similaridade:
  - Consulta por k-vizinhos mais próximos
  - Consulta por abrangência

Nas seções seguintes são apresentados a estrutura dos nós e os algoritmos de inserção e consultas.

4.3.1- Estrutura da MM-Tree

A MM-Tree é uma estrutura dinâmica construída "de cima para baixo" (*top-down*). A idéia principal da MM-Tree é armazenar os objetos hierarquicamente em forma de árvore, distribuindo-os na estrutura de tal maneira que em cada nó armazenam-se dois representantes e a respectiva distância métrica entre eles, tal como é feito na GH-Tree. No entanto, de maneira distinta daquela árvore, na MM-Tree a informação da distância entre os objetos representantes origina em cada nível quatro regiões possíveis onde um objeto pode ser colocado. Para ilustrar melhor esta idéia considere um objeto "o" a ser inserido, seja S_1 e S_2 dois objetos representantes e $d()$ a função distância métrica; a posição onde o objeto "o" será inserido na estrutura é escolhido com base na distância entre os dois objetos representantes e o objeto a ser inserido:

(1) Se $d(o, S_1) \leq d(S_1, S_2)$ e $d(o, S_2) \leq d(S_1, S_2)$ o objeto será colocado na região I;

(2) Se $d(o, S_1) \leq d(S_1, S_2)$ e $d(o, S_2) > d(S_1, S_2)$ será colocado na região II ;

(3) Se $d(o, S_1) > d(S_1, S_2)$ e $d(o, S_2) \leq d(S_1, S_2)$ será colocado na região III;

(4) Se $d(o, S_1) > d(S_1, S_2)$ e $d(o, S_2) > d(S_1, S_2)$ será colocado na região IV.

A idéia para a disposição das regiões vem da tabela verdade mostrada na tabela 2, onde o valor 0 representa ≤ e o valor 1 representa o valor >.

$d(o, S_1) \leq d(S_1, S_2)$	$d(o, S_2) \leq d(S_1, S_2)$	Índice da Região
0	0	I
0	1	II
1	0	III
1	1	IV

Tabela 2: Representação Lógica das regiões de cobertura, onde 0 indica falso, e 1 indica verdade.

A figura 8 mostra como ficam graficamente dispostas as regiões usando uma função métrica L_2 para dois representantes S_1 e S_2.

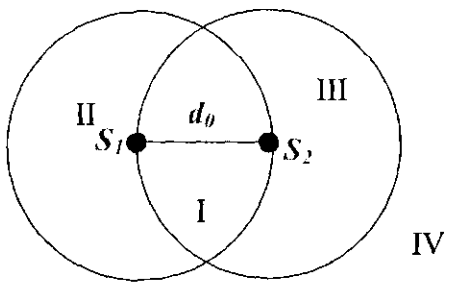
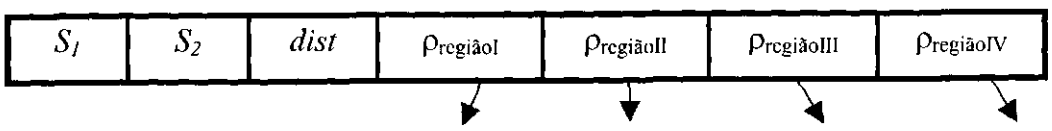


Figura 8: Ilustração das regiões de cobertura obtidas para cada nó na estrutura, usando uma função métrica L_2, sendo neste caso S_1 e S_2 os dois objetos representantes.

O nó da MM-Tree tem a seguinte estrutura:

$$Estrutura\_No[S_1, S_2, dist, \rho_{regiãoI}, \rho_{regiãoII}, \rho_{regiãoIII}, \rho_{regiãoIV}],$$

onde S_1 e S_2 são os objetos representantes, *dist* é a distância métrica calculada entre os dois objetos representantes e $\rho_{regiãoI}$, $\rho_{regiãoII}$, $\rho_{regiãoIII}$ e $\rho_{regiãoIV}$ são os ponteiros para as sub-árvores de cada uma das quatro regiões. A figura 9(a) mostra como é a estrutura física de um nó da MM-Tree, e a figura 9(b) mostra um exemplo da estrutura montada para um pequeno conjunto de palavras.



(a)

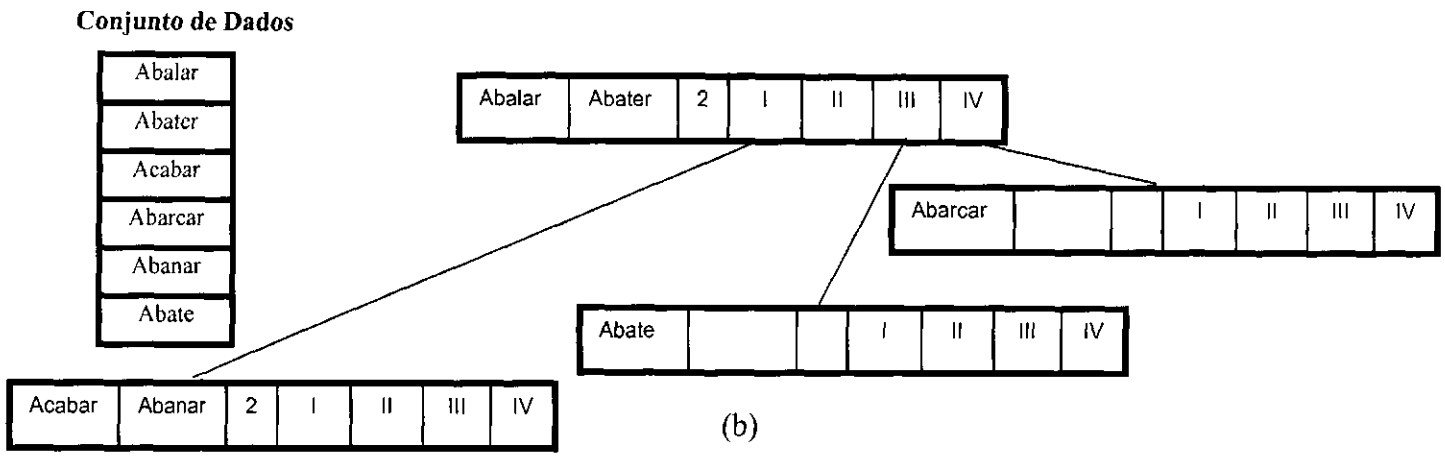


Figura 9: (a) Mostra a organização física de um nó da MM-Tree. Na parte (b) é apresentado um conjunto exemplo e como ficam dispostos os dados na MM-Tree construída.

4.3.2- Algoritmo de Construção

A árvore é construída partindo-se do nó raiz. Para cada novo objeto a ser inserido, deve-se percorrer os nós já existentes na estrutura a partir da raiz para encontrar a posição de inserção, ou seja, encontrar o nó folha onde o novo objeto deve ser inserido. Para isso, executa-se uma busca por existência do objeto. Caso o objeto seja encontrado, não é feita a nova inserção e a inserção termina. Caso contrário, como o nó onde ele deveria estar é único, ele terá sido encontrado pelo algoritmo que realiza essa busca. Esse nó pode ou não ter espaço para o novo objeto. Caso o nó esteja cheio, calcula-se a distância deste novo objeto para os objetos do nó, com base nessa distância, decide-se em qual das regiões ele deve ser colocado. Sabendo-se a região, aloca-se um novo nó e insere-se o novo objeto no novo nó alocado. Caso ainda haja espaço no nó, insere-se o novo objeto como segundo objeto no nó e calcula-se a distância entre os dois objetos. A figura 10 apresenta o algoritmo de inserção da MM-Tree.

Para um conjunto de N objetos, a construção da MM-Tree é da ordem N^2 se a árvore for completamente degenerada e $N\log N$ se a árvore estiver totalmente balanceada. A partir de testes realizados, verifica-se que a árvore tende a ser criada aproximadamente balanceada.

Para a montagem da estrutura é muito importante a ordem que os objetos são fornecidos, pois dependendo da distribuição dos objetos fornecidos a estrutura pode ser mais ou menos balanceada. Para resolver isso, poder-se-ia usar uma forma para manter um balanceamento da estrutura durante a montagem como é feito nas árvores AVL.

Um conjunto de dados que gera uma árvore balanceada irá gerar a execução da construção na menor complexidade possível, que é de $\Omega(N*\log N)$ considerando o número de cálculos de distância necessários. Um conjunto que gere uma árvore completamente degenerada será da ordem de $O(N^2)$.

Note que no pior caso, a complexidade da MM-Tree é igual a da GH-Tree. No entanto, no melhor caso, a operação de inserção em uma MM-Tree que for armazenar N objetos requer $2*\log_4 N$ cálculos de distância, enquanto a GH-Tree requer $2*\log_2 N$ cálculos de distância.Como $\log_4 N = ½* \log_2 N$, então o número de cálculos de distância necessários para criar a MM-Tree balanceada é aproximadamente a metade daqueles necessários para criar uma GH-Tree balanceada, para o mesmo número de objetos N. Assim, no caso geral, onde uma árvore não é balanceada, mas tende a não ser muito degenerada, a criação da uma MM-Tree tende a ser duas vezes mais eficiente do que a GH-Tree.

```
Nome: Insert
Entradas: novo objeto a ser inserido(ob)
          Raiz da árvore(root)
Saída: árvore atualizada.

Inicio
        Se root é nulo
                Aloca um novo nó (novo)
                novo.representante1 ← ob ( copia o novo objeto como primeiro representativo do nó
                root = novo (atualiza o novo nó como raiz para a árvore)
        Senão
                pos_ins(root,ob, Pos)
                Se o número de elementos de Pos = 1 (ainda tem espaço no nó)
                        Pos. representante2 = ob (Insere o novo objeto como segundo representativo do nó)
                        Pos.dist = distancia entre os dois representativos do nó.
                Senão
                        Aloca um novo nó(novo)
                        novo. representante1 ← ob
                        D = cálculo de d(ob, representante1) do nó Pos.
                        D1 = cálculo de d(ob , representante2) do nó Pos.
                        Se D <= Pos.dist ^ D1<=Pos.dist
                                Pos.p_regiãoI = novo ( coloca o novo nó como filho para a regão1)

                        Senão
                                Se D<= Pos.dist ^ D1 > Pos.dist
                                        Pos.p_regiãoII = novo
                                Senão
                                        Se D> Pos.dist ^ D1<=Pos.dist
                                                Pos.p_regiãoIII = novo
                                        Senão
                                                Se D > Pos.dist ^ D1>Pos.dist
                                                        Pos.p_regiãoIV = novo
Fim
```

Figura 10: Algoritmo de Inserção da MM-Tree

```
Nome: pos_ ins
Entradas: objeto de consulta(q)
          Raiz da árove(root)
Saída: o nó onde o objeto será inreiro.(Pos)
       True e False indicando se encontrou ou não o objeto.

Inicio
        Pos = root
        Se a raiz é nula
                Retorna false
        Senão
                Se root tem apenas um representante ( não tem filhos)
                        Compara q e o representante1
                        Se q e o representante1 são iguais
                                Retorna true
                        Senão
                                Retorna false
                Senão  (supondo que o  nó tem os dois representantes)
                        Compara q e o representante1
                        Se q e o representante1 são iguais
                                Retorna true
                        Senão
                                Compara q e o representante2
                                Se q e o representante2 são iguais
                                        Retorna true
                                Senão
                                        Verifica qual das regiões tem que ser percorrida
                                        Retorna pos_ins(ob, ρregião_escolhis,Pos)
Fim
```

Figura 11: Algoritmo usando para encontrar na árvore a posição de inserção para um novo objeto. E verificar se o objeto já existe na árvore.

O algoritmo apresentado na figura 10 é a inserção individual de objetos. Por isso, para executar a construção da estrutura é necessário que o algoritmo seja repetido para todos os objetos do conjunto.

O algoritmo usado para encontrar a posição de inserção, apresentado na figura 11, tem o praticamente o mesmo comportamento da *point query*. Entretanto, ele retorna mais uma informação, o nó onde o objeto será inserido.

4.3.3- Mecanismos de Consultas

Foram implementados os três principais tipos de consultas possíveis em domínios métricos, para a árvore MM-Tree: a consulta pela existência de um objeto (*point query*) e as duas consultas por similaridade (consulta por abrangência e por k-vizinhos mais próximos).

Consulta por Existência (*Point Query*)

A consulta existência tem por objetivo verificar se um objeto existe ou não no conjunto de dados armazenado na MM-Tree. Para executar esta consulta, um único caminho de busca é percorrido na estrutura, pois a partir da raiz, para cada nível da árvore, apenas uma das regiões tem que ser verificada até que o objeto seja encontrado ou uma folha seja atingida e se conclua que o objeto está ou não na estrutura.

O algoritmo para responder a uma consulta por existência é apresentado na figura 12.

```
Nome: Point_query
Entradas: objeto de consulta(q)
          Raiz da árove(root)
Saída: booleana (se encontrou – true, se não encontrou – false)

Inicio
        Se a root é nula
                Retorna false
        Senão
                Se  root tem apenas um representante (não tem filhos)
                        Compara q e o representante1
                        Se q e o representante1 são iguais
                                Retorna true
                        Senão
                                Retorna false
                Senão
                        Compara q e o representante1
                        Se q e o representante1 são iguais
                                Retorna true
                        Senão
                                Compara q e o representante2
                                Se q e o representante2 são iguais
                                        Retorna true
                                Senão
                                        Verifica qual das regiões tem que ser percorrida
                                        Retorna point_query (q, ρ região_escohida)
Fim
```

Figura 12: Algoritmo da consulta por existência

Consulta por Similaridade

As duas formas de consulta por similaridade desenvolvidas são: as consultas por abrangência e as consultas por k-vizinhos mais próximos.

Consulta por Abrangência

As consultas por abrangência são executadas percorrendo os nós da estrutura que interceptem a região de cobertura definida para a consulta formando um conjunto resposta com os objetos que estejam dentro do raio de consulta. Assim, para cada nó a seguinte verificação é efetuada.

Dados:

- S_1 e S_2 os representantes para o nó
- d_0 é a distância entre S_1 e S_2, já armazenada no nó
- Objeto de consulta q
- $d_1 = d(q,S_1)$ e $d_2 = d(q,S_2)$
- r o raio da consulta sendo realizada.

Então, as regras usadas para saber quais as regiões que deverão ser percorridas para responder a consulta por abrangência são apresentadas a seguir.

1. Região I é percorrida se: $|d_1-r| <= d \wedge |d_2-r| <= d$
2. Região II é percorrida se: $|d_1-r| <= d \wedge d_2+r > d$
3. Região III é percorrida se: $d_1+r > d \wedge |d_2-r| <= d$
4. Região IV é percorrida se: $d_1+r > d \wedge d_2+r > d$

Note-se que embora a consulta por existência possa realizar a navegação da árvore através de uma descida em profundidade (percorrendo apenas um nó em cada nível até atingir uma folha), as buscas por abrangência e por vizinhos mais próximos podem obrigar a busca em mais de uma região. Assim, qualquer uma das quatro regras acima, incluindo todas elas, pode resultar verdade, fazendo com que cada sub-árvore correspondente seja percorrida. A figura 13 mostra graficamente um exemplo para as regras que foram apresentadas.

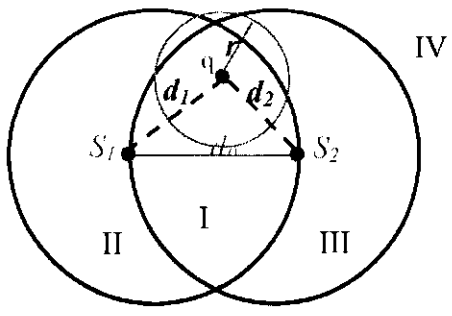


Figura 13: Exemplo gráfico das regras das regiões a serem percorridas, para uma busca por abrangência.

O algoritmo para a busca por abrangência é apresentado na figura 14.

```
Nome: Range_query
Entradas: objeto de consulta (q)
          Raio de consulta(r)
          Raiz da arvore(root)
Saída: Numero de objetos que respondem a consulta

Inicio
        Nresp ← 0
        Se root é não nulo
        Se é um nó com dois objetos
                D1 = cálculo de d(q,S1)
                Se D1<= r
                        O objeto S1 é parte da resposta.
                        Nresp ← Nresp+1
                D2 = cálculo de d(q, S2)
                Se D2 <= r
                        O objeto S2 é parte do conjunto resposta
                        Nresp ← Nresp+1

                Verificam-se quais regiões devem ser percorridas

                Se |D1-r| <= root.dist ^ |D2-r| <= root.dist
                        Range_query(q,r,root.p_região1)
                Se |D1-r| <= root.dist ^ D2+r > root.dist
                        Range_query(q,r,root.p_região11)
                Se  D1+r >root.dist ^ |D2-r |<=root.dist
                        Range_query(q,r,root.p_região111)
                Se D1+r > root.dist ^ D2+r > root.dist
                        Range_query(q,r,root.p_regiãoIV)
        Senão
                D1 = cálculo de d(q, S1)
                Se D1<= r
                        O ojbeto S1 é parte da resposta.
                        Nresp ← Nresp+1
        Retorna (Nresp)
Fim
```

Figura 14: Algoritmo da consulta por Abrangência.

Consulta por k-Vizinhos mais Próximos

Na MM-Tree as consultas por vizinhos mais próximos são feitas por lista global, ou seja, conforme os níveis da arvore são percorridos, um único conjunto de reposta é mantido até quc se chegue no conjunto final de respostas. Em cada nó, verificam-se quais regiões estão interceptando o

raio de consulta atual. Este raio é inicialmente definido como infinito, e sempre que uma quantidade k de objetos é encontrada, o raio atual é reduzido para a distância ao objeto mais distante desses k objetos. Esse raio é usado para podar a neccssidade de percorrer sub-árvores que já estão fora da região coberta por esse raio nas comparações subseqüente.

O algoritmo para obtenção dos vizinhos mais próximos é apresentado na figura 15. Na parte (a) é apresentado o algoritmo para execução da consulta por k-vizinhos mais próximos e na parte (b) é apresentado o algoritmo usado na pesquisa de cada nó verificando se os objetos do nó fazem parte do conjunto resposta e ainda atualiza o raio de consulta atual com base no maior raio de consulta presente no conjunto de respostas atual.

Como o raio de consulta atual diminui, os objetos que atendem às condições de navegação num dado estágio do algoritmo podem deixar de atendê-los num estágio posterior, quando o raio de consulta atual terá diminuído. Portanto, é importante a ordem que as regiões são navegadas. Por isso, quando mais de uma região atender às condições de navegação em um nó, a ordem de execução deve estabelecer que a primeira região a ser navegada é aquela onde o objeto central da consulta está. Isso causa uma busca em profundidade da árvore, equivalente à *point query*, sendo que todos os objetos encontrados que satisfazem as condições de navegação, tanto nos nós intermediários quanto nos nós folha, vão sendo incluídos na lista de respostas, sempre que um nó é atingido pela primeira vez. Desta maneira acelera-se o processo de redução do raio de consulta atual.

Como um dos pontos importantes da MM-Tree é a sua simplicidade de implementação, definimos quc no retorno da navegação de uma sub-árvore, qualquer sub-árvore que não tenha sido percorrida pode ser escolhida em qualquer ordem. Caso seja importante obter o máximo dc desempenho nas consultas de vizinhos mais próximos, a ordem que as regiões devem ser percorridas obedece às condiçõcs mostradas no algoritmo da figura 16. Entretanto, este algoritmo só indica uma possível ordem para se percorrer as regiões e não quais as regiões devem ser percorridas. Desta forma, para se saber se uma região deve ou não ser percorrida usa-se as regras de comparação das distâncias já apresentadas anteriormente.

Nome: knn
Entradas: Objeto de consulta(q)
Número de Vizinhos(k)
Raiz da Árvore (*root*)
Saídas: conjunto de respostas de tamanho k(*resp*)

Inicio
Se *root* é não nulo
rmax = Encontra_resp(*root,q, resp*)
Verifica usando as mesmas regras da *range query*, quais regiões tem que entrar baseando-se no raio atual *rmax*.
Knn (q,k,$root.p_{regiões\_de\_intersecção}$)
Fim

Figura 15: (a) Algoritmo da consulta por vizinhos mais próximos

Nome: Encontra_resp
Entradas: Nó atual da árvore a ser pesquisado (*at*)
Objeto de consulta(q)

Saída: raio de consulta atual modificado(*Rmax*)
Conjunto de resposta atualizado(*resp*)

Inicio
Se o número de objetos no nó = dois
D1 = cálculo de d(q,S_1)
Se *D1* <= *Rmax*
Copia S_1 para o conjunto resposta.
Ordena o conjunto resposta pelo raio.
D2 = cálculo de d(q, S_2)
Se *D2* <= *Rmax*
Copia S_2 para o conjunto resposta.
Ordena o conjunto resposta pelo raio.
Rmax ← Maior raio do conjunto resposta.
Senão
D1 = distância de q para S_1
Se *D1* <= *Rmax*
Copia S_1 para o conjunto resposta.
Ordena o conjunto resposta pelo raio.
Rmax ← Maior raio do conjunto resposta.
Retorna *Rmax*
Fim

Figura 15: (b) Algoritmo para montagem do conjunto resposta e atualização do raio de consulta máximo.

Nome: MaxNN
Entradas: Nó atual(*at*)
Objeto e Query(*q*)
Saídas: sequência para testar qual das regiões que deverão ser percorridas
Inicio
Se tem dois elementos no nó
D1 = distância de q para S_1
D2 = distância de q para S_2
Se *at.dist-D1* < at.dist-*D2*
Teste se entra região III
Teste se entra região II
Teste se entra região IV
Senão
Teste se entra região II
Teste se entra região III
Teste se entra região IV
Se *D2-at.dist* < *D1*
Teste se entra região I
Teste se entra região IV
Teste se entra região III
Senão
Teste se entra região IV
Teste se entra região I
Teste se entra região III
Se *D1- at.dist* <*D2*
Teste se entra região I
Teste se entra região IV
Teste se entra região II
Senão
Teste se entra região IV
Teste se entra região I
Teste se entra região II
Se *D1-at.dist* < *D2-at.dist*
Teste se entra região II
Teste se entra região I
Teste se entra região III
Senão
Teste se entra região III
Teste se entra região I
Teste se entra região II

Fim

Figura 16: Algoritmo de Otimização para o algoritmo de consulta dos vizinhos mais próximos.

4.4- Considerações Finais

Neste capítulo foi apresentada a estrutura métrica MM-Tree. Esta estrutura foi proposta para atender algumas características que ainda não estavam presentes em estruturas existentes na literatura. Dentre as características ainda não implementadas em outras estruturas métricas estão:

- Busca por existência percorrendo um único caminho de busca
- Estrutura dinâmica com armazenamento em memória principal

O próximo capitulo aprcsenta os resultados empíricos obtidos nos testes individuais e comparativos com outras duas estruturas métricas já existentes na literatura.

CAPÍTULO 5

RESULTADOS OBTIDOS

5.1- Introdução

Neste capítulo são apresentados os conjuntos de dados usados para coleta das medidas de desempenho, os principais resultados experimentais individuais da estrutura e mais os resultados comparativos feitos usando a MM-Trec, a VP-Tree e a Slim Tree.

5.2- Conjuntos de Dados

Uma estrutura métrica deve ser capaz de lidar bem com vários tipos de dados de diferentes dimensões. Por isso, a MM-Tree foi testada usando vários conjuntos de dados reais, cada um deles apresentando diferentes características e diferentes dimensões. Para cada um dos conjuntos de dados foi usada uma função métrica adequada.

Para a verificação da eficiência nas consultas, foi executada uma coleta de dados para medir o comportamento da estrutura com os diferentes conjuntos de objetos avaliando cada consulta através de várias requisições em que os parâmetros permanecem constantes, muda-se o objeto central da consulta e a seguir obtém-se a média de cada conjunto. As medidas coletadas são: a quantidade de distâncias calculadas para executar uma determinada consulta e o tempo total que uma consulta leva para ser executada. Uma outra medida usual, mas que não pode ser considerada para esta estrutura é a quantidade de acessos a disco, já que a estrutura proposta neste trabalho é totalmente armazenada em memória principal.

A seguir são apresentados os principais conjuntos de dados usados e suas principais características.

Eigenfaces

É uma coleção de 11900 vetores de dimensão 16 extraídos de fotos de faces humanas. Este conjunto de dados foi obtido do projeto *Informedia* da Universidade Carnegie Mellon.

PortugueseSWords

É um conjunto de 21473 palavras retiradas do dicionário da Língua Portuguesa. Os objetos deste conjunto não têm dimensão.

Conjunto Sintético

É um conjunto de 100000 objetos sintéticos representando partes no cubo unitário tridimensional.

Cities

É um conjunto de 2755 objetos com os dados da localização (latitude e longitude) das cidades brasileiras. Estes objetos têm dimensão dois.

5.3- Resultados Experimentais da MM-Tree

Nesta seção são apresentados em forma de gráficos os resultados experimentais da MM-Tree para cada um dos conjuntos de dados apresentados na seção anterior. A MM-Tree foi implementada em Borland© C++ rodando na plataforma Windows 2000. Os experimentos foram executados num PC Pentium III 850MHz com 256MB de memória principal.

Para cada conjunto de dados foram feitos dois tipos de experimentos. No primeiro experimento usou-se todo o conjunto de dados na construção da árvore e para execução das consultas foram gerados objetos aleatórios. Esse experimento considera, portanto, que os objetos centrais das consultas têm distribuição uniforme. No segundo experimento nem todos os objetos do conjunto de dados foram usados na construção da árvore. Os objetos não inseridos foram usados para a execução das consultas. Para este segundo experimento, os objetos retirados foram escolhidos randomicamente dentre os

objetos do conjunto de dados que deveria ter sido inserido. Esse experimento considera, portanto, o caso em que as consultas são polarizadas para os dados armazenados na base de dados.

A seguir são apresentados os resultados individuais obtidos para cada um dos conjuntos e as informações de escalabilidade para a estrutura.

5.3.1- Medidas de construção e consultas nos conjuntos de dados Reais

Nesta seção mostra-se o comportamento para a inserção e consulta dos três conjuntos de dados de teste reais: Eigenfaces, PortugueseSWords e cities. Para cada conjunto foram feitos dois conjuntos de experimentos: o primeiro considera consultas uniformemente distribuídas e o segundo consultas polarizadas. Para cada conjunto mostra-se o tempo total de montagem da estrutura, o número de distâncias calculadas e as alturas mínimas e máximas obtidas em cada árvore. Um total de seis árvores foram construídas para obtenção das medidas: duas para cada conjuntos de dados, sendo uma árvore para cada experimento 1 e 2.

Eigenfaces

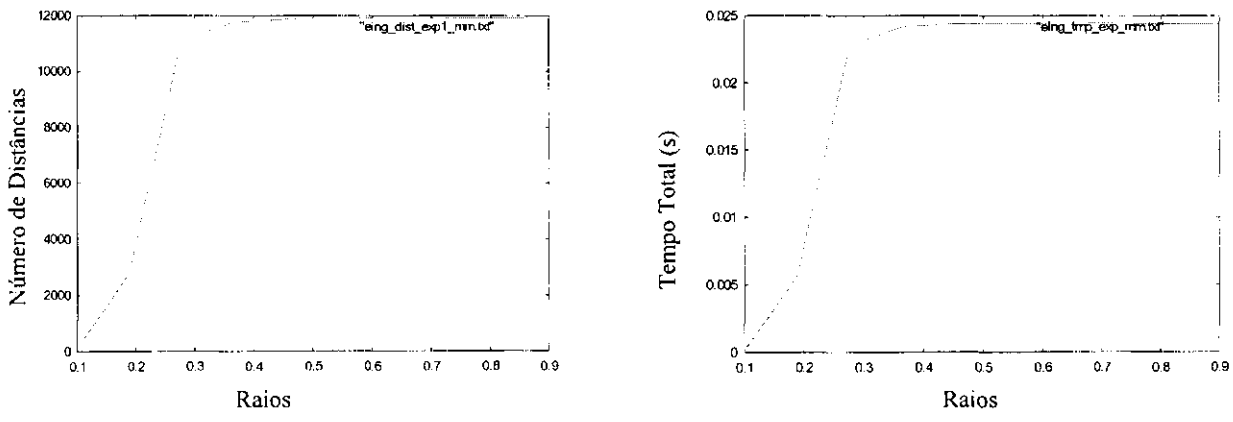
A tabela 3 mostra os parâmetros usados para construir as duas árvores do conjunto Eigenfaces, bem como as medidas de tempo total, número de cálculos de distâncias e as alturas mínima e máxima obtidas em cada experimento.

Experimento 1	*Experimento 2*
Número de Objetos Inseridos: 11900	Número de Objetos Inseridos: 11400
Número de Objetos não inseridos: 0	Número de Objetos não inseridos: 500
Dimensão: 16	Dimensão: 16
Métrica: L_0	Métrica: L_0
Tempo Total de Montagem da Estrutura: 14.86 segundos.	Tempo Total de Montagem da Estrutura: 12.83 segundos.
Número de Distâncias Calculadas para a Inserção: 235818	Numero de Distâncias Calculadas para a Inserção: 221768
Altura máxima: 21	Altura Máxima: 20
Altura Mínima: 8	Altura Mínima: 8

Tabela 3: Dados e medidas da inserção para os dois experimentos usando o conjunto eigenfaces.

Consulta por Abrangência

A figura 17(a) mostra a média de números de cálculos de distâncias por consulta e a figura 17(b) mostra o tempo total para responder as quinhentas consultas por abrangência uniformemente distribuídas sobre o conjunto Eigenfaces. Essas consultas foram geradas variando o raio de consulta desde 10% até 90% do diâmetro do conjunto. Note-se que são processadas quinhentas consultas para cada ponto de dado em cada gráfico.



(a) (b)

Figura 17: (a) Gráfico do Número de Distâncias X Valores Possíveis de Raio. (b) Gráfico do Tempo Total (em segundos) X Valores de Raio usados na consulta, usando o conjunto de dados Eigenfaces para o experimento 1.

A figura 18(a) mostra a média de números de cálculos de distâncias por consulta e a figura 18(b) mostra o tempo total para responder as quinhentas consultas por abrangência com distribuição polarizada pelos objetos do conjunto Eigenfaces. Essas consultas foram geradas variando o raio de consulta desde 10% até 90% do diâmetro do conjunto.

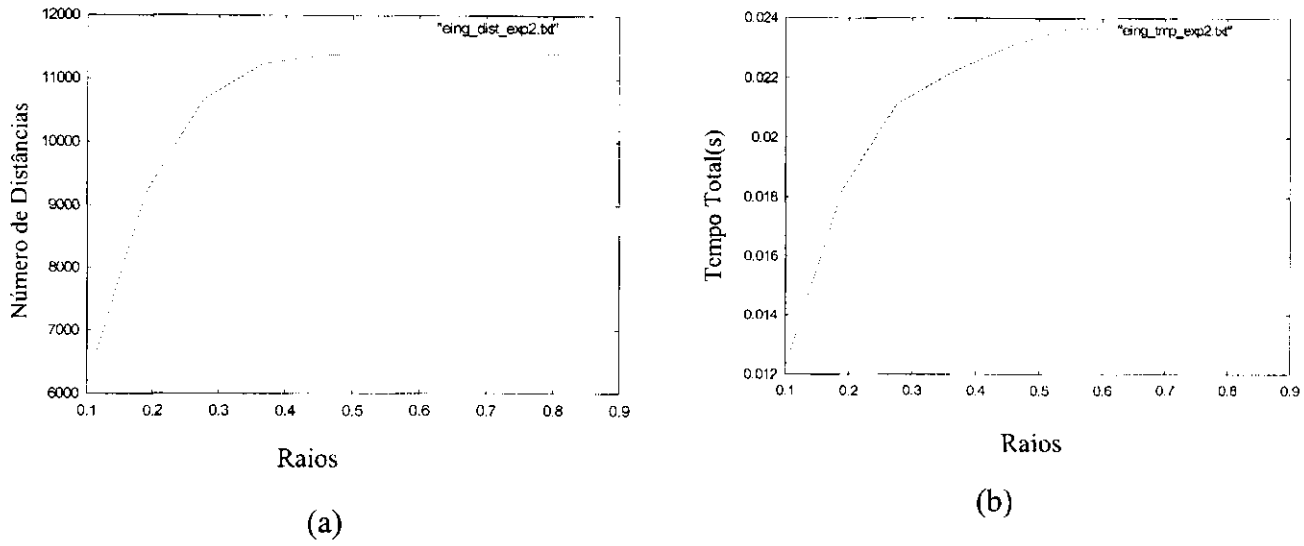


Figura 18: (a) Gráfico do Número de Distâncias X Valores Possíveis de Raio. (b) Gráfico do Tempo Total (em segundos) X Valores de Raio usados na consulta, usando o conjunto de dados Eigenfaces para o experimento 2.

Consulta por k-vizinhos mais próximos

A figura 19(a) mostra a média de números de cálculos de distâncias por consulta e a figura 19(b) mostra o tempo total para responder as quinhentas consultas por k-vizinhos mais próximos uniformemente distribuídas sobre o conjunto Eigenfaces. Essas consultas foram geradas variando o valor de k de 1% até 10% do total de objetos do conjunto indexado.

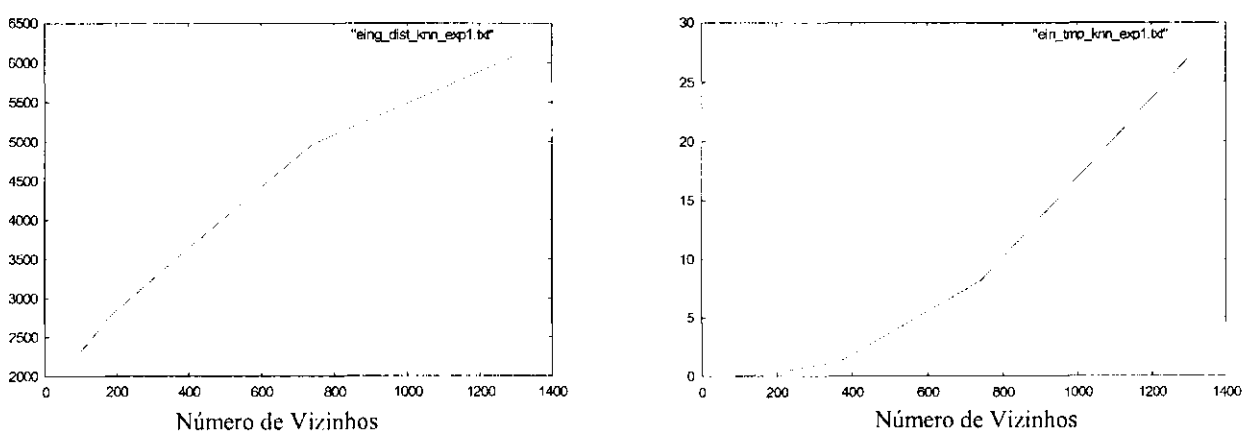


Figura 19: (a) Gráfico do Número de Distâncias X Pela Quantia de Vizinhos. (b) Gráfico do Tempo Total (em segundos) X Pela Quantia de Vizinhos, usando o conjunto de dados Eigenfaces para o experimento 1.

A figura 20(a) mostra a média de números de cálculos de distâncias por consulta e a figura 20(b) mostra o tempo total para responder as quinhentas consultas por k-vizinhos mais próximos com distribuição polarizada pelos objetos do conjunto Eigenfaces. Essas consultas foram geradas variando o valor de k de 1% até 10% do total de objetos do conjunto indexado.

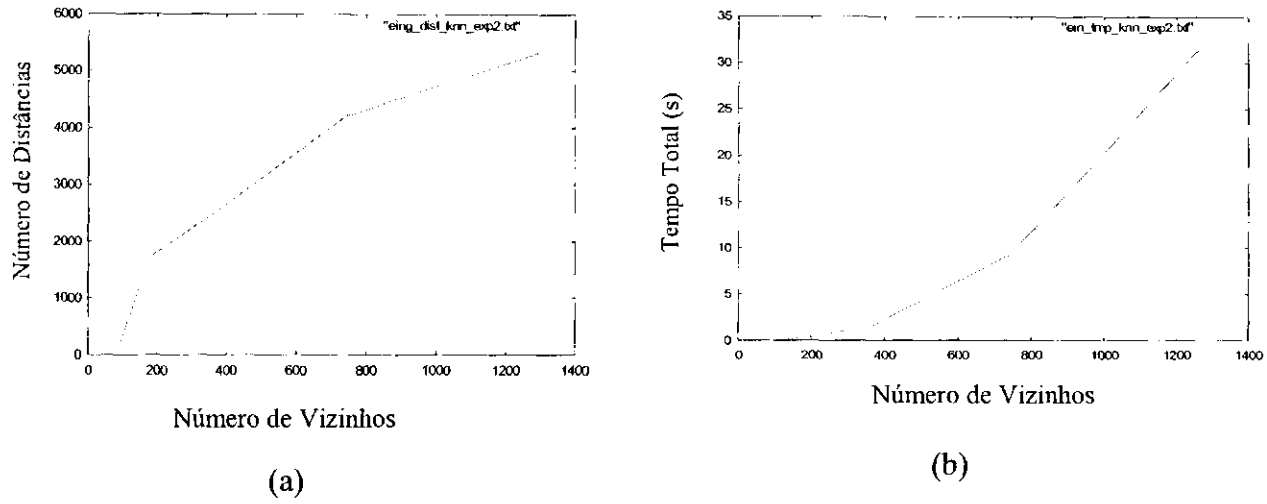


Figura 20: (a) Gráfico do Número de Distâncias X Pela Quantia de Vizinhos. (b) Gráfico do Tempo Total (em segundos) X Pela Quantia de Vizinhos, usando o conjunto de dados Eigenfaces para o experimento 2.

PortugueseSWords

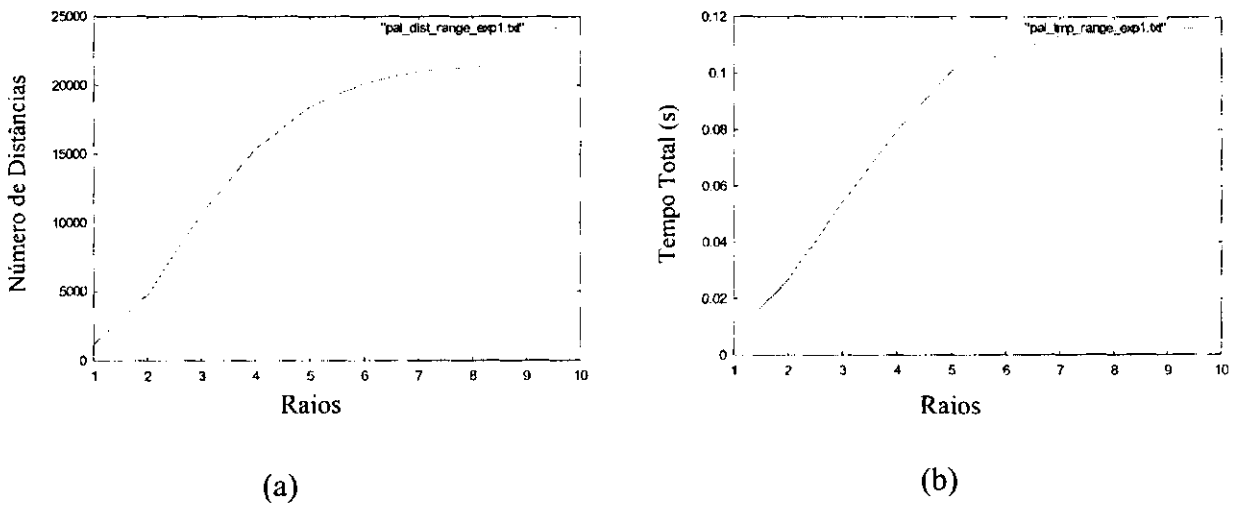
A tabela 4 mostra os parâmetros usados para construir as duas árvores do conjunto de palavras em português, bem como as medidas de tempo total, número de cálculos de distâncias e as alturas mínima e máxima obtidas em cada experimento.

Experimento 1	***Experimento 2***
Número de Objetos Inseridos: 21473	Número de Objetos Inseridos: 20973
Número de Objetos não inseridos: 0	Número de Objetos não inseridos: 500
Dimensão: -	Dimensão: -
Métrica: L_{edit}	Métrica: L_{edit}
Tempo Total de Montagem da Estrutura: 63.73 segundos	Tempo Total de Montagem da Estrutura: 60.42 segundos.
Número de Distâncias Calculadas para a Inserção: 673124.	Numero de Distâncias Calculadas para a Inserção: 635468.
Altura máxima: 30	Altura Máxima: 27
Altura Mínima: 0	Altura Mínima: 0

Tabela 4: Dados e medidas da inserção para os dois experimentos usando o conjunto PortugueseSWords.

Consulta por Abrangência

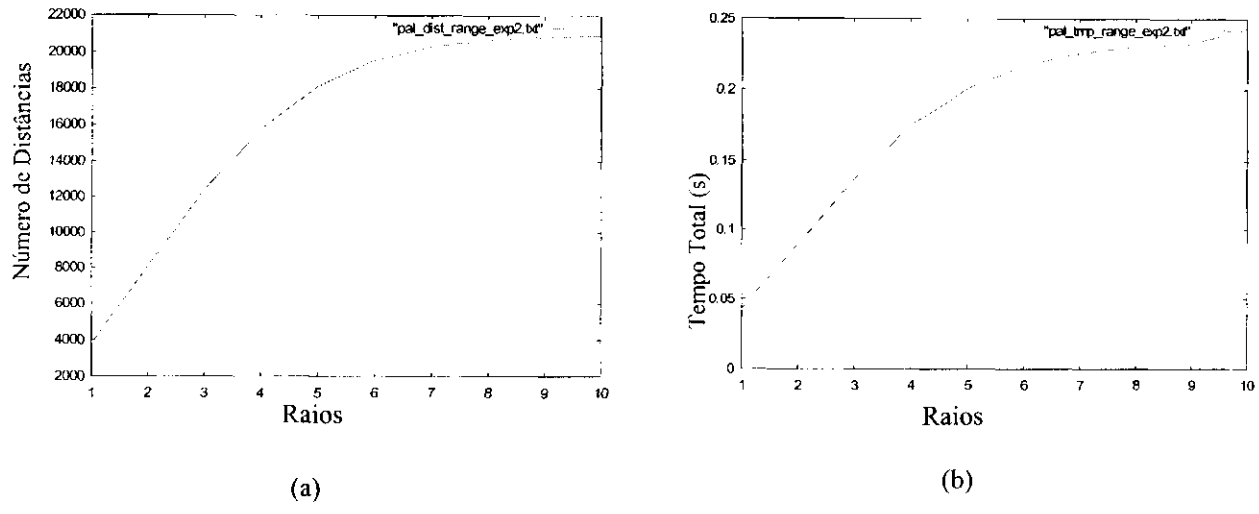
A figura 21(a) mostra a média dc números de cálculos de distâncias por consulta e a figura 21(b) mostra o tempo total para responder as quinhentas consultas por abrangência uniformemente distribuídas sobre o conjunto PortugueseSWords. Essas consultas foram geradas variando o raio de consulta desde 1 até 10 caracteres inseridos/removidos/substituídos para localizar as respostas.



(a) (b)

Figura 21: (a) Gráfico do Número de Distâncias X Valores Possíveis de Raio. (b) Gráfico do Tempo Total (em segundos) X Valores de Raio usados na consulta, usando o conjunto de dados PortugueseSWords para o experimento 1.

A figura 22(a) mostra a média de números de cálculos de distâncias por consulta e a figura 22(b) mostra o tempo total para responder as quinhentas consultas por abrangência com distribuição polarizada pelos objetos do conjunto PortugueseSWords. Essas consultas foram geradas variando o raio de consulta desde 1 até 10 caracteres inseridos/removidos/substituídos para localizar as respostas.

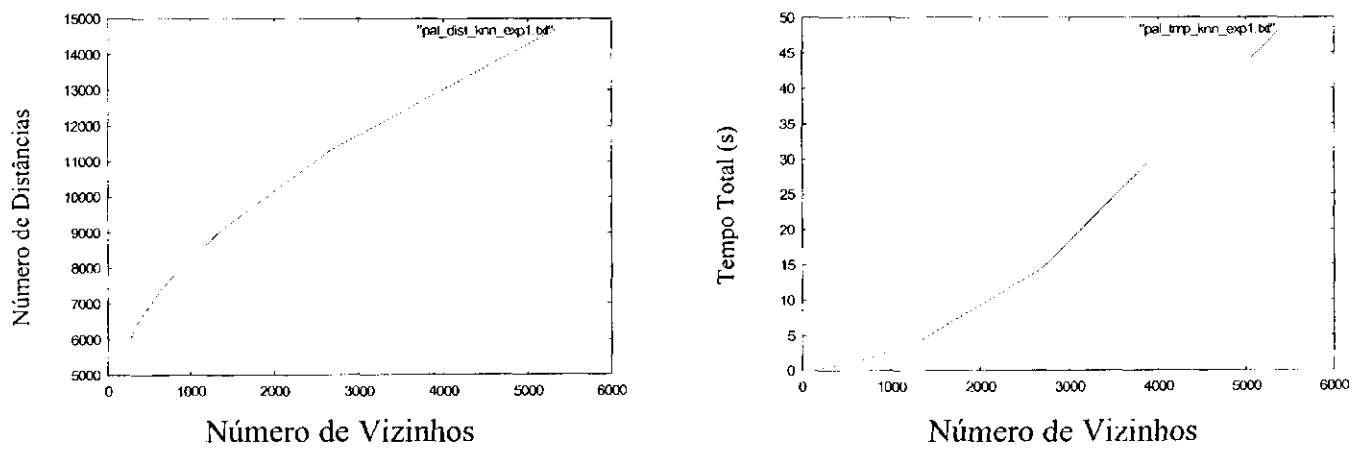


(a) (b)

Figura 22: (a) Gráfico do Número de Distâncias X Valores Possíveis de Raio. (b) Gráfico do Tempo Total (em segundos) X Valores de Raio usados na consulta, usando o conjunto de dados PortugueseSWords para o experimento 2.

Consulta por k-vizinhos mais próximos

A figura 23(a) mostra a média de números de cálculos de distâncias por consulta e a figura 23(b) mostra o tempo total para responder as quinhentas consultas por k-vizinhos mais próximos uniformemente distribuídas sobre o conjunto PortugueseSWords. Essas consultas foram geradas variando o valor de k de 1% até 10% do total de objetos do conjunto indexado.



(a) (b)

Figura 23: (a) Gráfico do Número de Distâncias X Pela Quantia de Vizinhos. (b) Gráfico do Tempo Total (em segundos) X Pela Quantia de Vizinhos, usando o conjunto de dados PortugueseSWords para o experimento 1.

A figura 24(a) mostra a média de números de cálculos de distâncias por consulta e a figura 24(b) mostra o tempo total para responder as quinhentas consultas por k-vizinhos mais próximos com distribuição polarizada pelos objetos do conjunto PortugueseSWords. Essas consultas foram geradas variando o valor de k de 1% até 10% do total de objetos do conjunto indexado.

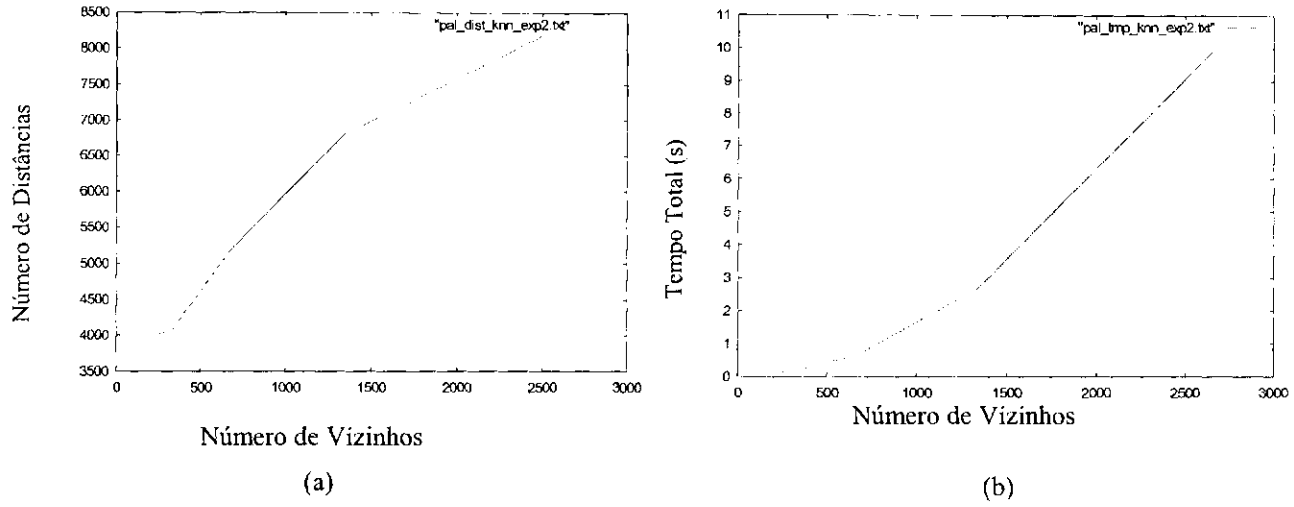


Figura 24: (a) Gráfico do Número de Distâncias X Pela Quantia de Vizinhos. (b) Gráfico do Tempo Total (em segundos) X Pela Quantia de Vizinhos, usando o conjunto de dados PortugueseSWords para o experimento 2.

Cities

A tabela 5 mostra os parâmetros usados para construir as duas árvores do conjunto cities, bem como as medidas de tempo total, número de cálculos de distâncias e as alturas mínima e máxima obtidas em cada experimento.

Experimento 1	***Experimento 2***
Número de Objetos Inseridos: 2755	Número de Objetos Inseridos: 2255
Número de Objetos não inseridos: 0	Número de Objetos não inseridos: 500
Dimensão: 2	Dimensão: 2
Métrica: L_2	Métrica: L_2
Tempo Total de Montagem da Estrutura: 0.53 segundos	Tempo Total de Montagem da Estrutura: 0.35 segundos.
Número de Distâncias Calculadas para a Inserção: 39682	Numero de Distâncias Calculadas para a Inserção: 30220
Altura máxima: 15	Altura Máxima: 12
Altura Mínima: 10	Altura Mínima: 9

Tabela 5: Dados e medidas de inserção para os dois experimentos usando o conjunto cities.

Consulta por Abrangência

A figura 25(a) mostra a média de números de cálculos de distâncias por consulta e a figura 25(b) mostra o tempo total para responder as quinhentas consultas por abrangência uniformemente distribuídas sobre o conjunto Cities. Essas consultas foram geradas variando o raio de consulta desde 5% até 90% do diâmetro do conjunto.

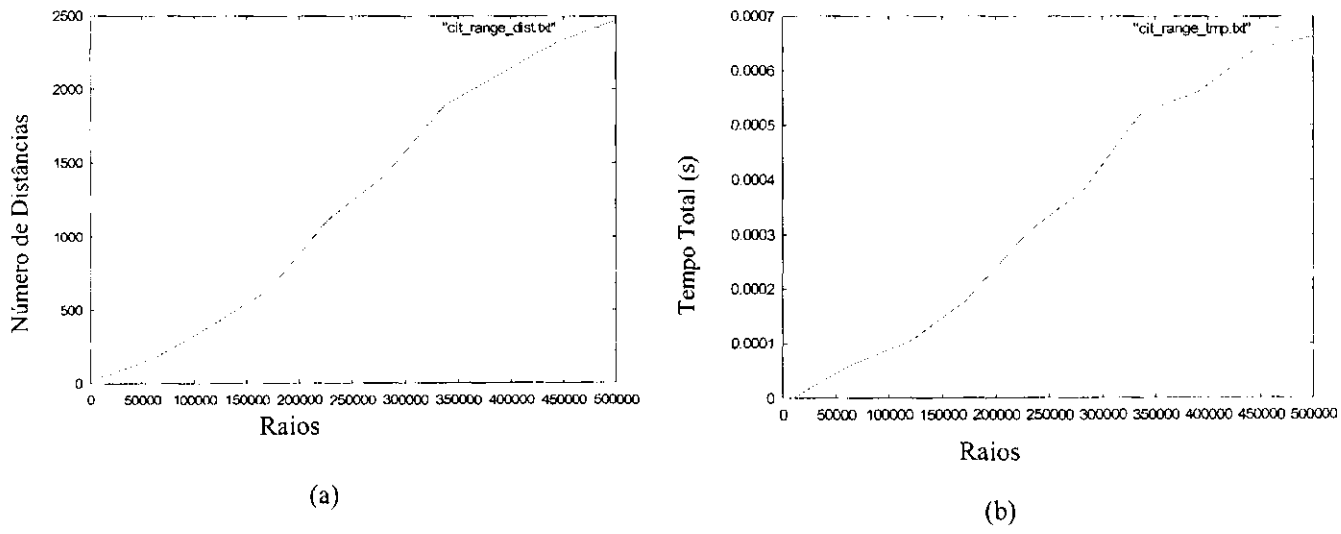


Figura 25: (a) Gráfico do Número de Distâncias X Valores Possíveis de Raio. (b) Gráfico do Tempo Total (em segundos) X Valores de Raio usados na consulta, usando o conjunto de dados cities para o experimento 1.

A figura 26(a) mostra a média de números de cálculos de distâncias por consulta e a figura 26(b) mostra o tempo total para responder as quinhentas consultas por abrangência com distribuição polarizada pelos objetos do conjunto Cities. Essas consultas foram geradas variando o raio de consulta desde 5% até 90% do diâmetro do conjunto.

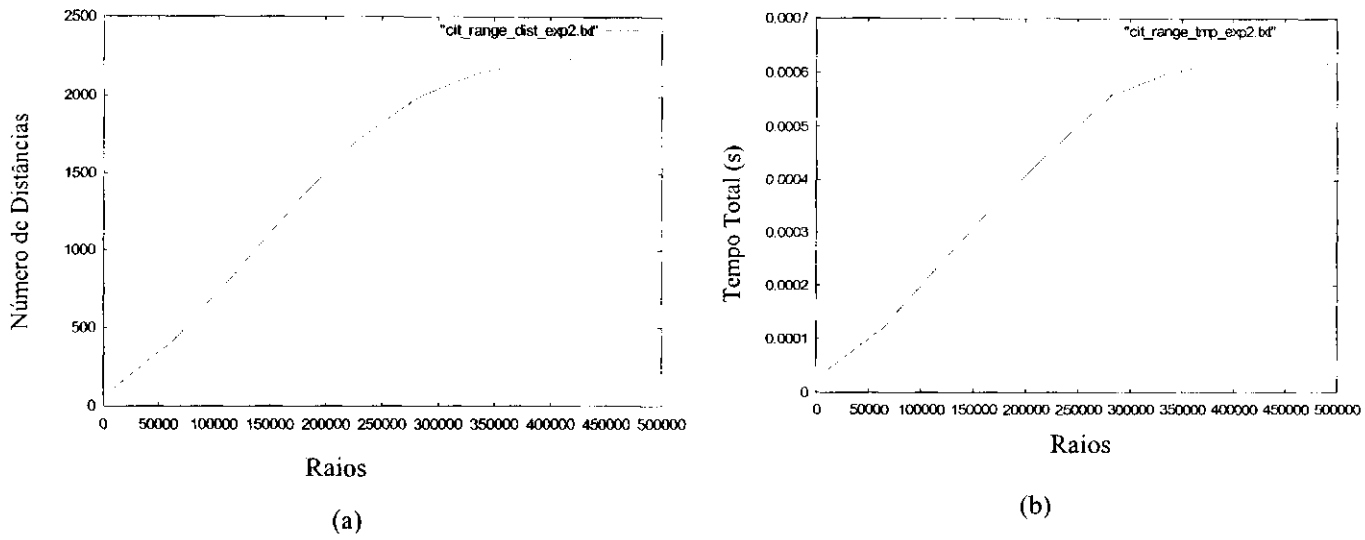


Figura 26: (a) Gráfico do Número de Distâncias X Valores Possíveis de Raio. (b) Gráfico do Tempo Total (em segundos) X Valores de Raio usados na consulta, usando o conjunto de dados cities para o experimento 2.

Consulta por k-vizinhos mais próximos

A figura 27(a) mostra a média de números de cálculos de distâncias por consulta e a figura 27(b) mostra o tempo total para responder as quinhentas consultas por k-vizinhos mais próximos uniformemente distribuídas sobre o conjunto Cities. Essas consultas foram geradas variando o valor de k de 1% até 10% do total de objetos do conjunto indexado.

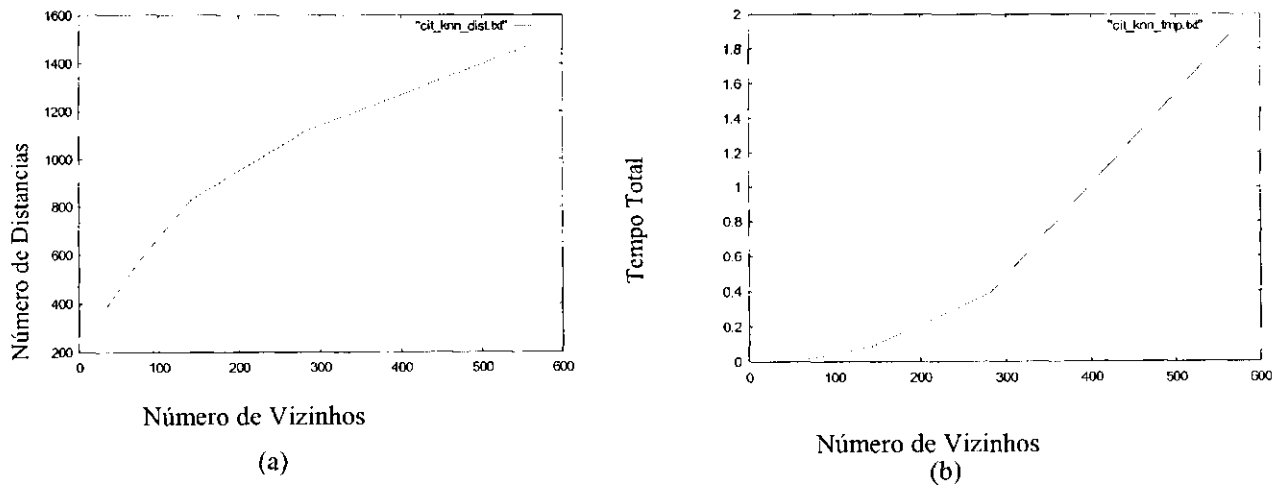


Figura 27: (a) Gráfico do Número de Distâncias X Pela Quantia de Vizinhos. (b) Gráfico do Tempo Total (em segundos) X Pela Quantia de Vizinhos, usando o conjunto de dados cities para o experimento 1.

A figura 28(a) mostra a média de números de cálculos de distâncias por consulta e a figura 28(b) mostra o tempo total para responder as quinhentas consultas por k-vizinhos mais próximos com distribuição polarizada pelos objetos do conjunto Cities. Essas consultas foram geradas variando o valor de k de 1% até 10% do total de objetos do conjunto indexado.

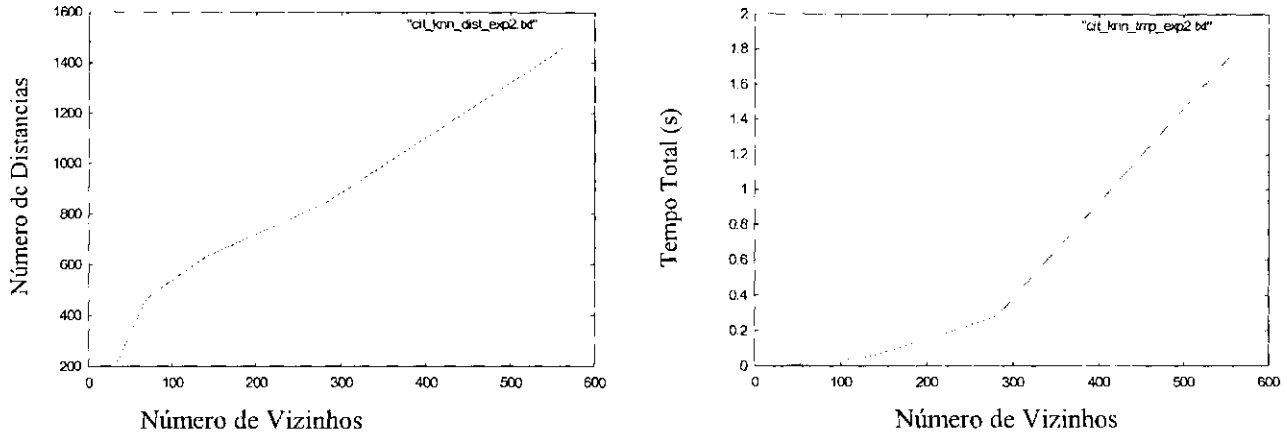


Figura 28: (a) Gráfico do Número de Distâncias X Pela Quantia de Vizinhos. (b) Gráfico do Tempo Total (em segundos) X Pela Quantia de Vizinhos, usando o conjunto de dados cities para o experimento 2.

5.3.2 – Escalabilidade da MM-Tree

Os testes de escalabilidade da estrutura foram feitos usando um conjunto sintético de dimensão 3, gerado aleatoriamente com valores que variam entre 0 e 1. O conjunto é composto de 100000 valores.

Para a realização destes testes o conjunto de dados foi dividido em 10 partes, de 10% até 100% do conjunto de dados. Os pontos para compor estas partes foram escolhidos aleatoriamente do conjunto original.

A seguir são apresentados os gráficos mostrando o comportamento da estrutura, considerando tempo e número de distâncias, para a montagem da estrutura e para a execução das consultas por similaridade.

A figura 29 mostra o comportamento da estrutura para a montagem considerando no gráfico o número de cálculos de distâncias e tempo para todas as 10 partes do conjunto usando. Como pode ser observado, a estrutura tem um comportamento linear para o número de cálculos de distância, e comportamento superlinear para o tempo total, confirmando assim a previsão teórica de um

comportamento entre linear, para árvores totalmente balanceadas, e de ordem quadrática para árvores totalmente degeneradas.

Na figura 30 são apresentados os resultados do comportamento da estrutura, considerando o número de cálculos de distância e tempo total, para a execução da consulta por abrangência. Para a obtenção de cada um dos valores colocados no gráfico, foram escolhidos objetos aleatórios que não faziam parte do conjunto de dados e um raio fixo no valor de 0.2. A distribuição usada para o experimento foi uma distribuição uniforme sobre o conjunto de dados, ou seja, os objetos escolhidos para a execução das consultas são objetos que não fazem parte do conjunto de dados usado para a montagem da estrutura.

Na figura 31 são apresentados os resultados do comportamento da estrutura, considerando o número de cálculos de distância e tempo total, para a execução da consulta por k-vizinhos mais próximos. Para a obtenção de cada um dos valores colocados no gráfico, foram escolhidos objetos aleatórios que não faziam parte do conjunto de dados e um valor para k(número de vizinhos) fixo no valor de 500. A distribuição usada para o experimento foi uma distribuição uniforme sobre o conjunto de dados.

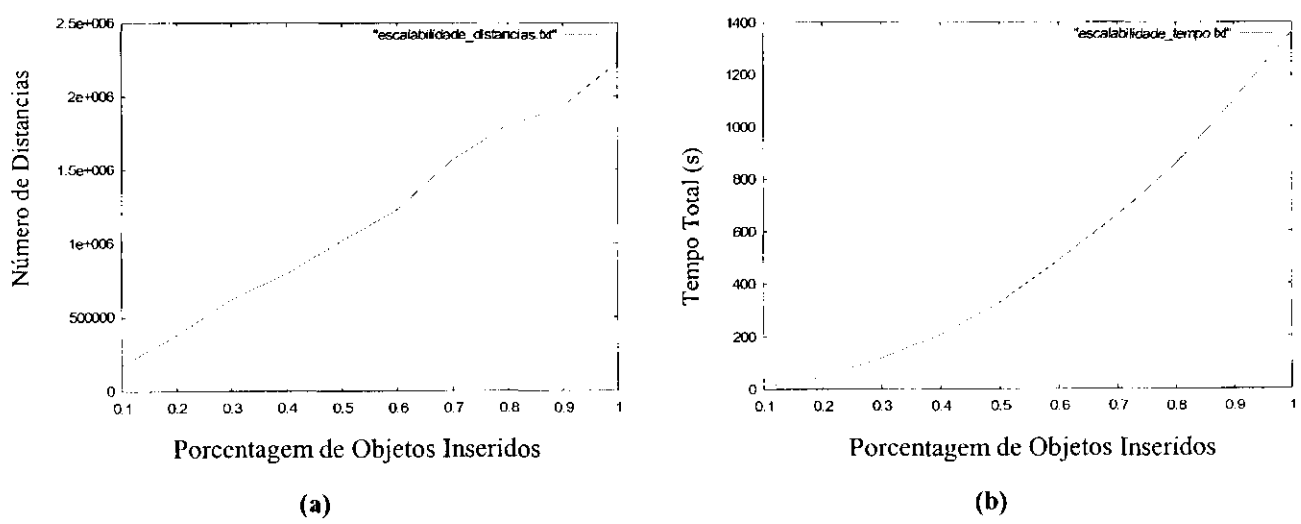
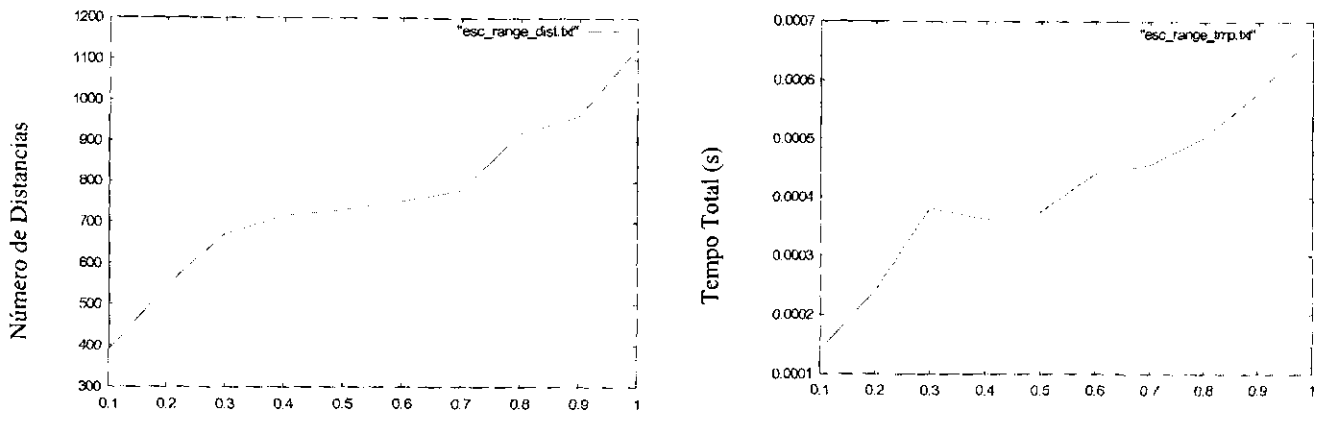
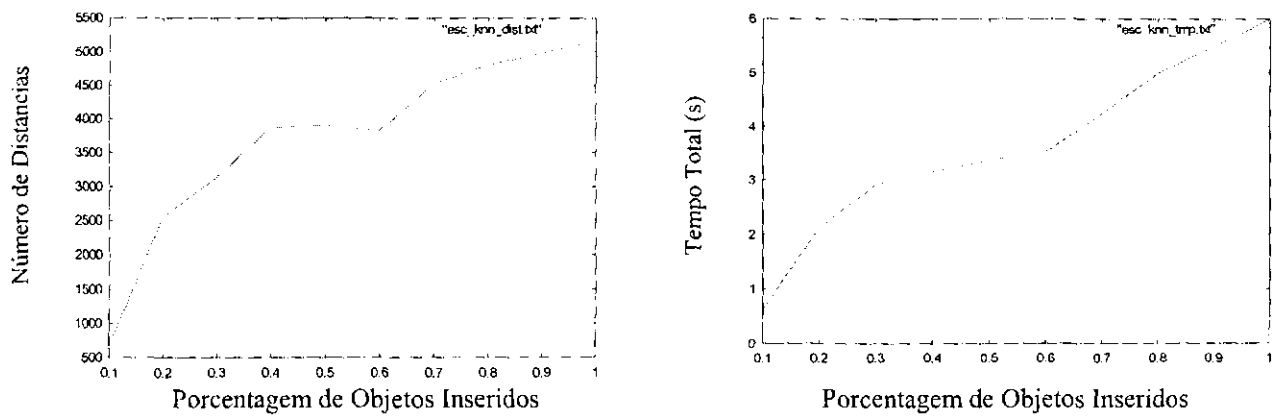


Figura 29: Gráfico de escalabilidade para a montagem da estrutura. Os dados considerados são: no eixo x, a porcentagem do conjunto inserida na estrutura; e no eixo y, o numero de cálculos de distância na parte (a) e o tempo total da inserção para cada uma das porcentagens consideradas na parte (b).



(a) **(b)**

Figura 30: Gráfico de escalabilidade para a consulta por abrangência. Os dados considerados são: no eixo x, a porcentagem do conjunto inserida na estrutura; e no eixo y, o numero de cálculos de distância na parte (a) e o tempo total da inserção para cada uma das porcentagens consideradas na parte (b).



(a) **(b)**

Figura 31: Gráfico de escalabilidade para a consulta por vizinhos mais próximos. Os dados considerados são: no eixo x, a porcentagem do conjunto inserida na estrutura; e no eixo y, o numero de cálculos de distância na parte (a) e o tempo total da inserção para cada uma das porcentagens consideradas na parte (b).

5.4- Resultados Experimentais Comparativos da MM-Tree

Apenas as informações individuais da estrutura não são suficientes para concluir sua eficiência. É preciso comparar os resultados obtidos com uma outra estrutura já existente na literatura. A estrutura escolhida para comparação foi a *Vantage Point Tree* (VP-Tree).

Além da VP-Tree foram realizados alguns testes comparativos usando a Slim-Tree, mas somente a consulta por abrangência foi analisada, e comparando-se apenas o número de cálculos de distância efetuados.

As estruturas foram testadas usando os mesmos conjuntos já apresentados anteriormente, usando todos os objetos dos conjuntos na inserção, para dois dos conjuntos de dados, e os objetos de consultas foram gerados segundo uma distribuição uniforme. Para um dos conjuntos de dados foram deixados quinhentos objetos sem inserir, estes objetos foram usados para a execução das consultas caracterizando uma distribuição polarizada dos objetos.

A seguir são apresentados os resultados obtidos com cada um dos conjuntos de dados para as duas estruturas e uma breve análise dos resultados. Para cada conjunto de dados foi usada um dos experimentos já apresentados na seção anterior. O experimento usando distribuição uniforme dos centros das consultas foi usado para os conjuntos Cities e PortugueseSWords e o experimento com distribuição polarizada para o conjunto Eigenfaces.

5.4.1 – Medidas de Construção e Consulta da MM-Tree X VP-Tree nos conjuntos de teste reais

Nesta seção mostra-se o comportamento da MM-Tree e da VP-Tree para a inserção e consulta dos três conjuntos de dados de teste reais: Eigenfaces, PortugueseSWords e cities. Para cada conjunto mostra-se o tempo total de montagem da estrutura e o número de distâncias calculadas obtidas em cada árvore. Um total de seis árvores foram construídas para obtenção das medidas: duas para cada conjuntos de dados, sendo uma MM-Tree e uma VP-Tree.

Cities

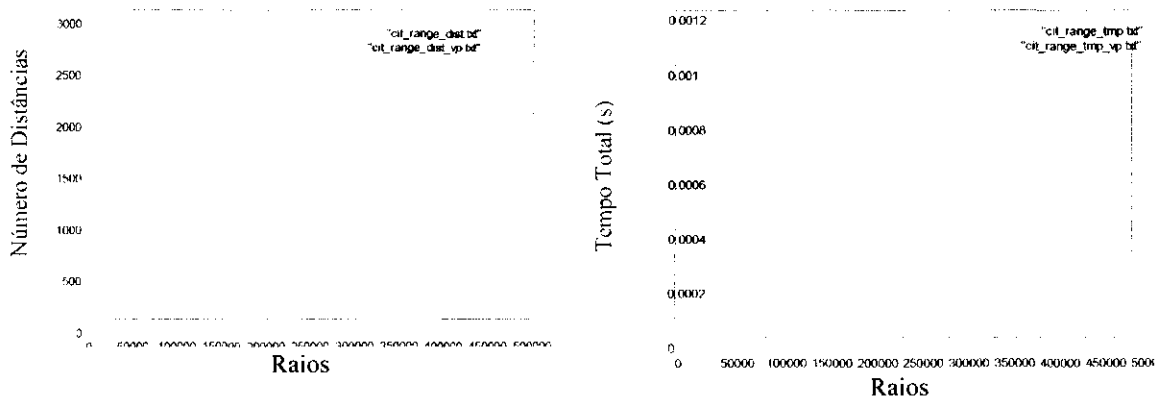
A tabela 6 mostra os parâmetros usados para construir as duas árvores do conjunto cities, bem como as medidas de tempo total e o número de cálculos de distância obtidos para cada uma das estruturas.

Informações	**VP-Tree**	**MM-Tree**
Número de Objetos Inseridos	2755	2755
Dimensão	2	2
Métrica usada	L_2	L_2
Tempo de Inserção	0.86 segundos	0.53 segundos
Número de Distâncias Calculadas	778851	39682

Tabela 6: Dados de inserção para a VP-Tree e MM-Tree usando o conjunto de dados cities.

Consulta por Abrangência

A figura 32(a) mostra a média de números de cálculos de distâncias por consulta e a figura 32(b) mostra o tempo total para responder as quinhentas consultas por abrangência uniformemente distribuídas sobre o conjunto Cities. Essas consultas foram geradas variando o raio de consulta desde 5% até 90% do diâmetro do conjunto.



(a) (b)

Figura 32: Gráfico Comparativo entre MM-Tree e VP-Tree, da consulta por abrangência usando o conjunto cities.

Consulta por k-vizinhos mais próximos

A figura 33(a) mostra a média de números de cálculos de distâncias por consulta e a figura 33(b) mostra o tempo total para responder as quinhentas consultas por k-vizinhos mais próximos uniformemente distribuídas sobre o conjunto Cities. Essas consultas foram geradas variando o valor de k de 1% até 10% do total de objetos do conjunto indexado.

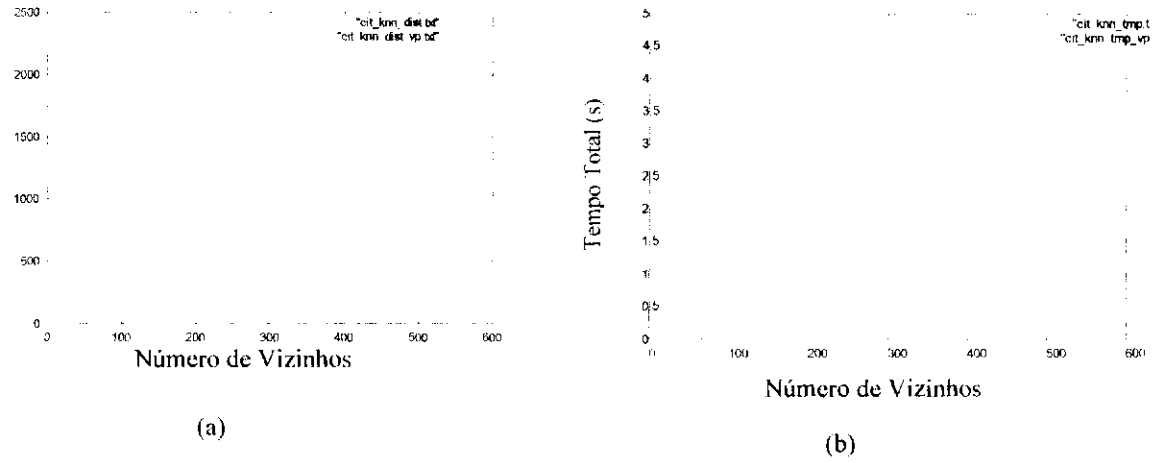


Figura 33: Gráfico Comparativo entre MM-Tree e VP-Tree, da consulta por k-vizinhos mais próximos usando o conjunto cities.

Para o conjunto de cidades, considerando-se a consulta por abrangência, pode-se observar que o número de distâncias calculadas pelas duas estruturas é praticamente o mesmo para raios menores e a MM-Tree chega a 10% menos cálculos de distância para raios maiores. No entanto, no tempo de execução da consulta por abrangência, a MM-Trre tem um ganho considerável para raios maiores que 200000, quando comparada com a VP-Tree.

Já quando se considera a consulta por k-vizinhos mais próximos, temos que a MM-Tree tem ganhos que se aproximam de 50% menos cálculos de distância e para tempos os ganhos ficam bastante consideráveis chegando também a uma execução em metade do tempo.

PortugueseSWords

A tabela 7 mostra os parâmetros usados para construir as duas árvores do conjunto PortugueseSWords, bem como as medidas de tempo total e o número de cálculos de distância obtidos para cada uma das estruturas usadas.

Informações	**VP-Tree**	**MM-Tree**
Número de Objetos Inseridos	21473	21473
Dimensão	-	-
Métrica usada	L_{edit}	L_{edit}
Tempo de Inserção	94.02 segundos	62.08 segundos
Número de Distâncias Calculadas	6754363	673124

Tabela 7: Dados de inserção para a VP-Tree e MM-Tree usando o conjunto de dados PortugueseSWords.

Consulta por Abrangência

A figura 34(a) mostra a média de números de cálculos de distâncias por consulta e a figura 34(b) mostra o tempo total para responder as quinhentas consultas por abrangência uniformemente distribuídas sobre o conjunto PortugueseSWords. Essas consultas foram geradas variando o raio de consulta desde 1 até 10 caracteres inseridos/removidos/substituídos para localizar as respostas.

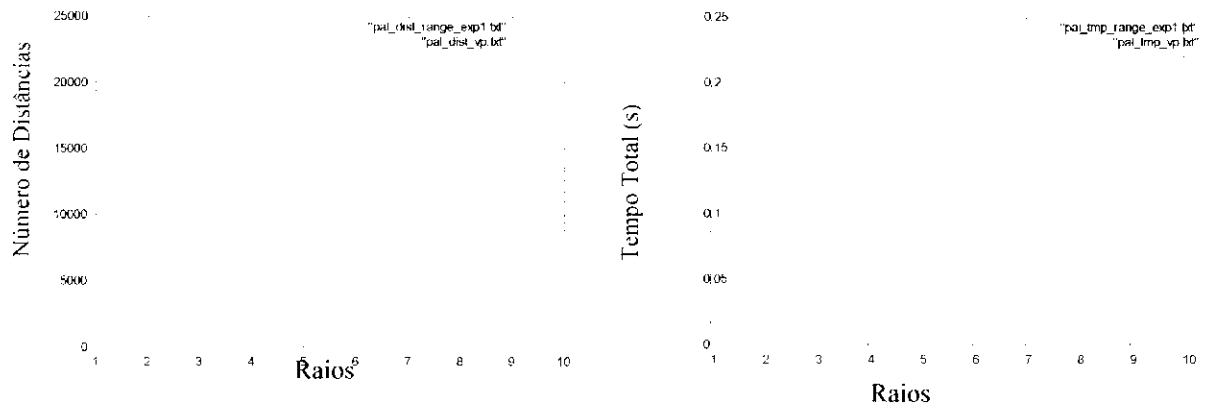


Figura 34: Gráfico Comparativo entre MM-Tree e VP-Tree, da consulta por abrangência usando o conjunto palavras em português.

Consulta por vizinhos mais próximos

A figura 35(a) mostra a média de números de cálculos de distâncias por consulta e a figura 35(b) mostra o tempo total para responder as quinhentas consultas por k-vizinhos mais próximos uniformemente distribuídas sobre o conjunto PortugueseSWords. Essas consultas foram geradas variando o valor de k de 1% até 10% do total de objetos do conjunto indexado.

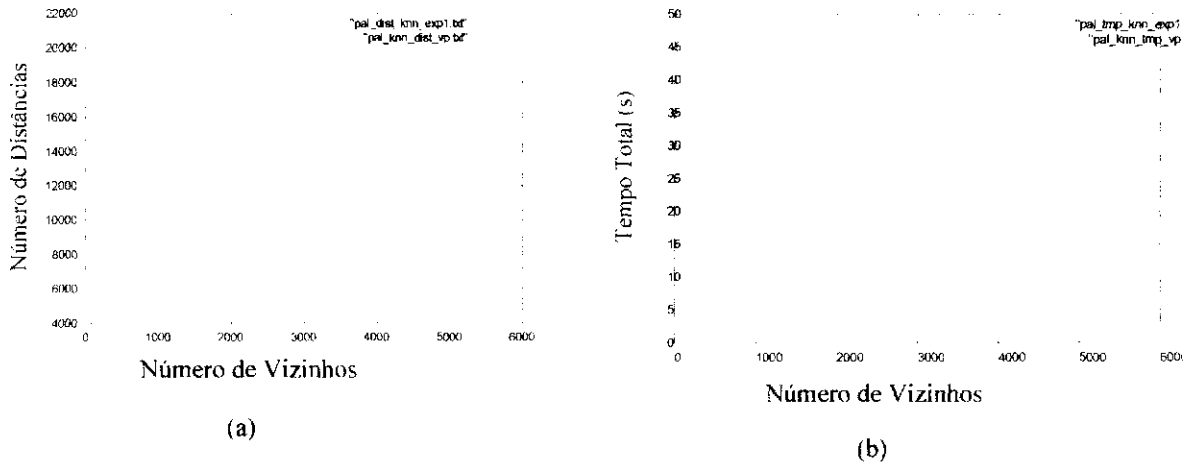


Figura 35: Gráfico Comparativo entre MM-Tree e VP-Tree, da consulta por vizinhos mais próximos usando o conjunto palavras em português.

Para o conjunto de palavras em português tem-se um ganho apenas em tempo para range query, pois em distâncias as duas estruturas ficam equivalentes. O ganho de tempo fica em torno de 50% ou mais quando comparada a MM-Tree com a VP-Tree.

Quando se trata da knn query o ganho da MM-Tree é bastante considerável quando verificado em números, assim como aconteceu no conjunto de cidades.

Eigenfaces

A tabela 8 mostra os parâmetros usados para construir as duas árvores do conjunto Eigenfaces, bem como as medidas de tempo total e o número de cálculos de distância obtidos para cada uma das estruturas usadas.

Informações	**VP-Tree**	**MM-Tree**
Número de Objetos Inseridos	11400	11400
Objetos não inseridos	500	500
Dimensão	16	16
Métrica usada	L_2	L_2
Tempo de Inserção	40.61 segundos	12.83 segundos
Número de Distâncias Calculadas	3298714	221768

Tabela 8: Dados de inserção para a VP-Tree e MM-Tree usando o conjunto de dados Enigenfaces.

Consulta por Abrangência

A figura 36(a) mostra a média de números de cálculos de distâncias por consulta e a figura 36(b) mostra o tempo total para responder as quinhentas consultas por abrangência com distribuição polarizada pelos objetos do conjunto Eigenfaces. Essas consultas foram geradas variando o raio de consulta desde 10% até 90% do diâmetro do conjunto.

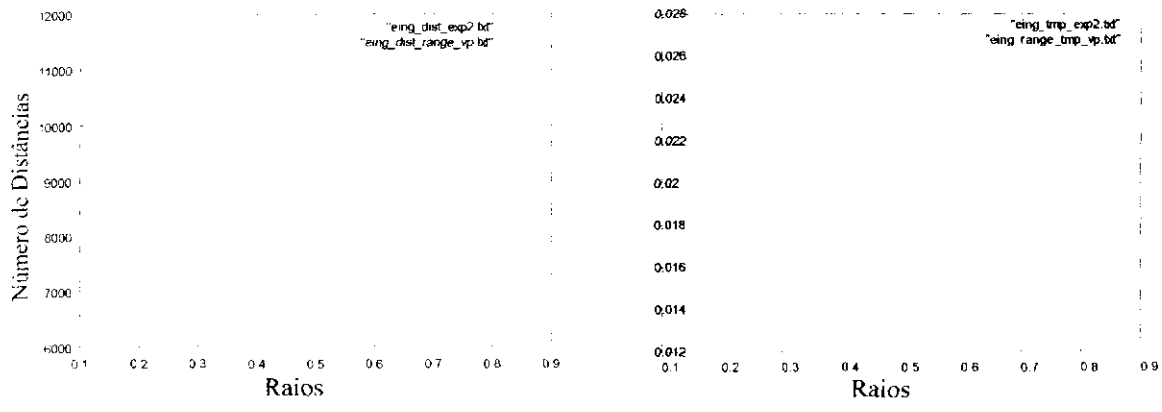


Figura 36: Gráfico Comparativo entre MM-Tree e VP-Tree, da consulta por abrangência usando o conjunto Eigenfaces.

Consulta por vizinhos mais próximos

A figura 37(a) mostra a média de números de cálculos de distâncias por consulta e a figura 37(b) mostra o tempo total para responder as quinhentas consultas por k-vizinhos mais próximos com distribuição polarizada pelos objetos do conjunto Eigenfaces. Essas consultas foram geradas variando o valor de k de 1% até 10% do total de objetos do conjunto indexado.

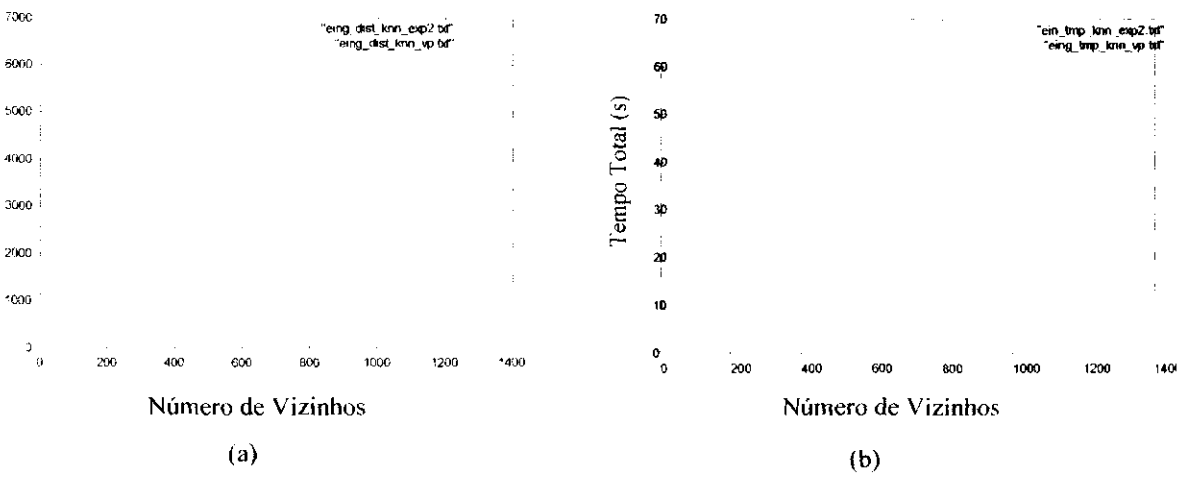


Figura 37: Gráfico Comparativo entre MM-Tree e VP-Tree, da knn query usando o conjunto Eigenfaces.

Quando se considera o conjunto de faces, tem-se um ganho da MM-Tree não só em tempo, mas também em cálculos de distâncias quando comparada com a VP-Tree para consultas por abrangência. Estes ganhos ficam em torno de 6% .

Quando consideramos a consulta por k-vizinhos mais próximos tem-se ganhos muito consideráveis da mesma forma que nos conjuntos anteriores, chegando a uma execução em 50% menos tempo.

Para a execução de uma comparação entre as duas estruturas executando a *point query* foram escolhidos alguns exemplos de objetos com distribuição uniforme e alguns exemplos de objetos com distribuição polarizada para executar as consultas. A tabela 9 mostra os resultados de cálculos de distâncias das duas estruturas considerando um dos exemplos de objetos usados para cada um dos conjuntos.

Os dados relativos a tempo da *point query* não estão mostrados, pois são muito rápidos para serem medidos, dada a baixa resolução permitida pelo sistema operacional.

Pode-se verificar que a estrutura desenvolvida tem uma performance melhor em tempo que a VP-Tree. Já para as consultas por k-vizinhos mais próximos a MM-Tree tem um ganho bastante acentuado nos cálculos de distâncias, chegando a 50% menos distâncias calculadas nos melhores casos, e nunca perdendo em nenhuma situação.

Conjunto de Dados	Tipo do objeto	Número de Distâncias	
		MM-Tree	VP-Tree
Eigenfaces	Objeto com distribuição uniforme	20	15
Eigenfaces	Objeto com distribuição polarizada	16	14
Palavras em Português	Objeto com distribuição uniforme	16	13
Palavras em Português	Objeto com distribuição polarizada	10	11
Cities	Objeto com distribuição uniforme	12	15
Cities	Objeto com distribuição polarizada	4	13

Tabela 9: Point Query executada para cada um dos conjuntos para as duas estruturas.

5.4.2 – Comparação de Número de distâncias Calculadas da MM-Tree com a Slim Tree

A seguir são apresentados os resultados de performance da comparação entre a MM-Tree e a Slim Tree usando os conjuntos de dados já apresentados. Os teste foram realizados comparando apenas as consultas por abrangência.

Nos testes considera-se apenas a comparação com o número de cálculos de distância, uma vez que a Slim Tree é uma estrutura para armazenagem em disco e, portanto não tem sentido em ser comparada com uma estrutura que armazena os dados apenas em memória.

Conjunto de dados Cities

A figura 38 mostra a média de números de cálculos de distâncias por consulta para responder as quinhentas consultas por abrangência uniformemente distribuídas sobre o conjunto Cities. Essas consultas foram geradas variando o raio de consulta desde 5% até 90% do diâmetro do conjunto.

Como pode ser visto, a MM-Tree chega a atingir até 80% menos cálculos de distâncias para pequenos raios, e chega no pior caso ganhando em até 20% menos cálculos de distâncias para raios da ordem do diâmetro do conjunto.

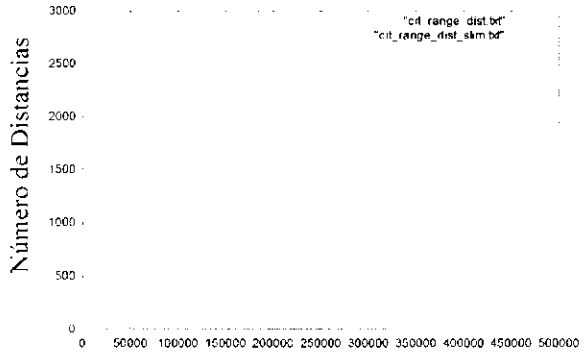


Figura 38: Gráfico Comparativo entre a Slim-Tree e a MM-Tree. Esse gráfico apresenta o número de distâncias X Valores de raio usados nas consultas. O conjunto considerado é um conjunto cities.

Conjunto de dados de PortugueseSWords

A figura 39 mostra a média de números de cálculos de distâncias por consulta para responder as quinhentas consultas por abrangência uniformemente distribuídas sobre o conjunto PortugueseSWords. Essas consultas foram geradas variando o raio de consulta desde 1 até 10 caracteres inseridos/removidos/substituídos para localizar as respostas.

Como pode ser visto, a MM-Tree chega a atingir até 85% menos cálculos de distância para pequenos raios, e chega no pior caso ganhando em até 8% menos cálculos de distâncias para raios da ordem do diâmetro do conjunto.

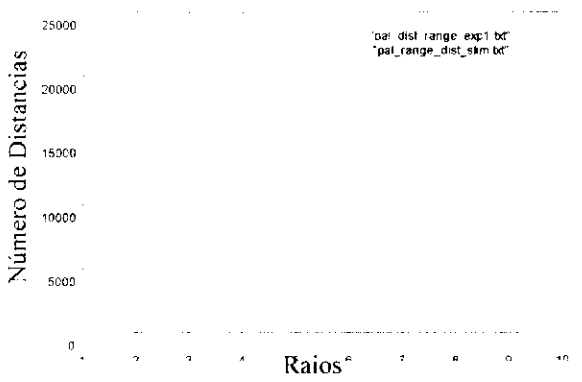


Figura 39: Gráfico Comparativo entre a Slim-Tree e a MM-Tree. Esse gráfico apresenta o número de distâncias X Valores de raio usados nas consultas, usando o conjunto das palavras em Português.

Conjunto de dados de Eingfaces

A figura 40 mostra a média de números de cálculos de distâncias por consulta para responder as quinhentas consultas por abrangência uniformemente distribuídas sobre o conjunto Eigenfaces. Essas consultas foram geradas variando o raio de consulta desde 10% até 90% do diâmetro do conjunto.

Como pode ser visto, a MM-Tree chega a atingir até 70% menos cálculos de distâncias para pequenos raios, e chega no pior caso ganhando em até 8% menos cálculos de distâncias para raios da ordem do diâmetro do conjunto.

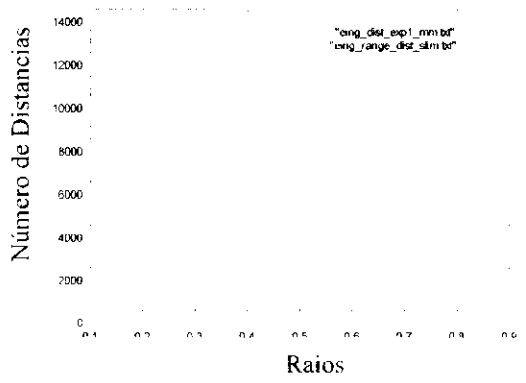


Figura 40: Gráfico Comparativo entre a Slim-Tree e a MM-Tree. Esse gráfico apresenta o numero de distâncias X Valores de raio usados nas consultas. O conjunto considerado é o conjunto Eigenfaces.

Note que pelos gráficos, a MM-Tree tem um ganho bastante considerável em cálculos de distância quando comparada a Slim Tree, que é uma estrutura métrica muito mais complexa, principalmente para consultas com raios inferiores a 20% do diâmetro do conjunto.

5.5- Considerações Finais

Neste capitulo foram apresentados os conjuntos de dados usados nos testes e os gráficos referentes aos resultados dos testes de performance realizados na estrutura MM-Tree.

Ainda neste capítulo, foram apresentados os resultados comparativos feitos usando duas estruturas já existentes na literatura. As estruturas usadas foram a VP-Tree e a Slim-Tree.

A partir dos dados coletados da estrutura, verifica-se que a MM-Tree tem um ganho no tempo de execução das consultas que chega a ser de 50% mais rápida que a VP-Tree além de ser muito simples e toda armazenada em memória. Já no cálculo de distâncias o ganho foi mais significativo nas consultas

por vizinhos mais próximos, quando comparado com a VP-Tree. Nas comparações feitas com a Slim-Tree os ganhos da MM-Tree em cálculos de distância também foram bastante significativos, chegando a 85% menos cálculos de distâncias.

CAPÍTULO 6

CONCLUSÃO

O processo de organização da informação efetuado pelos sistemas de gerenciamento banco de dados utiliza como base as estruturas de indexação para acelerar a busca aos dados. Desta forma, as estruturas de indexação são ferramentas fundamentais que habilitam os sistemas gerenciadores de dados a armazenar e recuperar informações eficientemente em grandes volumes de dados. Esse trabalho de mestrado está centrado na área de Banco de Dados focalizando estruturas de indexação para dados métricos. A motivação deste trabalho baseou-se nas necessidades apresentadas pelas aplicações que necessitam a indexação de dados em espaços métricos que devem manipular dados de maneira transiente, e, portanto não precisam que estes sejam tornados persistentes pela sua armazenagem em disco. Exemplos imediatos de classes de aplicações que têm essa necessidade é a integração de dados para Web, a geração de simuladores que demandam operação de classificação de dados, e sistemas de processamento de imagens que envolvam realismo na geração de imagens dinâmicas.

Esse trabalho contribuiu para o estado da arte da área da indexação de dados métricos com o desenvolvimento de uma nova estrutura de indexação métrica, denominada MM-Tree. Esta estrutura é bastante simples, e faz a partição dos dados em domínios métricos baseando-se em dois representantes por nó e na distância métrica entre eles. Com base nesta distância, forma-se em cada nível quatro regiões possíveis onde um objeto a ser inserido pode ser colocado. Esta estrutura é dinâmica, pois permite que novos objetos sejam inseridos na estrutura a qualquer instante, mesmo depois que a estrutura já estiver construída. Ela é uma estrutura totalmente armazenada em memória, o que dispensa a necessidade de particionar os nós em blocos de tamanho fixo. Para um conjunto de N objetos, a ordem de complexidade de construção da estrutura no pior caso é de $N2$ quando se tem uma árvore totalmente degenerada e de $\Omega(N*\log N)$ quando se tem uma árvore balanceada. Os testes demonstraram que o pior caso raramente ocorre. Para realização de uma consulta por existência é executado um único

caminho de busca, percorrendo-se em cada nível somente uma das regiões. Além da consulta por existência foram desenvolvidos os algoritmos para executar as buscar por similaridade por abrangência e por vizinhos mais próximos.

Essa árvore pode ser classificada como aquela que ocupa, para dados em domínios métricos, o mesmo lugar que a árvore binária ocupa junto aos métodos que tratam dados em domínios que atendem B propriedade de ordem total, ou que as *Quad-tree* ocupam para dados em domínios vetoriais espaciais, ou que as kd-tree ocupam para dados em domínios vetoriais pontuais.

A estrutura foi implementada num protótipo que permitiu as medidas de desempenho e a comparação com duas outras estruturas já existentes na literatura para domínios métricos. As duas estruturas escolhidas para a comparação foram a VP-Tree e a Slim-Tree. Resumindo-se, podemos dizer que criação da primeira estrutura métrica dinâmica para execução em memória é uma contribuição inédita deste trabalho. Esta estrutura possui ainda a propriedade de permitir a execução de consultas do tipo *consulta* por existência sem que haja sobreposição de nós em um mesmo nível da árvore, quesito no qual se caracteriza também como inédita (primeira árvore dinâmica para espaços métricos que permite consultas por existência sem sobreposição dos nós).

6.1 – Sugestões de Trabalhos Futuros

A estrutura desenvolvida, ainda tem muitos aspectos que permitem várias opções ainda a serem estudadas e muito que ainda pode ser desenvolvido na estrutura MM-Tree, visando melhorar seu desempenho e adaptá-la para aplicações específicas. Alguns dos possíveis trabalhos futuros são:

- Desenvolvimento do algoritmo para Inserção em Blocos (*bulk loading*): este algoritmo não insere os objetos individualmente, mas sim em conjuntos de objetos. Os objetos são agrupados em conjuntos seguindo algum critério ou mesmo agrupados aleatoriamente, e dos conjuntos são escolhidos os pares de representantes que são posteriormente inseridos na estrutura. Este tipo de inserção permite que os objetos sejam analisados e representantes mais significativos possam ser escolhidos, tornando a árvore mais bem balanceada.Esse algoritmo é básico para permitir a operação de "mistura" (*merging*) de duas árvores já criadas.
- Algoritmos de balanceamento: para um melhor desempenho nas consultas, um algoritmo de balanceamento pode conseguir melhor desempenho. Este algoritmo pode ser derivado de conceitos equivalentes ao usado no o algoritmo existente nas árvores AVL, ou seja, a cada inserção nova o algoritmo verifica se a árvore tornou–se desbalanceada. Caso tenho ocorrido um desbalanceamento,

pode-se aplicar um procedimento para rearranjar os objetos na árvore visando manter o balanceamento.

- Mistura de Árvores (*Merging* de duas árvores): dispondo-se de duas estruturas prontas, freqüentemente é mais eficiente percorrer as duas estruturas, verificando-se como são as interseções das regiões de cobertura dos representantes de cada uma das árvores em cada nível. Com base nessa intersecção pode-se prever que a definição de onde os objetos devem ser colocados em uma única árvore deve ser mais eficiente do que re-inserindo-se individualmente os elementos já armazenados em uma árvore na outra.
- Re-implementação da busca por vizinhos mais próximos usando as otimizações já propostas no algoritmo do capítulo 4.

CAPÍTULO 7

REFERÊNCIAS BIBLIOGRÁFICAS

[Aggarwal_1999] C. C. Aggarwal, J. L. Wolf, P. S. Yu, "*Caching on the World Wide Web*," em *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, Janeiro/Fevereiro 1999.

[Ashish_1999] N. Ashish, C. A. Knoblock, C. Shahabi, "*Selectively Materializing Data in Mediators by Analyzing User Queries*," apresentado em Fourth IEEE International Conference on Cooperative Information Systems - IECIS, Edinburgh, Scotland, 2 - 4 de Setembro de1999.

[Baeza-Yates_1994] R. A. Baeza-Yates, W. Cunto, U. Manber, S. Wu, "*Proximity Matching Using Fixed-Queries Trees*," apresentado no 5th Annual Symp on Combinatorial Pattern Matching (CPM), Asilomar, CA, 5-8 de Junho de 1994.

[Beckmann_1990] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, "*The R*-tree: An Efficient and Robust Access Method for Points and Rectangles*," apresntado na ACM Int'l Conference on Data Management (SIGMOD), 1990.

[Bentley_1975] J. L. Bentley, "*Multidimensional Binary Search Trees Used for Associative Searching*," em*Comunications of the ACM*, vol. 18, Setembro de 19975, pp. 509-517.

[Böhm_2000] C. Böhm, B. Braunmüller, H.-P. Kriegel, M. Schubert, "*Efficient Similarity Search in Digital Libraries*apresntado no IEEE Advances in Digital Libraries 2000 - ADL2000, Washington, D.C., 22 - 24 de Maio de 2000.

[Bozkaya_1997] T. Bozkaya and Z. M. Özsoyoglu, "*Distance-Based Indexing for High-Dimensional Metric Spaces*," apresnado na ACM Int'l Conference on Data Management (SIGMOD), Tucson, AZ, 1997.

[Bozkaya_1999] T. Bozkaya and Z. M. Özsoyoglu, "*Indexing Large Metric Spaces for Similarity Search Queries*," em *ACM Transactions on Database Systems (TODS)*, vol. 24,Setembro de1999, pp. 361-404.

[Burkhard_1973] W. A. Burkhard and R. M. Keller, "*Some Approaches to Best-Match File Searching*," em *Communications of the ACM*, vol. 16,1973, pp. 230-236.

[Chan_2000] E. P. F. Chan and K. Ueda, "*Efficient Query Result Retrieval over the Web*," apresntado na Seventh IEEE International Conference on Parallel and Distributed Systems (ICPADS'00), Iwate, Japão, 4 - 7 de Julho de 2000.

[Chavez_2001] Chávez, E., G. Navarro, *"Searching in Metric Spaces."* Para aparecer na the ACM Computing Surveys, 2001.

[Cheong_2000] J.-H. Cheong and S.-G. Lee, "*A Boolean Query Processing with a Result Cache in Mediator Systems*," apresntado na IEEE Advances in Digital Libraries 2000 - ADL2000, Washington, D.C., 22 - 24 Maio de 2000.

[Chidlovskii_2000] B. Chidlovskii and U. M. Borghoff, "*Semantic Caching of Web Queries*," no *The VLDB Journal*, vol. 9, 17 Maio de 2000, pp. 2-17.

[Ciaccia_1997] P. Ciaccia, M. Patella, P. Zezula, "*M-tree: An efficient access method for similarity search in metric spaces*," aprestando na Intl Conf on Very Large Databases (VLDB), Athens, Greece, Setembro de 1997.

[Comer_1979] D. Comer, "*The Ubiquitous B-Tree*," em *ACM Computing Surveys 11(2):*, vol. 11, Junho de 1979, pp. 121-137.

[Faloutsos_1996] C. Faloutsos, *Searching Multimedia Databases by Content*. Moston, MA: Kluwer Academic Publishers, 1996.

[Faloutsos_1997] C. Faloutsos, "Indexing of Multimedia Data," in *Multimedia Databases in Perspective*: Springer Verlag, 1997, pp. 219-245.

[Freeston_1987] M. Freeston, "*The BANG File: A New Kind of Grid File*," presented at SIGMOD Conference 1987: 260-269, San Francisco, California, 27-29 de Maio de 1987.

[Gaede_1995] V. Gaede, "*Geometric Information Makes Spatial Query Processing More Efficient*," presented at 3rd ACM International Workshop on Advances in Geographic Information Systems - ACM-GIS, Baltimore, Maryland, 1-2 de Dezembro de 1995.

[Gaede_1998] V. Gaede and O. Günther, "*Multidimensional Access Methods*," in *ACM Computing Surveys*, vol. 30, Junho de1998, pp. 170-231.

[Guttman_1984] A. Guttman, "*R-Tree : A dynamic Index Structure for Spatial Searching*," apresentado na ACM Int'l Conference on Data Management (SIGMOD), Boston, MA, 1984.

[Korth_1999] H. F. Korth, A. Silberschatz, S. Sudarshan, "Sistemas de Bancos De Dados", 3. ed., São Paulo: Makron Books, 1999.

[Lomet_1989] D. B. Lomet and B. Salzberg, "*A Robust Multi-Attribute Search Structure*," aprestando na IEEE Fifth International Conference on Data Engineering - ICDE, Los Angeles - CA, 6-10 de Fevereiro de 1989.

[Lomet_1990] D. B. Lomet and B. Salzberg, "*The hB-Tree: A Multiattribute Indexing Method with Good Guaranteed Performance*," em *ACM Trans on Database Systems*, vol. 15, Dezembro de 1990, pp. 625-658.

[Lomet_2001]: D. B. Lomet, "*The Evolution of Effective B-tree: Page Organization and Techniques: A Personal Account*," em *SIGMOD Record*, vol. 30, 2001, pp. 64-69.

[Orlowska_2001] M. E. Orlowska, "*Integrating Web Based Applications - Challenges and Opportunities (Tutorial)*," apresentado na 7th International Conference on Database Systems for Advanced Applications, Hong Kong, 18-20 de Abril de 2001.