Avaliação de Desempenho de Estruturas de Acesso a Dados Hiperdimensionais

Nathan Gevaerd Colossi

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Avaliação de Desempenho de Estruturas de Acesso a Dados Hiperdimensionais

Nathan Gevaerd Colossi

21 de março de 2000

Banca Examinadora:

- Prof. Dr. Mário Antonio do Nascimento
  University of Alberta (Orientador)
- Prof. Dr. Marcelo Finger
  Universidade de São Paulo
- Profª Drª Claudia Bauzer Medeiros
  Universidade Estadual de Campinas
- Prof. Dr. Luiz Eduardo Buzato (Suplente)
  Universidade Estadual de Campinas

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Colossi, Nathan Gevaerd

C719a Avaliação de desempenho de estruturas de acesso a dados hiperdimensionais / Nathan Gevaerd Colossi -- Campinas, [S.P. :s.n.], 2000.

Orientador : Mário Antônio Nascimento

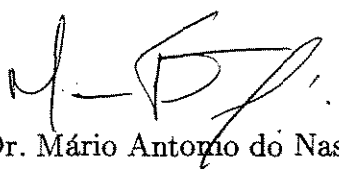
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Banco de dados. 2 Indexação. 3. Estruturas de dados (Computação). 4. Multimídia. I. Nascimento, Mário Antônio. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

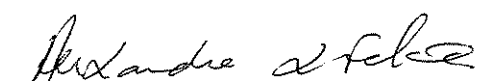
Avaliação de Desempenho de Estruturas de Acesso a Dados Hiperdimensionais

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Nathan Gevaerd Colossi e aprovada pela Banca Examinadora.

Campinas, 22 de março de 2000.



Prof. Dr. Mário Antonio do Nascimento
University of Alberta (Orientador)

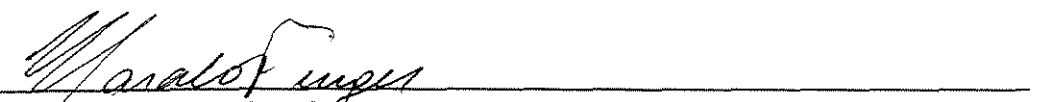


Prof. Dr. Alexandre Xavier Falcão
Universidade Estadual de Campinas
(Co-orientador)

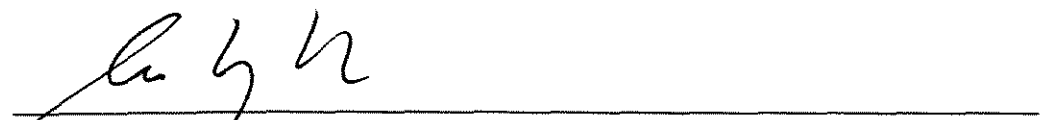
Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

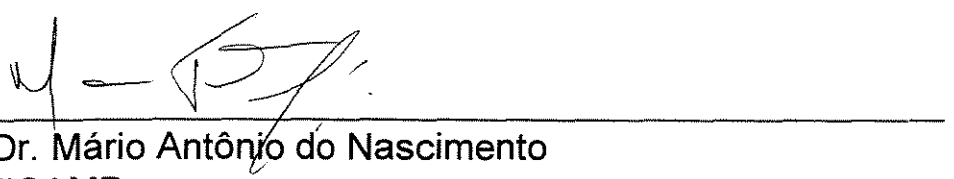
Tese defendida e aprovada em 21 de março de 2000, pela Banca Examinadora composta pelos Professores Doutores:



Prof. Dr. Marcelo Finger
IME-USP



Profa. Dra. Cláudia Maria Bauzer Medeiros
IC-UNICAMP



Prof. Dr. Mário Antônio do Nascimento
IC-UNICAMP

Agradecimentos

- Ao CNPq e ao Projeto PRONEX II-MCT/FINEP SAI (Sistemas Avançados de Informação) pelo apoio financeiro concedido.
- Ao meu orientador e amigo, Mario A. Nascimento, pela excelente orientação e pela liberdade de criação dada a mim.
- A meus pais e familiares por terem sempre me incentivado nas decisões importantes da minha vida.
- Ao Grupo de Banco de Dados, comandado por Claudia Bauzer Medeiros, pelas valiosas discussões de "almoço".
- Aos amigos Paulo, Bruno, Sandro, *Maringuetes*, Marcelo, Leonardo e, em especial, Marta, pelo companherismo e amizade sinceros.
- À Alexandre Falcão, pelo apoio logístico realizado.
- Aos colegas e funcionários do Instituto de Computação pelo agradável ambiente de trabalho.
- A Deus e ao meu anjo da guarda, pela proteção dada durante esta caminhada.

Resumo

Em bancos de dados multimídia é comum a representação de objetos utilizando vetores de características, que são, por sua vez, mapeados em um espaço multidimensional. Nesta dissertação, os objetos utilizados são imagens, e os vetores de características são obtidos através dos seus histogramas de cores. O mapeamento dos vetores de características em um espaço multidimensional permite a utilização de estruturas de indexação espaciais, proporcionando a realização de consultas de similaridade de forma eficiente.

Neste trabalho são avaliadas algumas estruturas de indexação para dados multidimensionais, que vão de estruturas espaciais tradicionais, como a R-tree e a R*-tree, a estruturas espaciais adaptadas para espaços hiperdimensionais, como a SS-tree e a SR-tree. De fato, este trabalho se concentra no aspecto da alta dimensionalidade dos vetores de características. Paralelo a estas estruturas, a M-tree, que realiza a indexação dos vetores de características de forma adimensional, i.e., no espaço métrico, é também avaliada. Para completar a avaliação, é feita a comparação dessa estruturas em relação a busca linear, a fim de confirmar a eficiência das estruturas avaliadas.

Para assegurar um ambiente de avaliação homogêneo, foi utilizado o ambiente de programação GiST para a implementação das estruturas, e, nas avaliações das estruturas, foi utilizado um conjunto de dados reais de 40.000 elementos. Um conjunto bastante amplo de parâmetros de construção e consulta dos índices permitiu a avaliação das estruturas.

Nos resultados obtidos, a SR-tree se mostrou a melhor estrutura com os conjuntos de dados reais. A M-tree mostrou poder alcançar bons resultados, dependendo da técnica de *split* utilizada. Nesta dissertação são propostas novas técnicas de *split* sendo uma delas mais robusta em relação ao aumento do número de dimensões. Além desses resultados, é mostrado que o uso de número de páginas acessadas como único indicador de desempenho pode levar a conclusões incorretas.

Abstract

In multimedia databases its common to represent objects using feature vectors, which are mapped onto a multidimensional space. In this thesis, the objects are image, and their feature vectors are obtained from their color histogram. The feature vectors mapping into a multidimensional space allows the utilization of spatial access structures, in order to efficiently perform similarity queries

In this research multidimensional indexing structures are evaluated, from traditional spatial structures, R-tree and R*-tree, up to structures specially designed for high-dimensional spaces, like the SS-tree and SR-tree. Indeed, this work focus on the issue of the high-dimensionality of the feature space. Along with these structures, the M-tree, that indexes feature vectors in a non-dimensional manner, i.e., using the metric space, is also evaluated. To complete the evaluation, all the above structures are evaluated against the linear scan, in order to confirm the efficiency of the structures.

To assure a homogeneous evaluation environment, the GiST framework was used to implement the structures, and the evaluation was performed using a data set of 40,000 feature vectors. A wide set of parameters was used evaluate construction and query processing.

The results obtained, indicate the SR-tree as the best structure for the real dataset. The M-tree was shown able to obtain good results, depending primarily upon the split technique used. This thesis also proposes new split techniques, and one of them was more resilient with respect to the increase in the number of dimensions. In addition, it is also shown that using the number of accessed pages as the only performance indicator may lead to wrong conclusions.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Esta dissertação apresenta um estudo comparativo de estruturas de indexação para dados multidimensionais. É crescente o número de aplicações de banco de dados que coletam e gerenciam dados multidimensionais. Banco de imagens [NBE+93, SC96], de genomas [SBK92], e de dados provenientes de sistemas CAD [MG93] são alguns exemplos de aplicações que utilizam este tipo de dado.

Os sistemas gerenciadores de bancos de dados (SGDBs) tradicionais não foram originalmente concebidos para gerenciar tal tipo de dado, mas sim dados "convencionais", p.ex., cadeias alfanuméricas. Assim sendo, tais SGBDs não conseguem gerenciar de forma adequada esta nova necessidade. Alguns dos novos e principais problemas são:

- Suporte para novos tipos de dados (p.ex.: imagens e áudio);
- Armazenamento eficiente desses tipos dados;
- Estruturas de acesso para tais tipos de dados e
- Apresentação de dados de diferentes tipos.

Dentre os problemas apontados, esta dissertação se concentrará em avaliar estruturas de indexação existentes para a organização e o acesso a dados multidimensionais.

1.1 Motivação

A recuperação de objetos (dados multidimensionais) é um dos principais problemas a serem resolvidos. As formas de recuperação de objetos mais comuns são duas: a *navegação* e a *consulta* [BYRN99]. Na primeira, são dados ao usuário objetos que representam classes de objetos para que a navegação possa ser feita até a localização dos objetos de interesse. Em sistemas hipermídias por exemplo, esse tipo de abordagem é implementada

com a utilização de âncoras. Na segunda, o usuário informa ao sistema, através de uma linguagem ou de um ambiente visual de consulta, quais características dos objetos são mais importantes na consulta. O sistema compila tais características, processa e apresenta o resultado para o usuário de forma ordenada. As estruturas de acesso analisadas nessa dissertação estão relacionadas à esta última forma de recuperação.

Na recuperação através de consultas é necessário expressar os objetos em um formato mais compacto, para que a tarefa de comparação, e consequentemente recuperação, torne-se mais simples. São então extraídos dos objetos os chamados *vetores de características*, que são representações simplificadas. A partir dos vetores de características dos objetos e de uma *métrica de similaridade* é realizada a consulta. A métrica de similaridade define o grau de (dis)similaridade entre dois vetores de características.

Dada a representação simplificada dos objetos (vetor de características) e a função de distância (métrica de similaridade) é necessário realizar uma filtragem eliminando os objetos não relevantes à consulta. Isto é realizado através de uma estrutura de acesso que organiza os objetos criando grupos de objetos mais similares, facilitando assim a tarefa da consulta.

Para avaliar uma estrutura de acesso, algumas características são fundamentais na escolha da estrutura mais adequada. Entre elas:

Dinamicidade. A estrutura de acesso deverá suportar inserções e remoções do banco de dados, idealmente sem grande perda de desempenho.

Escalabilidade. Dado que o tamanho dos repositórios das aplicações pode chegar a ser da ordem de milhões de objetos, a estrutura de acesso deverá manter o desempenho relativo a medida que o banco de dados cresce.

Independência da distribuição dos dados. A estrutura de acesso deverá ter um bom desempenho para várias possíveis distribuições de dados.

Uso do armazenamento secundário. Devido à enorme quantidade de dados, não é concebível armazenar todo o conjunto de dados na memória. A estrutura de acesso, então, deverá explorar eficientemente o armazenamento em memória secundária.

Tempo de processamento de consultas. Este talvez seja o principal item a ser considerado, dado que a principal motivação para o uso de estruturas de acesso é o de acelerar o acesso a dados, e por conseguinte, tornar o processamento de consultas mais eficientes.

Neste contexto, a motivação deste trabalho é a investigação e comparação de várias estruturas de acesso para dados multidimensionais (vetores de características), com respeito às características acima.

1.1.1 Problema Abordado

Dentre as aplicações que utilizam dados multidimensionais, a aplicação de banco de imagens foi escolhida para a avaliação das estruturas de acesso presentes nesta dissertação. O principal objetivo nesta aplicação é a realização de consultas de similaridade sobre as imagens do banco de dados. Ou seja, consultar quais imagens são semelhantes à uma dada imagem (ou esboço) em particular.

Para que a realização das consultas seja feita, é necessária a extração de informação das imagens. A partir dessa informação, ou vetor de características, é possível a criação de uma estrutura de acesso que consiga responder às consultas de forma mais eficiente do que se efetuando uma busca linear sobre os objetos.

Para efeitos dos estudos desta dissertação, assume-se que uma das principais informações extraídas das imagens para posterior comparação são cores. Formalmente, uma cor, ou luz colorida, nada mais é do que uma radiação eletromagnética $F(\lambda)$, onde o comprimento de onda $\lambda \in [380nm, 780nm]$ [Nie90]. Porém, a representação computacional de uma cor não é feita desta maneira, e sim na forma de conjuntos de vetores. Estes conjuntos de vetores formam um espaço multidimensional, onde um ponto neste espaço define uma cor. Como exemplos de espaços de cores temos o padrão RGB (do inglês Red, Green, Blue) e HSV (Hue – Matiz, Saturation – Saturação e Value – Valor) [WS82]. Um estudo realizado por Smith [Smi97] compara vários espaços de cores e conclui que o melhor ponto de partida é o espaço de cor HSV, de modo que doravante este espaço de cores será utilizado no restante desta dissertação.

Do ponto de vista da consulta de imagens, histogramas de cores são muito robustos em relação a escala, orientação, perspectiva e oclusão de uma imagem [SB91]. Os principais aspectos a serem considerados no desenvolvimento de um sistema de recuperação baseado em histogramas de cores incluem: o espaço de cor, a quantização das cores, a representação do histograma e as métricas de similaridade empregadas na comparação dos histogramas.

Os histogramas de cores são representações discretas da ocorrência das cores no espaço de cor escolhido. Por exemplo, um histograma de cores que utilize o espaço de cor HSV e que tenha sido quantizado em k *bins*[1] para H, l *bins* para S e m *bins* para V, é capaz de representar M cores, onde $M = k \times l \times m$. Desta forma, o histograma é representado por um vetor M-dimensional onde M é o número total de *bins* (cores). Esta representação é apresentada na Equação 1.1 a partir de uma imagem colorida I[x,y], onde X e Y são a largura e a altura da imagem respectivamente, em pixels.

$$h_c[m] = \sum_{x..0}^{X-1} \sum_{y..0}^{Y-1} \begin{cases} 1 & \text{se } I[x,y] = m \\ 0 & \text{caso contrário} \end{cases} \tag{1.1}$$

[1] termo inglês que indica a faixa discreta de representação para o eixo

Na Figura 1.1, são apresentadas três imagens com os respectivos histogramas de cores. Nesse exemplo, as cores das imagens foram quantizadas para apenas 16 cores. As duas primeiras imagens, além de serem semelhantes, possuem histogramas de cores semelhantes. A terceira imagem difere tanto na semelhança quanto nos histogramas de cores.

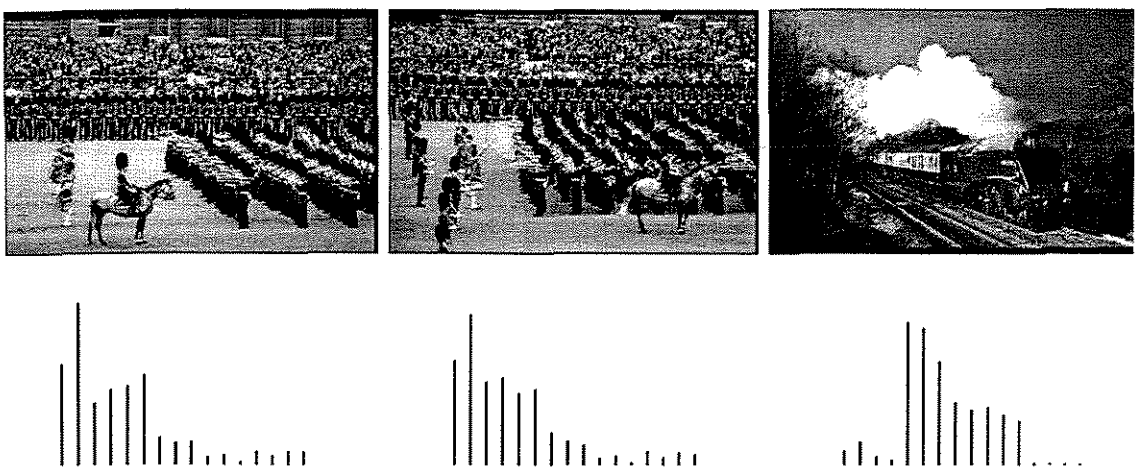


Figura 1.1: Exemplos de histogramas de cores de imagens

Para a comparação dos histogramas de cores é necessário a adoção de uma métrica de similaridade. Várias métricas são adequadas para a comparação de histogramas de cores, entre elas a distância da interseção de histogramas [SB91], a distância quadrática de histogramas [NBE+93] e a distância euclidiana de histogramas [Smi97]. Porém, nenhuma delas mostrou-se consistentemente melhor [SS94, Smi97].

Nesta dissertação a métrica utilizada na comparação de histogramas foi a distância Euclidiana. Dados dois histogramas, h_q e h_t, a distância é dada por:

$$D(q,t) = \sqrt{\sum_{m=0}^{M-1} (h_q[m] - h_t[m])^2} \tag{1.2}$$

Utilizando as três imagens dadas na Figura 1.1, considere como imagem consulta a primeira e as demais como possíveis respostas. As distâncias para as duas comparações são: $D(I_1, I_2) = 4337.9235$ e $D(I_1, I_3) = 31566.287$. Os valores obtidos com a métrica dizem respeito à dissimilaridade entre duas imagens. Portanto, no exemplo, a terceira imagem é mais dissimilar do que a segunda, em relação à primeira ($D(I_1, I_3) > 7 \times D(I_1, I_2)$).

1.2 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma:

- No Capítulo 2 são categorizadas e apresentadas algumas estruturas de acesso para a resolução de consultas em bancos de dados não lineares, i.e., dados em dimensões n-D, $n > 1$. As características, vantagens e desvantagens de cada uma das estruturas são exploradas neste capítulo.

- No Capítulo 3 é realizada a comparação das estruturas apresentadas no capítulo 2. A avaliação é feita utilizando-se vários critérios para determinar a estrutura mais adequada ao ambiente de banco de dados multidimensionais.

- No Capítulo 4 é realizada uma conclusão geral das avaliações das estruturas de acesso. Também é realizado um resumo das contribuições e direções futuras de pesquisa.

Capítulo 2

Revisão Bibliográfica

Conforme apresentado no Capítulo 1, aplicações recentes têm gerado a necessidade da utilização de estruturas de indexação especiais. Tipicamente, esses novos tipos de dados (aqui chamados vetores de características) são mapeados no espaço multidimensional tornando-se pontos nesse espaço. No caso dos histogramas de cores, as proporções de cada cor tornam-se uma coordenada em uma dimensão. Supondo que uma imagem seja quantizada em três cores, por exemplo, ela seria representada por um ponto no espaço tridimensional. Desta forma, imagens com distribuições de cores semelhantes estarão espacialmente próximas.

Dado o mapeamento dos vetores de características em pontos no espaço d-dimensional, é necessário realizar o agrupamentos dos pontos, para que a consulta seja facilitada. A Seção 2.1 mostra como a organização desses pontos é feita com relação às estruturas de indexação espacial tradicionais, e mostra também porque estas estruturas não são adequadas para o caso onde $d \gg 2$. As Seções 2.2 e 2.3 apresentam estruturas de indexação especializadas para o caso de espaços hiperdimensionais. Estruturas de indexação mais recentes, não estudadas nesta dissertação, são apresentadas na Seção 2.4.

2.1 Indexação Espacial Tradicional

As estruturas de indexação espacial tradicionais têm como objetivo resolver o problema de indexar dados com mais de uma dimensão. Nos casos onde os dados possuem apenas uma dimensão, as estruturas de indexação, como a B-tree [Knu72] por exemplo, utilizam uma ordem total sobre os dados para realizar agrupamentos e criar hierarquias. Porém, não existe uma ordem total sobre dados com mais de uma dimensão, que preserve a distâncias entre os dados no espaço original, o que impossibilita o uso das estruturas de indexação presentes em bancos de dados tradicionais.

As estruturas de indexação espacial procuram criar hierarquias sobre regiões no espaço

onde são encontrados objetos, no caso, pontos. A abordagem tradicional utiliza retângulos envolventes mínimos para delimitar regiões no espaço onde são encontrados pontos. Esse tipo de abordagem é tomado pelas estruturas de acesso da família R-tree, descritas a seguir.

2.1.1 Estrutura de Acesso R-tree

Uma R-tree [Gut84] é uma árvore balanceada similar à B-tree, contendo identificadores e ponteiros para os dados nos seus nós folha. Cada nó corresponde a uma página de disco, e a estrutura utiliza heurísticas durante a construção para que, em uma consulta espacial, o número de nós visitado, e por conseguinte o número de operações de I/Os, seja minimizado. O índice é totalmente dinâmico e inserções e exclusões podem ser intercaladas a consultas sem a necessidade de reorganização da estrutura.

Uma coleção de tuplas representa os objetos espaciais em um banco de dados espacial. Cada tupla possui um identificador único que pode ser usado para recuperar a mesma. Na R-tree, os nós folhas possuem entradas com a forma (I_o, o), onde o refere-se à tupla no banco de dados (p.ex, seu endereço no disco ou chave primária) e I_o é o mínimo retângulo d-dimensional envolvente (MBR—de *Minimum Bounding Rectangle*) do objeto espacial (o) indexado. Genericamente, I é definido como $I = (L_0, L_1, \ldots, L_{d-1})$ onde d é o número de dimensões e L_i é um intervalo fechado que descreve a projeção do objeto na dimensão i. Os nós internos contém entradas com a forma (I_p, p), onde p é o endereço do nó imediatamente descendente na sub-árvore e I_p cobre (exatamente) todos os retângulos das entradas deste nó.

Um nó da R-tree possui no máximo M entradas e no mínimo $m \leq \frac{M}{2}$ entradas, sendo m um parâmetro especificado. Além dessas propriedades, a R-tree também satisfaz as seguintes:

- Todo nó deve possuir entre m e M entradas, a menos que seja o nó raiz.
- O nó raiz deve ter no mínimo dois nós filhos.
- Todas as folhas estão no mesmo nível. A altura da R-tree é no máximo $\lceil \log_m(N) \rceil$, sendo N o número de objetos $(N > 1)$.

A Figura 2.1 mostra a estrutura de uma R-tree e ilustra a relação de conteúdo entre retângulos (pai contém o filho) e a possibilidade de sobreposição entre eles. Na figura, um exemplo de sobreposição é dado com os objetos m1 e m2.

Tipicamente, as consultas utilizando a R-tree são do tipo de "interseção", i.e., quais são os objetos indexados que tem uma relação de interseção com o MBR consultado. A pesquisa na R-tree é semelhante à pesquisa na B-tree. Em cada nó v do índice, todas

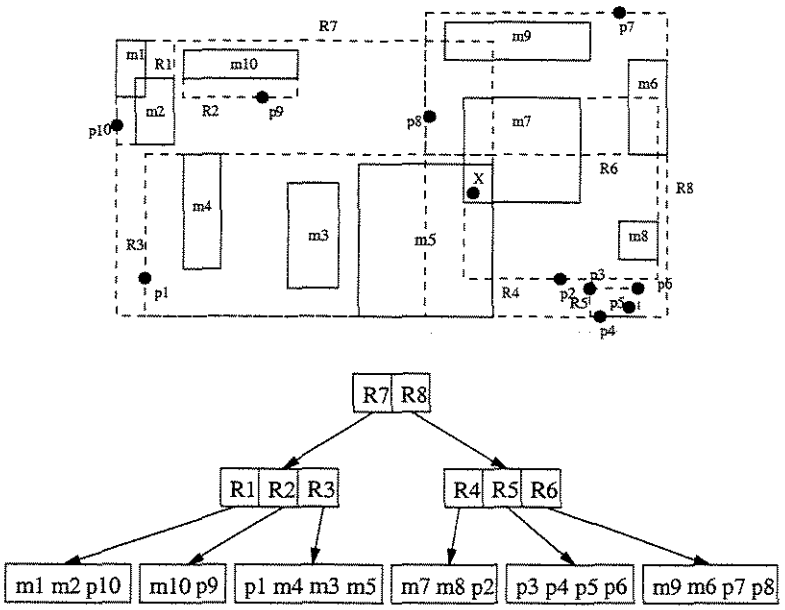


Figura 2.1: R-tree

as entradas são testadas quanto à interseção com relação ao MBR de pesquisa I_s. Para cada nó v_i tal que $I(v_i) \cap I_s \neq \emptyset$, o nó é visitado. Devido à definição da R-tree, podem haver diversos $I(v_i)$ que satisfaçam o predicado de pesquisa, nesse caso interseção. No pior caso, todas as páginas deverão ser visitadas. No exemplo da Figura 2.1, pi e X são pontos e mi correspondem a MBRs de polígonos. Assim sendo, uma pesquisa pontual X (ou seja, quais são os objetos que intersectam X) resulta em dois caminhos: R8 $\rightarrow$ R4 $\rightarrow$ m7 e R7 $\rightarrow$ R3 $\rightarrow$ m5.

Para inserir um objeto o, é inserido o mínimo retângulo envolvente I_o e o seu identificador na árvore. Ao contrário da pesquisa, um único caminho é percorrido da raiz até a folha. Em cada nível da árvore, é escolhido o nó filho v cujo MBR correspondente I_v necessita o menor aumento para conter o MBR do objeto o. No caso de haver mais de um MBR nessa situação, escolhe-se o menor MBR. Uma vez alcançado o nível de folha, o objeto é inserido. Caso isso requeira o aumento do MBR da folha, é feito o ajuste de forma apropriada e a modificação é propagada para os níveis superiores. Se não houver espaço suficiente na folha, os elementos da folha são divididos entre esta e uma nova folha criada. Novamente, os MBRs são ajustados e a divisão, se necessária, é propagada para os níveis superiores.

Note que o procedimento de inserção pode disparar a divisão (*split*) de um nó. Neste ponto, existem $M + 1$ entradas que precisam ser distribuídas entre dois nós. O *split* deve ser feito de forma que em uma próxima pesquisa seja minimizada a necessidade de analisar os dois nós. Uma vez que a decisão de visitar um nó depende do quanto o MBR sobrepõe a

região de consulta, a área total dos MBRs dos dois novos nós deve ser minimizada. Dada essa heurística, em [Gut84] Guttman propõe três técnicas de *split* sendo uma exponencial, outra quadrática, e outra linear. A técnica mais utilizada com a R-tree é a quadrática, e numa primeira etapa divide as entradas em dois grupos. Abaixo, são definidos os passos para a divisão de entradas em grupos:

1. *Escolha a primeira entrada de cada grupo:* Para cada par de entradas, construa o MBR e calcule a área não utilizada pelos dois MBRs, chamada de *área desperdiçada.* O par que tiver a maior área desperdiçada tem cada uma das entradas atribuída a um grupo. (Passo de complexidade quadrática.)

2. *Verifica-se se terminou:* Se um grupo tiver poucas entradas tal que as entradas restantes tenham que ser atribuídas ao grupo para atingir o mínimo m, atribua a este tais entradas. Se todas as entradas já foram atribuídas, termina.

3. *Seleciona entrada para atribuição:* Para cada entrada não atribuída, calcule o aumento necessário de área no MBR associado para atribuir a entrada a cada um dos grupos. Escolha a entrada que gera o menor aumento de área, e atribui a entrada ao nó correspondente. No caso de não haver necessidade de aumento, escolha o grupo com a menor área. Retorna ao passo 2.

Na segunda etapa do algoritmo, as entradas de cada grupo são armazenadas no nó que provocou o *split* e no novo nó criado. Os MBRs correspondentes a cada grupo são atualizados no nível superior.

Para excluir objetos, primeiro é feita uma consulta pontual para a localização do objeto em questão. Caso o objeto seja localizado, ele é excluído. Se a exclusão não deixar o nó com menos de m elementos (*underflow*), é verificado se o MBR pode ser reduzido. Caso positivo, a alteração é propagada para os níveis superiores da árvore. Quando a exclusão resultar em um *underflow* do nó, os seus elementos são copiados para uma área temporária, o nó é excluído, e os elementos órfãos são reinseridos na árvore. Uma outra alternativa é a junção dos elementos órfãos com elementos de nós irmãos. Em ambos os casos, é necessário a verificação dos MBRs envolvidos para possíveis ajustes.

2.1.2 Estrutura de Acesso R*-tree

A R*-tree [BKSS90], descendente direta da R-tree, foi projetada após a execução de inúmeros experimentos em dados padronizados sobre uma grande variação de dados, consultas e operações. Diferente da R-tree na heurística de otimização, que leva apenas em conta a área do MBR dos objetos internos a cada nó, a R*-tree leva em conta na sua heurística de otimização a área, a margem e a sobreposição entre cada MBR dos nós.

As propriedades utilizadas pela R*-tree são as mesmas da R-tree (Seção 2.1.1), ou seja, mesma forma dos nós folhas e internos, e mesma política quanto ao número de entradas em um nó (de m a M). A diferença entre as duas está relacionada com a heurística de otimização. Abaixo são relacionados os critérios levados em consideração pela R*-tree:

- *A área do MBR deve ser minimizada,* i.e. a área coberta pelo MBR do nó e não utilizada pelos MBRs das entradas do nó (espaço não utilizado) deve ser minimizada. Isso melhora o desempenho uma vez que a decisão de qual caminho tomar pode ser feita nos níveis mais altos da árvore.

- *A sobreposição entre os MBRs deve ser minimizada.* Com isso, o número de caminhos a serem percorridos em uma consulta diminui.

- *A margem do MBR deve ser minimizada.* A margem é a soma das arestas do retângulo. Considerando uma área fixa, o objeto com a menor margem é um quadrado. Portanto, minimizando a margem tornará a forma dos retângulos mais quadrada. Dado que quadrados são mais fáceis de empacotar, os MBRs de um nível irão gerar um MBR menor no nível acima.

- *O espaço de armazenamento deve ser minimizado.* A alta utilização do espaço de armazenamento reduzirá o custo da consulta por manter a altura da árvore pequena.

Estes critérios, necessários para um melhor desempenho nas consultas, estão relacionados entre si de forma complexa, tal que é impossível otimizar um deles sem influenciar os demais, o que causaria uma deterioração geral no desempenho. Portanto, a heurística utilizada foi baseada em diferentes experimentos, gerados a partir de um ambiente de testes.

Para resolver o problema da escolha do caminho mais apropriado para a inserção, a R-tree leva em consideração apenas o critério da área. Na R*-tree foram testados os critérios de área, sobreposição e margem em diferentes combinações, e o melhor resultado alcançado foi utilizado no algoritmo de inserção, definido a seguir:

1. Sendo N um ponteiro para um nó, N $\leftarrow$ nó raiz.

2. Se N é folha, retorna N. Se os ponteiros das entradas de N apontam para nós folha, determine o custo da sobreposição mínima. Escolha a entrada em N que o MBR necessita do menor aumento da sobreposição para acomodar o novo MBR. Em casos de empates, escolha a entrada cuja área necessita do menor aumento. No caso em que os ponteiros das entradas de N não apontam para nós folha, determine o custo da área mínima, ou seja, escolha a entrada cuja área necessita do menor aumento.

3. N ← nó apontado pelo ponteiro da entrada escolhida. Retorne ao passo 2.

A utilização do critério da sobreposição, nos casos dos nós que apontam para nós folhas, melhorou o desempenho das consultas em alguns casos. Nos demais casos, o desempenho permaneceu igual ao da R-tree, o que significou maior robustez para a R*-tree.

Outra modificação realizada foi a utilização de uma nova técnica de *split*. Nesta nova técnica estão diretamente relacionados os critérios de margem e sobreposição, e foi baseada no paradigma de *Planesweep* [PS85]. O procedimento a seguir descreve o algoritmo:

1. *Escolha a dimensão para split.* Para cada dimensão, ordena os MBRs de acordo com o lado do MBR na dimensão. A partir desta ordem, cria distribuições factíveis (respeitando m) para os dois grupos, e calcula a soma das margens das distribuições. A dimensão escolhida é a que possuir a menor soma das margens.

2. *Escolha a distribuição na dimensão.* Na dimensão escolhida para o *split*, entre as possíveis distribuições escolha a que tiver a menor sobreposição. Resolva empates escolhendo a distribuição com a menor área. (Passo de complexidade exponencial)

3. *Realize a distribuição.* Dada a distribuição escolhida no passo 2, realize a distribuição nos dois grupos.

A Figura 2.2 mostra a organização da R*-tree, utilizando o mesmo conjunto de dados da Figura 2.1 (R-tree). O algoritmo de *split* foi testado com valores de $m = 20\%$, 30%, 40% e 45% de M. Os melhores resultados foram obtidos usando $m = 40\%$.

Além da nova técnica de *split*, a R*-tree introduziu uma nova política chamada *reinserção forçada*. Nessa política, quando é necessário a realização de um *split* em um nó, esse *split* não é feito de imediato. Ao invés disso são removidas p entradas do nó e em seguida reinseridas na árvore. O valor sugerido para p é o de 30% de M.

A idéia da reinserção vem da exclusão da R-tree, onde quando o número de entradas de um nó fica inferior a m, as suas entradas são excluídas, o nó é excluído, e as entradas são novamente inseridas, no mesmo nível. Ao inserir novamente as entradas, o algoritmo de inserção pode escolher diferentes caminhos, procurando melhor acomodar as entradas. Isso beneficia principalmente os primeiros objetos inseridos na árvore, que geralmente são os que estão mais deslocados. O uso da política de reinserção forçada faz com que uma espécie de reorganização ocorra na árvore durante o seu crescimento.

Em resumo, a R*-tree difere da R-tree principalmente no algoritmo de inserção; a exclusão e a consulta são essencialmente as mesmas. Os resultados reportados em [BKSS90] mostram uma melhora de desempenho de até 50% sobre a R-tree original.

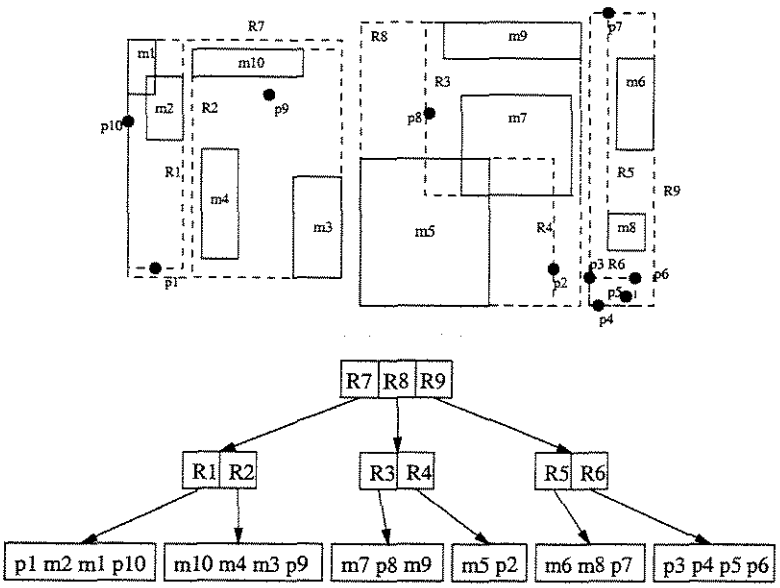


Figura 2.2: R*-tree

2.1.3 Problemas na Indexação de Espaço Hiperdimensional

A utilização de vetores de características é comum para a representação de objetos multimídia. Porém, como tais objetos possuem alta complexidade, os vetores de características gerados para esses objetos são grandes. No exemplo da indexação de imagens, com a utilização de histogramas de cores, é comum a necessidade da utilização de 64 ou até mais cores. O mapeamento direto para o espaço dimensional correspondente, no caso 64, acaba se tornando um problema para estruturas de indexação espacial tradicionais, projetadas para um número baixo de dimensões (2 ou 3 geralmente).

Estruturas espaciais tradicionais, como a R-tree e a R*-tree, têm seu desempenho deteriorado rapidamente com o aumento do número de dimensões. Essa deterioração ocorre principalmente por dois problemas:

1. *Problema do fanout:* Nesse contexto, *fanout* significa a quantidade de entradas que um nó é capaz de armazenar. As estruturas baseadas na R-tree, utilizam um retângulo (MBR) para agrupar os dados que estão em uma determinada sub-árvore, o que requer o armazenamento das coordenadas do MBR. Para duas dimensões, por exemplo, são necessárias as coordenadas de dois cantos opostos. Assumindo um sistema de coordenadas reais, seriam necessários 4 números reais, o que gastaria no mínimo 16 bytes. Além disso, cada entrada também armazena um ponteiro, para sua sub-árvore ou para seu objeto. O problema ocorre com o aumento do número de dimensões, que, devido ao custo da representação do MBR, torna o tamanho da entrada bastante grande. Com entradas grandes, poucas entradas são armazenadas

em um nó, o que torna a árvore menos larga e mais alta, aumentando o custo da consulta.

2. *Problema da sobreposição:* Em estruturas como a R-tree e a R*-tree, é possível ocorrer a sobreposição de MBRs. O problema é que quando grau de sobreposição dos MBRs de um mesmo nível é muito alto, durante uma consulta é necessário que vários caminhos sejam percorridos, até mesmo para consultas pontuais. As heurísticas que mantém essas estruturas não conseguem evitar o crescimento do grau de sobreposição entre os MBRs, quando o número de dimensões é grande.

Devido a estes problemas, as estruturas de indexação espaciais tradicionais não se adaptam adequadamente em ambientes onde o número de dimensões é alto.

2.2 Indexação de Espaço Hiperdimensional

Devido aos problemas encontrados na indexação de dados hiperdimensionais com estruturas tradicionais, novos tipos de estruturas foram desenvolvidas.

As estruturas apresentadas nesta seção possuem abordagem direcionada à solução dos problemas apontados na Seção 2.1.3. A estrutura apresentada na Seção 2.2.1, a SS-tree [WJ96], ataca diretamente o problema do *fanout*. Na Seção 2.2.2 é apresentada a SR-tree [KS97] que possui um maior enfoque ao problema da sobreposição.

2.2.1 Estrutura de Acesso SS-tree

A SS-tree [WJ96] foi uma das primeiras estruturas projetadas para suportar dados hiperdimensionais, diferente da família R-tree que foi projetada para baixas dimensões. Além disso, a SS-tree teve, um forte embasamento em consultas de similaridade. Através de uma nova forma de agrupar os dados, a SS-tree conseguiu criar agrupamentos em que os dados estão mais próximos espacialmente.

A similaridade entre os objetos na SS-tree é calculada através de métricas de similaridade baseadas na métrica Euclidiana ponderada, definida como:

$$D(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \mathrm{d}iag(\mathbf{w})(\mathbf{x} - \mathbf{y})} \tag{2.1}$$

Na equação, $\mathbf{x}$ e $\mathbf{y}$ são vetores de características e $\mathbf{w}$ é um vetor representando o peso relativo de cada dimensão na distância total. Cada elemento em $\mathbf{w}$ deve ser não negativo.

Os dados na SS-tree são agrupados com a utilização de esferas envolventes. O uso de esferas traz de imediato, duas vantagens:

1. *Reduz o problema do fanout:* Para a representação de uma esfera, é necessário apenas um ponto, no espaço D-dimensional, e o raio. Isso implica na redução de quase à metade do tamanho gasto pelos retângulos da R-tree. Portanto, o número de entradas que um nó é capaz de armazenar, praticamente dobra em relação à R-tree.

2. *Melhora o agrupamento dos dados:* Isso ocorre devido à esfera conseguir agrupar os dados em regiões de menor diâmetro. Conforme estudos realizados em [KS97], o diâmetro médio das esferas na SS-tree foi 40% menor do que o diâmetro dos retângulos (maior diagonal) na R*-tree. Uma vez que o diâmetro dos agrupamentos (ou seja a distância máxima entre dois objetos) tem grande influência em consultas de similaridade, a SS-tree foi favorecida.

Similar à R-tree, a SS-tree também utiliza uma hierarquia de regiões para organizar os dados. A diferença entre elas está no formato dos agrupamentos de dados, que, ao invés de retângulos, são utilizados as esferas. Portanto, o formato de uma entrada em um nó folha é: (S_o, o), onde o representa o objeto indexado e S_o representa a mínima esfera envolvente a tal objeto. Nos nós internos, uma entrada é formada por: (S_p, w, p), onde p é o endereço do nó imediatamente abaixo, S_p representa a esfera que contém todas as esferas contidas neste nó, sendo composta por um centróide e um raio. O número w representa o número total de objetos contidos na sub-árvore apontada por p. Na Figura 2.3 é apresentada a forma como os dados são agrupados na SS-tree.

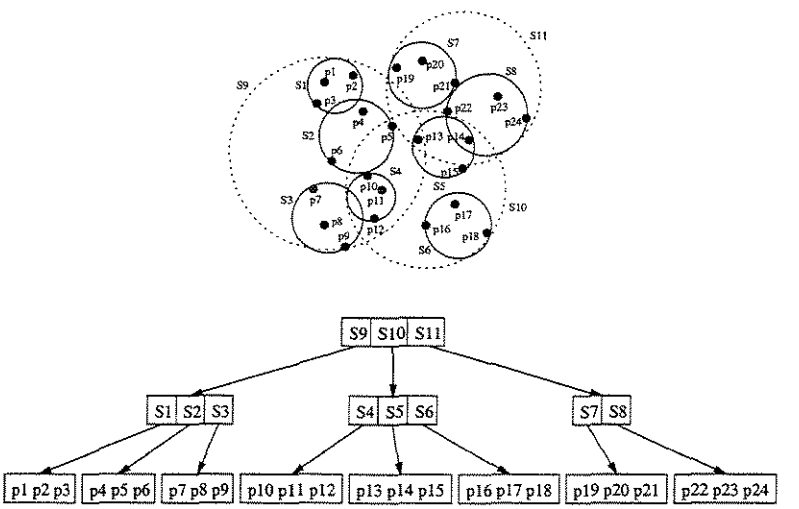


Figura 2.3: SS-tree

Na SS-tree, diferente da R-tree, as regiões não são necessariamente mínimas. Na R-tree, em todos os níveis da árvore, as representações são feitas com o uso de retângulos

mínimos (MBRs). Na SS-tree, o centróide das esferas é escolhido pela média ponderada entre os centróides dos nós filhos. O fator ponderador é dado pelo número de descendentes das sub-árvores dos nós filhos (número w).

O algoritmo de inserção da SS-tree é similar ao da R*-tree ao usar o conceito de reinserção forcada e cada nó possuir o mínimo de m e o máximo de M entradas (exceto pelo nó raiz). Na inserção de uma nova entrada, o algoritmo de inserção determina a sub-árvore mais adequada para acomodar a nova entrada escolhendo a sub-árvore cujo centróide é mais próximo. Quando um nó está cheio, e ainda não foi realizada a reinserção neste nível, a reinserção é feita utilizando, como na R*-tree, 30% das entradas para isso. No caso em que o *split* é inevitável, é localizada a dimensão cujos valores geram a maior variância para que seja feito o *split*. Após isso, divide-se as entradas de maneira a minimizar a soma das variâncias. O centróide escolhido para a nova esfera é uma média ponderada entre os centróides das esferas, levando em conta o número de objetos de cada sub-árvore. Após a determinação do centróide, o raio é determinado de maneira a conter as esferas do conjunto.

A SS-tree é uma estrutura construída para a resolução de consultas de similaridade, sendo as consultas de vizinho mais próximo as mais utilizadas. O algoritmo, discutido em [RKV95], pesquisa os k objetos mais similares a uma dada consulta, e é baseado em duas listas: uma contendo os nós que ainda não foram examinados (PR), e outra contendo o resultado parcial da consulta (NN). Neste ponto, uma nova métrica pode ser escolhida, desde que baseada na distância Euclidiana ponderada. O algoritmo da consulta é melhor descrito abaixo:

1. Insere-se o nó raiz em PR.
2. Retira-se de PR o nó cuja distância, entre os centróides, está mais próxima à consulta, e caso o nó seja:
   (a) *Nó interno:* insere-se os nós filhos em PR.
   (b) *Nó folha:* Os elementos são inseridos na lista NN de acordo com a similaridade com a consulta. São mantidos na lista NN apenas os k elementos mais similares.
3. Caso NN já possua k elementos, realiza-se poda de nós em PR, retirando os nós cuja distância seja maior que a distância do k-ésimo elemento.
4. Retorna para 2.

O algoritmo é encerrado quando não houverem mais nós em PR para serem examinados. Esse tipo de consulta é também amplamente utilizado em estruturas como a R-tree e a R*-tree.

A exclusão na SS-tree segue a mesma lógica da exclusão na R-tree, ou seja, é realizada uma consulta pontual para a localização do elemento, é realizada a exclusão, e são atualizadas as esferas de forma a manter a coerência da estrutura.

2.2.2 Estrutura de Acesso SR-tree

A SR-tree [KS97], como a SS-tree, é uma estrutura projetada para suportar dados com um grande número de dimensões. A ênfase dada à SR-tree foi a minimização da sobreposição de entradas de um mesmo nível. Conforme foi citado na Seção 2.1.3, o aumento da sobreposição causa a deterioração da estrutura, uma vez que em uma pesquisa, vários caminhos têm que ser seguidos.

No estudo realizado por Katayama e Satoh [KS97], são avaliadas as propriedades de retângulos (R*-tree) e esferas (SS-tree) no espaço dimensional alto. O primeiro ponto, a favor das esferas, é o custo de representação. Nas esferas é necessário pouco mais da metade do espaço necessário por um retângulo. Além disso, o uso de esferas faz com que sejam criadas regiões com diâmetro menor, ou seja, a distância máxima entre dois objetos é menor, chegando a 40% a menos nos nós folhas. Porém, os retângulos ocupam, em nós folhas, cerca de 2% do volume ocupado pelas esferas. Esse volume excessivo ocasiona uma grande sobreposição entre as esferas, o que deteriora o desempenho.

Considerando que esferas são vantajosas em termos de diâmetro, e retângulos são vantajosos em termos de volume, surgiu a idéia de combinar o uso de esferas e retângulos. Como suas propriedades são complementares, a interseção pareceu resultar em regiões de pequeno volume e pequena área.

A estrutura da SR-tree, como na R*-tree e na SS-tree, é uma hierarquia de regiões. Entretanto, a principal diferença da SR-tree é que uma região é determinada pela interseção de uma esfera e um MBR. Na Figura 2.4 é mostrada, a esquerda, uma visão global da estrutura. No centro e na direita, são mostrados os nós folhas e internos, respectivamente. Na parte inferior da figura, é mostrada como foi feita a organização dos nós da estrutura.

Um nó folha na SR-tree, é composto de entradas da forma: (P_o, o), onde P_o corresponde ao ponto no espaço D-dimensional que representa o objeto o. Como nas outras estruturas, o número de entradas de um nó folha respeita o mínimo de m_F e o máximo de M_F. Um nó interno, possui um formato diferente: (S_p, R_p, w, p), onde p é o endereço do nó imediatamente descendente, S_p corresponde à esfera deste nó, R_p corresponde ao MBR do nó, e w corresponde ao número total de objetos contidos na sub-árvore cuja raiz é representada por p. Nos nós internos também é respeitado o mínimo m_I e o máximo M_I de entradas.

O algoritmo de inserção da SR-tree é baseado no da SS-tree. Foi utilizado o algoritmo

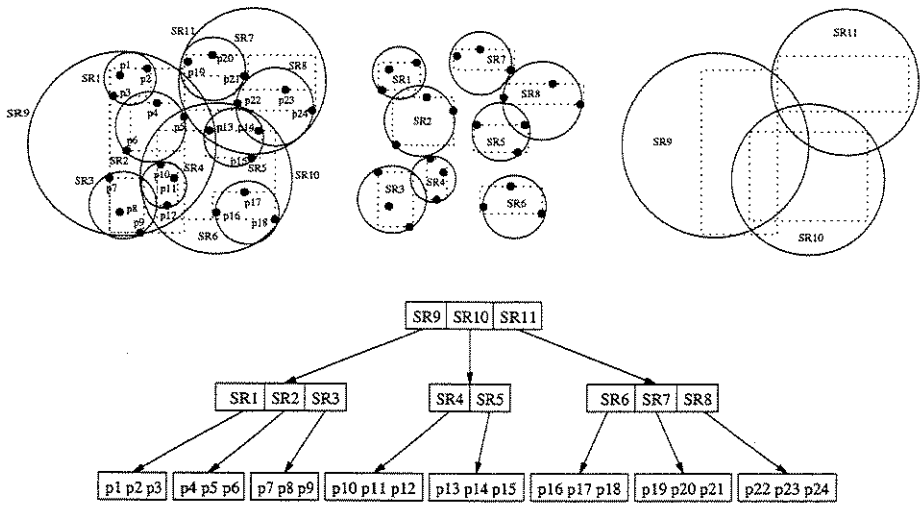


Figura 2.4: SR-tree

baseado no centróide da esfera, devido aos bons resultados em consultas de vizinho mais próximo. Portanto, o algoritmo determina a sub-árvore cujo centróide é mais próximo da nova entrada. Como na SS-tree, se o nó escolhido está cheio, a reinserção de entradas é feita. No caso em que o *split* é indispensável, quando a reinserção já ocorreu no mesmo nível, é escolhida a dimensão cuja variância é maior para a realização do *split*. A mudança no algoritmo ocorre na forma de atualizar as regiões. Na SR-tree é necessária a atualização do MBR além da esfera. A forma de atualizar o MBR é a mesma utilizada na R*-tree, porém a forma de atualizar a esfera é diferente. A esfera contém não apenas as esferas no nível abaixo, mas também os MBRs.

A consulta na SR-tree é também bastante similar à da SS-tree. A única diferença está na forma de calcular a distância de um ponto a uma região. Na SR-tree, essa distância é dada pela maior distância entre a distância da esfera ao ponto e a distância do retângulo ao ponto.

Na abordagem feita pela SR-tree, o objetivo de ter pequenos diâmetros e pequenos volumes em regiões foi alcançado. Isso foi demonstrado pela comparação do diâmetro e do volume da região da SR-tree, ou seja, da esfera e do retângulo. Além disso, ficou provado que a influência do problema do *fanout* pode não ser determinante no desempenho de uma estrutura.

2.3 Indexação Adimensional de Dados

Nas Seções 2.1 e 2.2 foram apresentadas estruturas espaciais para a indexação de vetores de características. Nessas estruturas, o agrupamento dos objetos é feito de acordo com

a localização espacial dos objetos. Problemas como o do *fanout* e da sobreposição são apontados, e possíveis soluções são mostradas.

Nesta seção é apresentado um conceito diferente de agrupar os objetos. Aqui, em vez de utilizar a localização espacial, é utilizada a distância entre objetos dada por uma métrica de similaridade. Nas chamadas árvores métricas, o objetivo é deixar os objetos similares (de acordo com uma métrica bem definida) próximos uns dos outros, e separar objetos com baixa similaridade. A consulta neste tipo de estrutura significa examinar as sub-árvores que possam armazenar objetos similares com a consulta.

Na Seção 2.3.1 é apresentada a M-tree [CPZ97], que segue os princípios das árvores métricas. Novas técnicas de *split* e uma forma de reorganização dinâmica da M-tree são apresentadas na Seção 2.3.2.

2.3.1 Estrutura de Acesso M-tree

A M-tree [CPZ97], como as outras estruturas apresentadas, é uma árvore balanceada. Os nós, em vez de representar em regiões no espaço dimensional, representam regiões no espaço métrico. Indexar um espaço métrico significa prover suporte eficiente para a resolução de consultas de similaridade.

Formalmente um espaço métrico é um par, $\mathcal{M} = (\mathcal{D}, d)$, onde $\mathcal{D}$ é o domínio dos vetores de características e d é uma função de similaridade. Na M-tree, a função de similaridade é considerada uma "caixa-preta", necessitando apenas respeitar as seguintes propriedades:[1]

- Simetria: $d(O_x, O_y) = d(O_y, O_x)$
- Não Negatividade: $0 < d(O_x, O_y) < \infty$, $O_x \neq O_y$ e $d(O_x, O_x) = 0$
- Desigualdade Triangular: $d(O_x, O_y) \leq d(O_x, O_z) + d(O_z, O_y)$

Os nós folhas da M-tree armazenam todos os objetos (vetores de características) indexados, enquanto os nós internos armazenam os *objetos roteadores*. O formato de uma entrada em um nó folha é dada por: (O_j, d, j), onde O_j representa o objeto (vetor de características), d representa a distância do objeto para seu pai (roteador), enquanto j representa a localização do objeto no banco de dados. As entradas dos nós internos tem o seguinte formato: (O_p, d, r, p), onde O_p corresponde ao objeto roteador, d corresponde a distância em relação ao seu pai, r indica o raio de cobertura (máxima distância) em relação aos filhos, e p aponta para o nó que estão os filhos deste roteador.

A heurística de inserção da M-tree procura manter a árvore compacta, minimizando os raios dos roteadores. Desta forma, para inserir um objeto n, o algoritmo escolhe o caminho

[1] Neste contexto, os vetores de características são chamados de objetos, sendo o vetor de características de um objeto x representado por O_x.

que evita qualquer aumento do raio de cobertura. Se mais de um caminho possui esta propriedade, é escolhido o roteador mais próximo. Quando não houver roteador com esta propriedade, $d(O_r, O_n) \leq r(O_r)$, é escolhido o roteador que precisa do menor aumento no raio. Ao inserir o objeto, pode ser necessário a realização de um *split*. Nessa situação, o nó N, cujo pai é N_p e o roteador é O_p, tem que ter suas entradas, conjunto $\mathcal{N}$, divididas em dois nós. Assim sendo, o *split* na M-tree ocorre da seguinte forma:

1. *Promoção:* Escolhe-se de $\mathcal{N}$ dois objetos para serem os novos roteadores, O_{p1} e O_{p2}. Nesse processo, existe a opção de *confirmar o pai* que significa utilizar o antigo roteador O_p como um dos escolhidos.

2. *Partição:* O conjunto $\mathcal{N}$ é particionado em dois subconjuntos, $\mathcal{N}_1$ e $\mathcal{N}_2$, procurando-se manter os raios pequenos. Armazena-se as entradas de $\mathcal{N}_1$ no nó N e as entradas de $\mathcal{N}_2$ em um novo nó N'.

3. *Atualização:* O atual roteador, O_p, é removido de N_p. Os dois novos roteadores, O_{p1} e O_{p2}, são inseridos em N_p referenciando os nós N e N', respectivamente. É realizada a atualização do raio do roteador de N_p, de modo a conter os dois novos nós.

4. *Verificação:* Caso o nó N_p não consiga armazenar as duas entradas, é ativado o procedimento de *split* para N_p.

A escolha de uma política de *split* determina a eficiência da M-tree tanto na construção quanto na consulta. Mais especificamente, a política de promoção que determina a forma e a eficiência de uma árvore. Algumas das políticas, eficientes em diferentes aspectos, são descritas a seguir:

`RANDOM` Esta política escolhe os objetos roteadores de forma aleatória. Apesar dela não parecer robusta, é a mais rápida e pode ser usada como referência para outras políticas.

`SAMPLING` Realiza a política `RANDOM` sobre uma amostra de objetos de tamanho $k > 1$. Para cada possível par de roteadores, gerado a partir dos $k(k-1)/2$ objetos, as entradas são distribuídas e os raios de cobertura estimados. O par com a menor soma dos raios, $r(O_{p1}) + r(O_{p2})$, é escolhido. A amostra padrão utilizada foi de $1/10$ do número de entradas do nó.

`M_LB_DIST` Escolhe para roteadores os objetos cuja distância entre eles é maximizada. Para isto, usa apenas as distâncias armazenadas, selecionando o mais próximo e o mais distante do atual roteador.

`m_RAD` Esta variação é a mais complexa em termos de distâncias computadas. Nela, para todos os possíveis pares de roteadores são calculados os raios, e o par com a menor soma dos raios é escolhido.

`mM_RAD` Semelhante à `m_RAD`, difere na escolha do par. Nesta variação, é escolhido o par cujo o maior dos dois raios é o menor entre os possíveis pares.

Entre as políticas de *split*, claramente a `m_RAD` e a `mM_RAD` são mais custosas computacionalmente. Entretanto, é possível a implementação destas políticas de forma otimizada. Para o cálculo do tamanho dos raios dos roteadores, é necessário o cálculo da distância dos dois roteadores para cada elemento do nó. Nos dois casos, durante o cálculo do raio do possível par de roteadores, é verificado se o novo par de roteadores já não é, mesmo antes do cálculo de todas as distâncias, uma opção pior que a opção atual de roteadores.

A realização de consultas de similaridade segue o mesmo princípio das demais estruturas. Dado um nó, o algoritmo de consulta escolhe o caminho cuja distância seja a menor entre o objeto de consulta e o roteador. Nas consultas de vizinho mais próximos, são armazenadas também duas listas, sendo uma para os nós ainda não examinados e outra para a resposta da consulta. Através do uso da propriedade da desigualdade triangular, é possível a realização de eliminações de nós não analisados.

No geral, a M-tree proporciona uma grande flexibilidade na métrica utilizada. As outras estruturas, que indexam o espaço, podem ter seu desempenho degradado ou até mesmo não serem capazes de utilizar funções de similaridade diferentes, principalmente as funções que não medem distâncias espaciais.

2.3.2 Novas Técnicas de Split e Reorganização para a M-tree

Na M-tree, como em qualquer estrutura, o desempenho da consulta depende fortemente da qualidade do agrupamento dos objetos similares. No caso das consultas de similaridade, é importante que os objetos estejam agrupados em relação a sua similaridade, o que não necessariamente corresponde a sua proximidade espacial.

O principal fator que determina um bom agrupamento dos objetos é a política de *split*. Entretanto, uma política de *split* não consegue utilizar critérios específicos devido ao não conhecimento do espaço métrico indexado [CPZ97]. Desta forma, o estudo de políticas alternativas de *split* torna-se um ponto importante.

Baseado em algumas heurísticas, políticas alternativas são propostas para a realização do *split*. Estas políticas, como as demais políticas da M-tree, são analisadas detalhadamente no Capítulo 3. Abaixo, são descritas as novas políticas de *split*:

`OVERLAP` A intuição nesta política, é escolher roteadores que minimizam a sobreposição

dos objetos de um roteador no outro. Dados os conjuntos de objetos já divididos entre os dois roteadores, denominados $\mathcal{N}_1$ e $\mathcal{N}_2$, essa sobreposição é definida como:

$$\texttt{OVERLAP}(\mathcal{N}_1) = \sum_{i=0}^{||\mathcal{N}_1||-1} \begin{cases} 1 & \text{se } d(O_i, O_{p2}) \leq r(O_{p2}) \\ 0 & \text{caso contrário} \end{cases} \quad (2.2)$$

É escolhido o par de roteadores cuja soma da sobreposição dos dois conjuntos é a menor. No caso de empate, é escolhido o par de roteadores com a menor soma dos raios.

`mM_MEAN` Nesta política, o objetivo é tentar escolher roteadores que melhor representem os objetos de um determinado conjunto. Isso é feito através da minimização da média das distâncias dos objetos ao roteador. Semelhante à `mM_RAD`, é escolhido o par de roteadores em que a maior das duas médias é a menor entre os outros pares.

`MM_RAD2` Esta política objetiva escolher roteadores em que a possibilidade de sobreposição entre eles é minimizada. Dado todos os possíveis pares de roteadores, é maximizando a diferença entre a distância do par de roteadores e o raio do maior deles.

Da mesma forma que algumas das políticas originais da M-tree (`m_RAD` e `mM_RAD`), as políticas propostas são de alto custo computacional. Entretanto, a otimização realizada naquelas políticas, só é possível de ser utilizada na proposta `MM_RAD2`. Isso acontece porque nas outras duas, `OVERLAP` e `mM_MEAN`, é necessário o cálculo do raio por completo para decidir entre dois pares candidatos a roteadores.

Outro fator que também pode influenciar na qualidade da consulta, é o quão robusta é uma estrutura a ponto de não ser influenciada pela ordem com que os objetos são inseridos. A reorganização periódica e completa do índice poderia ser uma solução, porém além de custosa, exige que o índice não seja usado enquanto isso é feito. Outra possibilidade é a realização de pequenas reorganizações no índice, de forma que não comprometa o seu uso.

Uma forma de reorganização é proposta para melhorar a eficiência dos roteadores. Durante a inserção de um objeto é chamado um procedimento para possivelmente modificar o roteador do objeto inserido. O objetivo desta reorganização, chamada de *Adaptive M-tree*, é de sempre manter os raios dos roteadores com o menor tamanho possível. O procedimento é descrito a seguir:

1. Após a inserção/alteração de uma entrada em um nó, é verificado se foi necessário o aumento do tamanho do raio de cobertura.

2. No caso desse aumento não ter sido realizado, retorna ao procedimento de inserção.

3. Se foi realizado aumento no raio, executa a re-promoção do roteador, elegendo o objeto com o menor raio de cobertura em relação aos demais objetos.

4. Retorna ao passo 1 caso haja atualização no nó pai.

Nos casos em que a técnica de *split* utilizada confirma o pai, os roteadores que foram confirmados durante a construção da árvore não são substituídos. Desta forma, é mantida a coerência necessária para que técnica de *split* funcione corretamente.

A diferença básica entre a M-tree e a Adaptive M-tree é a modificação do algoritmo de inserção. A modificação acarreta no aumento do custo de inserção devido ao aumento do número de computações de distâncias, necessárias para o recálculo dos roteadores. Por outro lado, os raios dos roteadores serão os menores possíveis, em uma dada árvore. A diminuição dos raios proporciona uma menor sobreposição entre os roteadores, o que implica em redução do número de nós visitados nas consultas.

2.4 Outras Estruturas de Indexação

Além das estruturas apresentadas neste capítulo, outras estruturas são adequadas para a indexação de dados hiperdimensionais. Algumas destas estruturas são apresentadas nesta seção.

A TV-tree [LJF94] melhora o desempenho da R*-tree para dados hiperdimensionais com a diminuição do número de dimensões e com a mudança das dimensões ativas. O número de dimensões é reduzido pela ordenação das dimensões baseado na sua importância e pela ativação apenas de poucas das mais importantes dimensões para a indexação. A mudança das dimensões ativas acontece quando um vetor de características numa sub-árvore tem as mesmas coordenadas na mais importante dimensão ativa. Então, esta dimensão se torna inativa e a próxima dimensão menos importante é novamente ativada para a indexação. Esta abordagem é efetiva para vetores de características que satisfazem as seguintes características: (1) dimensões podem ser ordenadas pela sua importância e (2) exista vetores de características que permitam a mudança das dimensões ativas. Conforme mencionado em [WJ96], a segunda condição nem sempre é válida para vetores de características com valores reais devido às suas coordenadas geralmente terem uma grande diversidade. Se a segunda condição falhar, a eficiência da TV-tree resulta apenas da redução das dimensões, o que pode ser facilmente aplicado em outras estruturas de indexação.

A X-tree [BKK96] é uma variante da R*-tree e melhora o desempenho da R*-tree com a utilização do *overlap-free split* e a utilização do mecanismo do supernó. O *overlap-free split* permite a pesquisa no espaço ser dividida em duas regiões disjuntas e melhora o desempenho de consultas pontuais. Um supernó é um nó de tamanho maior que é

utilizado para contornar o problema da sobreposição entre nós e melhorar o desempenho de leitura e escrita de nós.

A LSD^h-tree [Hen98] é uma estrutura típica de particionamento de espaço, que significa que quando há a necessidade de *split* de um nó, o espaço representado pelo nó é dividido, e os elementos são distribuídos entre os nós antigo e novo. Na LSD^h-tree são utilizadas heurísticas para melhorar a forma que os *splits* são realizados para o caso de dados hiperdimensionais. Da mesma forma, é proposta uma nova técnica para evitar a manutenção de espaços vazios, comuns em estruturas de particionamento de espaço.

A Hybrid Tree [CM99], ao contrário das outras não pode ser classificada como uma estrutura de particionamento de dados (DP — *Data Partitioning*) pura, como a R-tree, SS-tree e SR-tree, nem como uma estrutura de particionamento do espaço (SP — *Space Partitioning*) pura, como LSD^h-tree. Em vez disso, ela combina os aspectos positivos dos dois tipos de estruturas para conseguir melhor escalabilidade para dados hiperdimensionais. Outra diferença com a maioria das estruturas é que a Hybrid Tree suporta consultas com funções arbitrárias. Entretanto, como a maioria das estruturas de particionamento do espaço, as árvores geradas não são balanceadas quanto a altura, podendo ser influenciada pela ordenação do conjunto de dados.

Capítulo 3

Análise de Desempenho

Conforme visto no Capítulo 1, é comum em banco de imagens a criação de vetores de características para a representação, recuperação e comparação de imagens, sendo esses vetores indexados em estruturas de acesso especiais. Embora existam várias propostas para tais estruturas (Capítulo 2), dada a complexidade envolvida no processamento e consulta dos vetores de características, não é clara qual a melhor estrutura para esta tarefa.

Neste capítulo é apresentada uma comparação entre algumas das estruturas de acesso apresentadas no Capítulo 2. Esta comparação objetiva identificar as estruturas que apresentem melhor comportamento neste ambiente de indexação. Na Seção 3.1 são descritos os critérios utilizados para a comparação das estruturas. Na Seção 3.2 são apresentados os resultados obtidos, e as conclusões do estudo na Seção 3.3.

3.1 Critérios de Mensuração

Para que bons resultados sejam obtidos, a comparação entre as diferentes estruturas deve ser feita com a utilização de um ambiente homogêneo. Este ambiente deve utilizar o mesmo conjunto de dados, proporcionar as mesmas condições de desenvolvimento das estruturas, e medir os mesmos fatores.

Nesta seção é discutido como foi criado um ambiente homogêneo, que possibilitou a avaliação das estruturas de forma adequada. Outro fator também discutido foi o procedimento com que os testes foram realizados.

3.1.1 Conjunto de Dados

Na avaliação das estruturas é necessário um conjunto de dados para a construção e avaliação de instâncias das estruturas, as quais, no caso, são todas árvores. Além disso, essas

instâncias são utilizadas para medir o desempenho de consultas de similaridade. Por esses motivos, o conjunto de dados deve ser representativo em relação à aplicação escolhida, banco de imagens.

Um banco de 40.000 imagens foi utilizado com a aplicação exemplo. As imagens, de um CD de coleções de imagens, não pertencem a nenhum domínio específico (alguns exemplos são exibidos na Figura 3.1). A partir desta coleção de imagens, foram extraídos, como vetores de características, histogramas de cores.

Figura 3.1: Exemplos de imagens do banco de imagens

Para a extração dos histogramas de cores, as imagens foram transformadas para o espaço de cor HSV, e posteriormente quantizadas para um número reduzido de cores. A métrica de similaridade utilizada para a comparação dos histogramas foi a distância euclidiana (Equação 1.2). Todos os passos realizados no processamento das imagens, bem como a forma de comparar as imagens, são descritos na Seção 1.1.1.

A escolha da distância euclidiana (L_2) como métrica de comparação entre as imagens foi realizada devido a limitações de algumas estruturas. Com exceção da M-tree, as estruturas comparadas foram desenvolvidas para a utilização da métrica L_2 em específico. Um relaxamento dessa restrição foi feito nas estruturas SS-tree e SR-tree, onde é utilizado a L_2 ponderada — pesos podem ser dados a cada dimensão. A adaptação de funções de similaridade arbitrárias nas estruturas é uma tarefa complicada e nem sempre possível. Um exemplo de adaptação sobre as estruturas da família R-tree é feito em [SK97]. Portanto, devido à inflexibilidade das estruturas, exceto pela M-tree, a métrica L_2 foi escolhida para a comparação entre as estruturas apresentadas.

Além do conjunto de dados de imagens, foi utilizado um conjunto de dados uniforme. Gerado sinteticamente, esse conjunto de dados é formado por vetores de "coordenadas" reais pertencentes ao intervalo fechado $[0, 1]$. A distribuição dos valores das "coordenadas" é realizada de forma uniforme.

3.1.2 Ambiente de Análise

A comparação de estruturas freqüentemente é falha quando baseada em diferentes implementações, onde o resultado pode ser determinado pela forma com que as estruturas

foram implementadas. Isso pode ocorrer principalmente quando a implementação das estruturas é feita por terceiros. Para tentar evitar esse problema, foi usado o ambiente de desenvolvimento de estruturas de indexação GiST (*Generalized Search Tree*) [HNP95].

O GiST é uma estrutura de dados extensível, que permite que sejam desenvolvidos índices sobre qualquer tipo de dados, suportando qualquer tipo de pesquisa sobre eles. Para a implementação de estruturas sobre o GiST, é necessário a definição de quatro métodos:

- O método *consistent* guia as consultas, respondendo quando a consulta deve seguir a partir de uma dada sub-árvore;
- Através do método *union* é gerada uma entrada que representa um conjunto de entradas (p.ex., relação entre nó pai e filhos);
- O custo gerado por uma possível inserção de uma entrada em um nó é dado pelo método *penalty*; e
- O último método, *picksplit*, determina os dois conjuntos de entradas resultantes quando é necessária a divisão de um nó.

A atual versão do GiST, a 2.0, já inclui código fonte das estruturas R-tree, R*-tree, SS-tree e SR-tree. Para completar o conjunto de estruturas do estudo, foi re-implementada a estrutura M-tree, após o estudo do código fonte da M-tree (GiST versão 0.9), fornecido pelos autores. Para esta última, foram implementadas todas as técnicas de *split* descritas nas Seções 2.3.1 e 2.3.2.

Além das estruturas apresentadas, foi implementada a busca linear de dados, que não cria nenhuma estrutura para tentar acelerar a consulta. Nela, são apenas armazenados os vetores de características e os identificadores dos objetos. Quando uma consulta é realizada, o índice é percorrido por completo. O custo da consulta no índice linear pode ser considerado como o maior custo admitido. Por não utilizar nenhuma forma de organização dos dados, não foi possível utilizar o GiST para a sua implementação, tendo sido escritos programas em C para isso. Na Seção 3.2.5 é realizada a análise dos resultados obtidos com esta abordagem.

Finalmente, para a realização das medições, foi utilizado um computador PC compatível com CPU Pentium II/300 Mhz, com 192 Mb de memória RAM, disco rígido sobre interface SCSI, e rodando o sistema operacional Linux (kernel 2.0.36). Durante as avaliações foi assegurado que nenhum processo adicional fosse executado, evitando um ambiente concorrente. Do mesmo modo, também foi verificado que o sistema operacional não realizou *swap* de páginas de memória durante a execução dos testes, completando assim um ambiente praticamente sem influências externas.

3.1.3 Parâmetros da Análise

A avaliação realizada sobre as estruturas procurou medir o comportamento de cada estrutura variando os seguintes parâmetros:

- Conjunto de dados;
- Tamanho de página de disco; e
- Número de imagens similares retornadas por consulta.

Com o cruzamento desses parâmetros, vários ambientes foram criados, o que proporciona uma maior robustez dos resultados obtidos.

Os conjuntos de dados de imagens e uniforme foram gerados com 16, 32 e 64 dimensões. Com essa variação no número de dimensões, é avaliada a escalabilidade das estruturas quanto ao crescimento do número de dimensões.

Como um nó de uma árvore é armazenado exatamente em uma página de disco nas estruturas analisadas, a variação no tamanho da página de disco implica diretamente na mudança da capacidade do nó. A maioria das publicações utiliza, nos estudos práticos, páginas de disco de 4 Kb, porém, conforme o estudo de Gray e Graefe [GG97], o tamanho das páginas de disco para os índices atuais deveria provavelmente ser de 16 Kb. Assim sendo, na análise das estruturas foram utilizados páginas de disco de 4 Kb, 8 Kb e 16 Kb. O tamanho da página de disco é configurado no GiST, sendo essa mudança transparente para as estruturas.

Diferente da maioria dos estudos, o tempo real de processamento foi utilizado nas avaliações das consultas sobre os índices além do número de acessos a disco. Isto foi feito devido ao fato de que a computação das distâncias entre objetos de alta dimensão requer um esforço computacional não trivial, e começa a influenciar efetivamente no tempo de resposta, tornando os resultados de número de acesso não necessariamente representativos por si sós.

Na avaliação das estruturas foram utilizadas consultas de similaridade de vizinhos mais próximos. Neste tipo de consulta, são retornados os k vizinhos mais próximos a um dado ponto. Para avaliar o impacto ocasionado com o número de vizinhos retornados, foram realizadas consultas retornando 11, 21, 51 e 101 vizinhos. Para estas consultas, os elementos utilizados são escolhidos a partir do próprio conjunto de dados , o que implica que para saber os 20 mais similares, é necessário buscar os 21 mais similares.

Para cada dimensão, tamanho de página de disco e conjunto de dados, foram construídas 10 árvores, variando em cada uma delas, a ordem da inserção dos objetos. Na construção de cada árvore foi medido o tempo de construção em segundos e o tamanho da árvore em bytes. A Figura 3.2 mostra o número de árvores avaliadas. Foram avaliadas as

estruturas R-tree, R*-tree, SS-tree, SR-tree e M-tree, sendo esta avaliada em 20 variações (incluindo as técnicas originais e as propostas nesta dissertação), embora nem todos os resultados obtidos usando-se todas as 20 variações sejam exibidos nesta dissertação, devido a, p.ex., "congestionamento" dos gráficos e similaridade de resultados.

Estruturas analisadas (24)	×	Conjunto de dados (2)	×	Dimensões (3)	×	Página de disco (3)	×	Construções aleatórias (10)	=	Árvores construídas (4.320)

Figura 3.2: Número de árvores construídas

Para cada uma das árvores construídas, foram realizadas 150 consultas de vizinho mais próximo, a partir da escolha aleatória de 150 elementos do conjunto de dados. Cada uma dessas consultas foi realizada retornando 11, 21, 51 e 101 vizinhos mais próximos, sendo, para cada uma delas, medidos o tempo de execução em segundos e o número de páginas de disco acessadas. A Figura 3.3 mostra o número total de consultas realizadas.

Árvores construídas (4.320)	×	Tipos de consultas de vizinho mais próximo (4)	×	Número de consultas (150)	=	Consultas realizadas (2.592.000)

Figura 3.3: Número de consultas realizadas

3.2 Resultados Experimentais

Após definir as estruturas, os conjuntos de dados, e a forma de mensuração, as estruturas foram submetidas às avaliações. Na estrutura M-tree foi realizado um estudo mais detalhado, que é descrito nas Seções 3.2.1 e 3.2.2. A comparação geral entre as estruturas, separada pelas avaliações de construção e consulta dos índices, é descrita nas Seções 3.2.3 e 3.2.4, respectivamente. Na Seção 3.2.5 são apresentados os resultados obtidos com a busca linear, i.e., sem o uso de uma estrutura de indexação propriamente dita.

3.2.1 Comparações da M-tree— *Splits* Originais

Na M-tree foi estudado um grande número de variações de *split*, além da técnica de reorganização proposta na Seção 2.3.2. Para facilitar a visualização dos resultados, as análises foram separadas em duas seções. Nesta seção é realizado o estudo das técnicas de *split* originais, descritas na Seção 2.3.1. A avaliação das novas técnicas de *split* e da proposta de reorganização para a M-tree é feita na seção seguinte.

As técnicas de *split* originais avaliadas são: RANDOM, SAMPLING, M_LB_DIST, m_RAD e mM_RAD. Para cada técnica de *split* foram avaliadas variações confirmando o pai, cujo nome recebe o sufixo _1, e não confirmando o pai, que recebe o sufixo _2. A diferença entre as duas variações está na determinação dos novos roteadores durante o *split*. Quando o pai é confirmado, apenas um roteador precisa ser escolhido. Portanto, as variações que não confirmam o pai são, a princípio, mais custosas em termos de processamento. Esse custo é evidenciado em técnicas como a m_RAD e mM_RAD, onde a complexidade torna-se quadrática em relação ao número de elementos do nó.

Os gráficos mostrados a seguir foram gerados a partir dos resultados obtidos com o conjunto de dados de imagens. Nos casos em que a dimensão está em análise, o tamanho da página de disco foi mantido em 8 Kb, enquanto nos casos em que o tamanho da página de disco está sendo avaliado, o número de dimensões foi mantido em 32. Devido ao grande número de técnicas presentes nas análises, as técnicas RANDOM_1, SAMPLING_2 e M_LB_DIST_2 foram omitidas para que os gráficos ficassem mais claros, uma vez que tais técnicas não obtiveram bons resultados em relação às respectivas variações.

Tempo de Construção do Índice

A Figura 3.4 mostra, conforme esperado, as técnicas de *split* m_RAD_2 e mM_RAD_2 como as mais lentas, sendo isto devido ao número de distâncias computadas, que é quadrático em relação ao número de entradas no nó. Com o aumento do número de dimensões, as entradas ficam maiores e menos entradas são armazenadas em um nó, diminuindo o esforço computacional por nó, ocasionando assim o comportamento decrescente dessas técnicas. Entretanto, a técnica mM_RAD_1, apesar do número de computações maior, obteve um desempenho não muito inferior às técnicas menos complexas. Isto acontece devido ao tamanho do índice gerado ser compacto em relação às outras técnicas, o que ocasiona em um número baixo de nós processados. As técnicas de baixa complexidade foram as mais rápidas, sendo a técnica RANDOM_2 a mais rápida.

O impacto gerado pelo aumento da capacidade, i.e., tamanho, de um nó é mostrado na Figura 3.5. De forma geral, as técnicas não sofreram grande variação no tempo de construção do índice. Apenas as técnicas mais complexas, m_RAD_2 e mM_RAD_2, sofrem com o aumento da página devido a quantidade de computações realizadas. Novamente, as variações da técnica RANDOM foram as mais rápidas por praticamente não serem influenciadas pelo número de elementos do nó.

Tamanho do Índice

As Figuras 3.6 e 3.7 mostram o comportamento das técnicas de *split* em relação ao tamanho do índice. Com o aumento do número de dimensões, o espaço gasto pelos índices

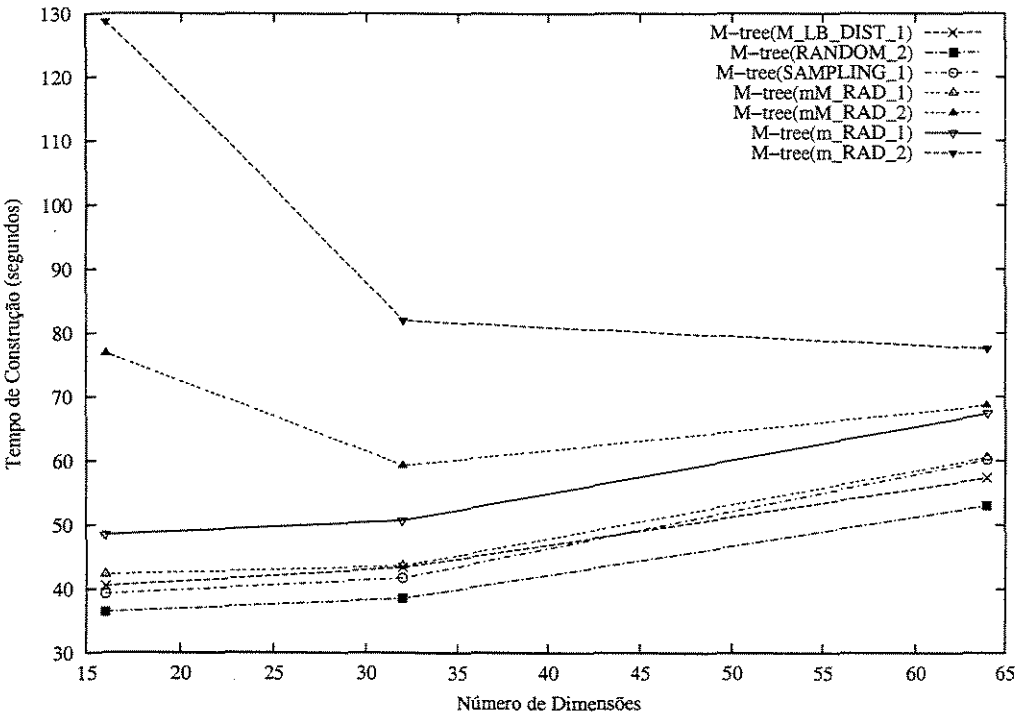


Figura 3.4: Tempo de construção do índice *vs.* número de dimensões

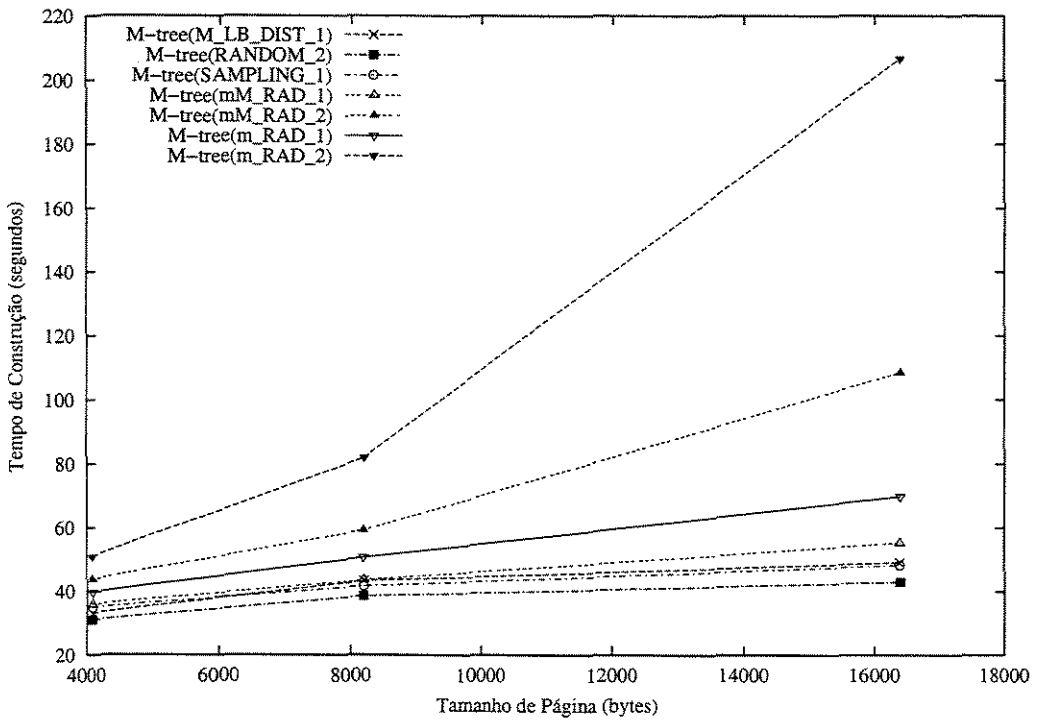


Figura 3.5: Tempo de construção do índice *vs.* tamanho da página de disco

cresceu linearmente (Figura 3.6). Entretanto, entre as técnicas mais e menos compactas, RANDOM_2 e m_RAD_2 respectivamente, há uma diferença média de 112%. Entre as técnicas mais complexas, as variações da técnica mM_RAD obtiveram desempenho semelhante à RANDOM_2, enquanto as variações da m_RAD obtiveram o pior desempenho.

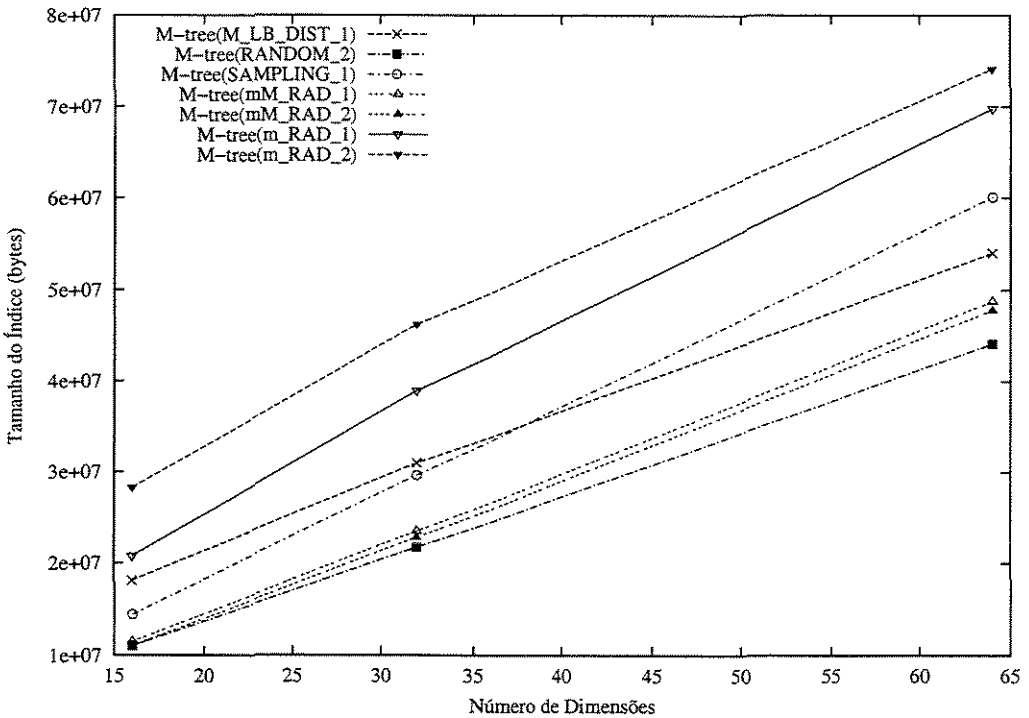


Figura 3.6: Tamanho do índice *vs.* número de dimensões

Com o aumento do tamanho da página, as variações das técnicas RANDOM, SAMPLING e mM_RAD diminuíram o tamanho dos índices gerados (Figura 3.7), o que indica uma melhor utilização do espaço das páginas de disco. Nas demais técnicas, M_LB_DIST e m_RAD, o tamanho do índice cresce com o aumento do tamanho da página, evidenciando que os *splits* são feitos de forma desbalanceada, o que acarreta numa baixa utilização do espaço de armazenamento.

Tempo de Consulta

Juntamente com a avaliação da construção dos índices, foi realizada a medição dos tempos de consultas, onde são mostradas figuras referentes à consultas dos 21 vizinhos mais próximos. Na Figura 3.8 é mostrado o tempo médio para a realização de uma consulta em relação ao número de dimensões. Claramente, a técnica RANDOM_2 teve o pior desempenho, em média 60% mais lento que as demais. Entre as melhores, nenhuma das técnicas foi

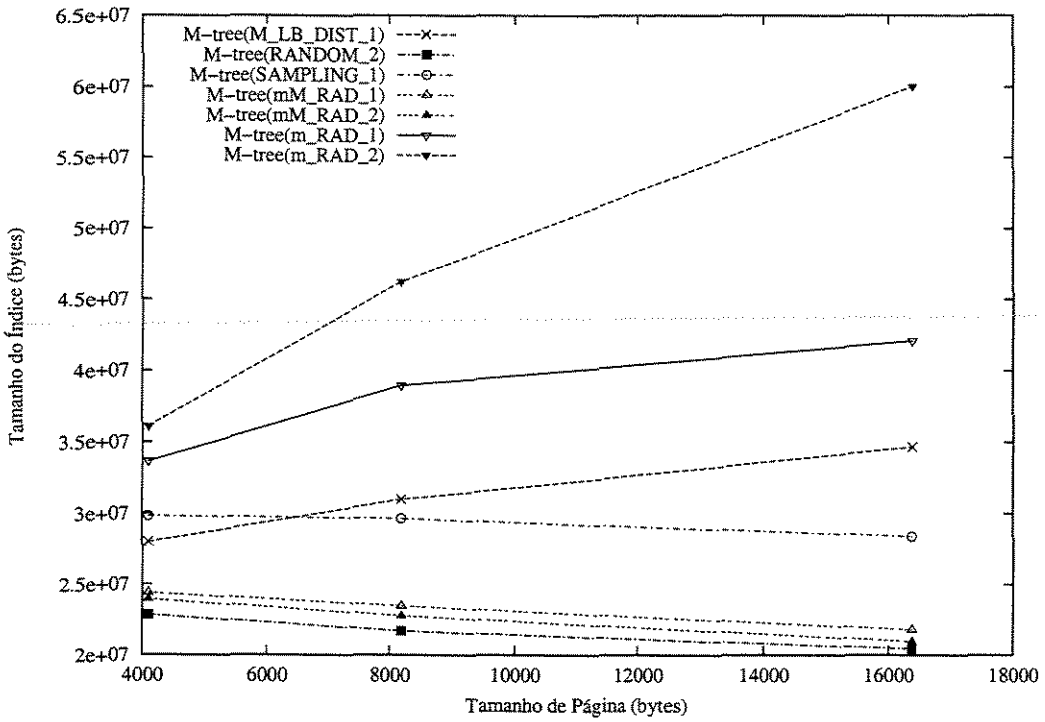


Figura 3.7: Tamanho do índice *vs.* tamanho da página de disco

superior em todas as dimensões. Entretanto, a técnica que apresentou os melhores resultados médios foi a `mM_RAD_1`. Entre as técnicas menos complexas, a técnica `M_LB_DIST_1` obteve melhor resultado médio, e foi cerca de 5% pior que a `mM_RAD_1`.

Na Figura 3.9 é mostrado o comportamento das técnicas em relação ao tamanho da página de disco na consulta. Para todas as técnicas, o aumento do tamanho da página de disco implicou na diminuição do tempo de consulta, notadamente na técnica `M_LB_DIST_2` (não mostrada), que reduziu 43% o tempo. A melhor técnica em média, a `mM_RAD_1`, reduziu cerca de 27% o tempo da consulta. Confirmando o pior desempenho nas consultas, a técnica `RANDOM_2` teve a menor redução do tempo de consulta, cerca de 18%.

O comportamento das técnicas em relação ao tamanho da consulta realizada é mostrado na Figura 3.10. O crescimento do tempo foi proporcional entre as técnicas, e praticamente não houve mudança nos resultados de desempenho. A figura foi construída utilizando dados de 32 dimensões e 8 Kb de página de disco, o que confirmou a superioridade da técnica `mM_RAD_1`.

Números de Páginas Acessadas Durante Consulta

Conforme discussão levantada anteriormente, o número de páginas acessadas na consulta não mais pode ser considerado como único argumento de comparação. Isto acontece

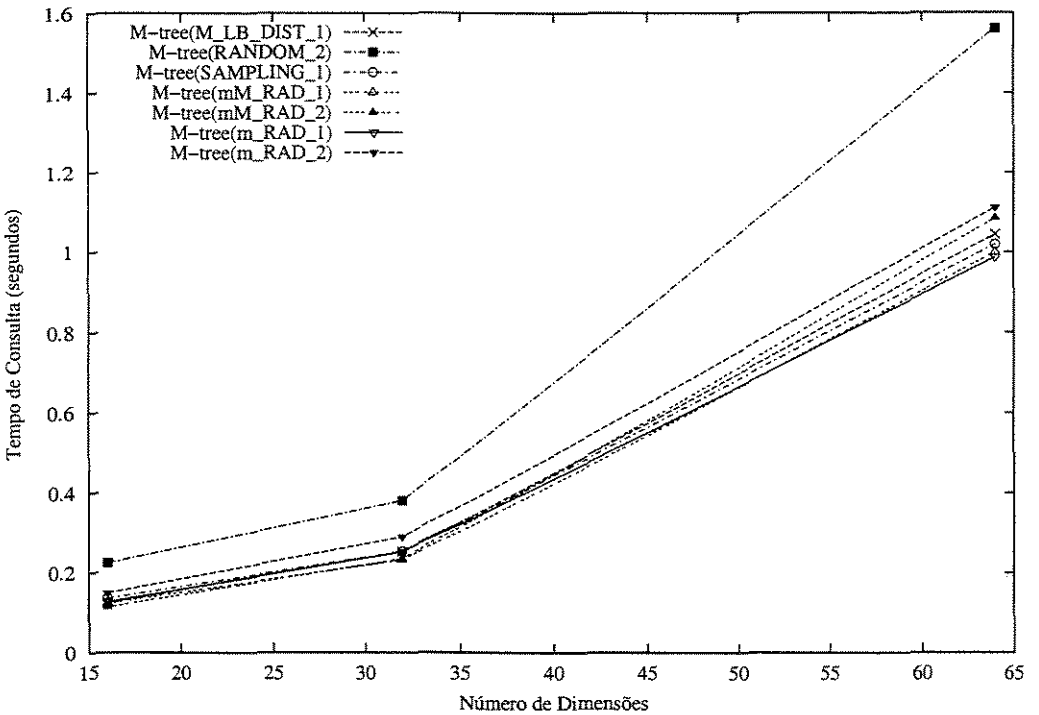


Figura 3.8: Tempo de consulta *vs.* número de dimensões

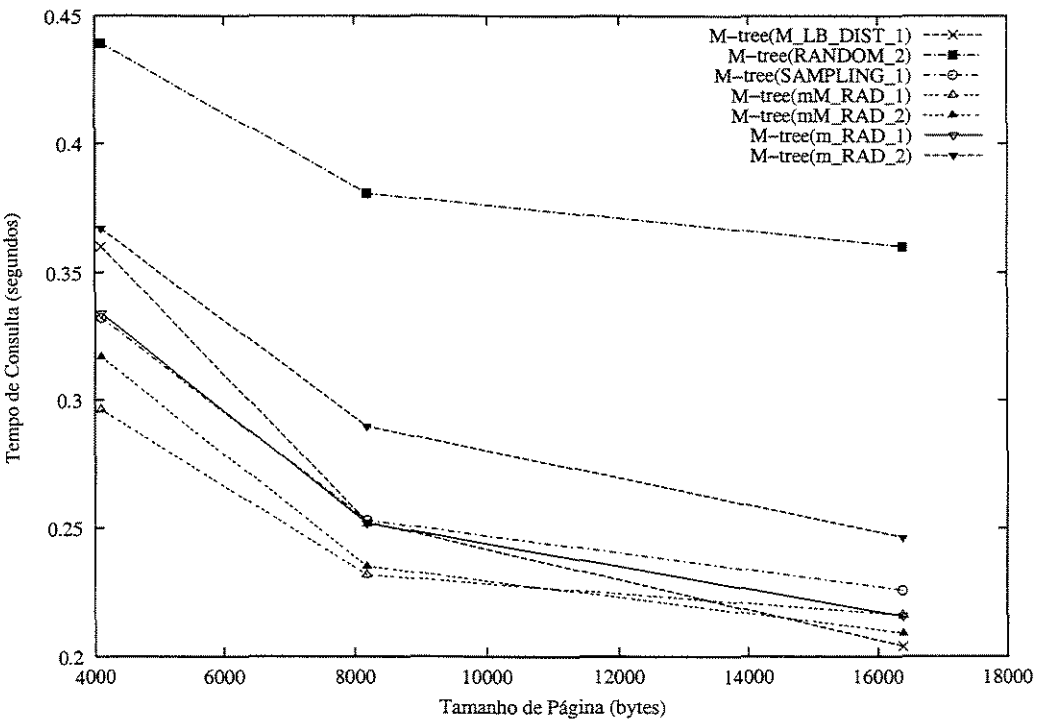


Figura 3.9: Tempo de consulta *vs.* tamanho da página de disco

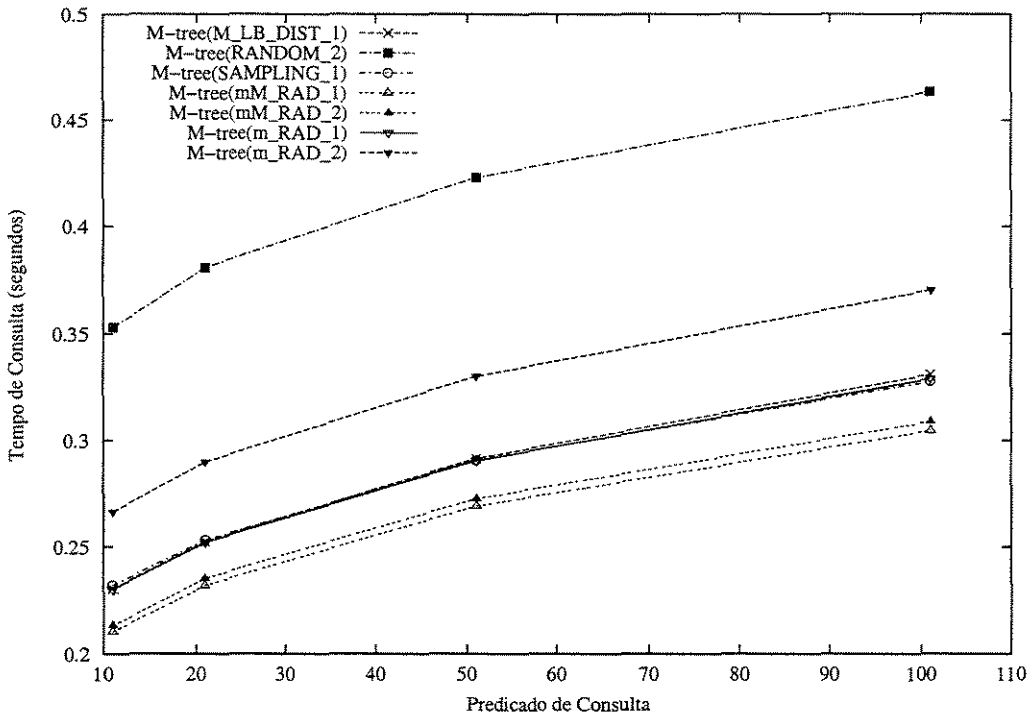


Figura 3.10: Tempo de consulta *vs.* número de vizinhos retornados

devido ao alto custo computacional existente para processar os vetores de características. A Figura 3.11 mostra o comportamento das técnicas em relação ao número de páginas acessadas nas consultas. Na figura, as variações da técnica `mM_RAD` mostram superioridade de cerca de 20% sobre as demais técnicas, mas quando é utilizado o tempo real da consulta, Figura 3.8, essa superioridade é de apenas 3%. Em um outro exemplo, a técnica `m_RAD_1` acessa, na média, cerca de 30% mais páginas que a técnica `mM_RAD_2`, e é, na verdade, apenas 3% mais lenta. Essa distorção acontece principalmente devido a baixa utilização do espaço de armazenamento, que ocasiona em um menor número médio de entradas por nó. Como, na resolução de uma consulta, é necessário a comparação com todos os elementos de um nó para a escolha do caminho, índices maiores são beneficiados, obtendo bons desempenhos na consulta.

Conclusão — *Splits* Originais da M-tree

Uma vez que não houve técnica que tenha se mostrado superior em todos os aspectos analisados, foi realizado uma ponderação dos resultados para a escolha da melhor técnica. Os aspectos utilizados na comparação foram: tempo de construção, tamanho do índice e tempo de consulta. A Tabela 3.1 mostra a comparação das técnicas em relação a estes três aspectos, e também o número de acesso a páginas, que foi colocado para efeito de

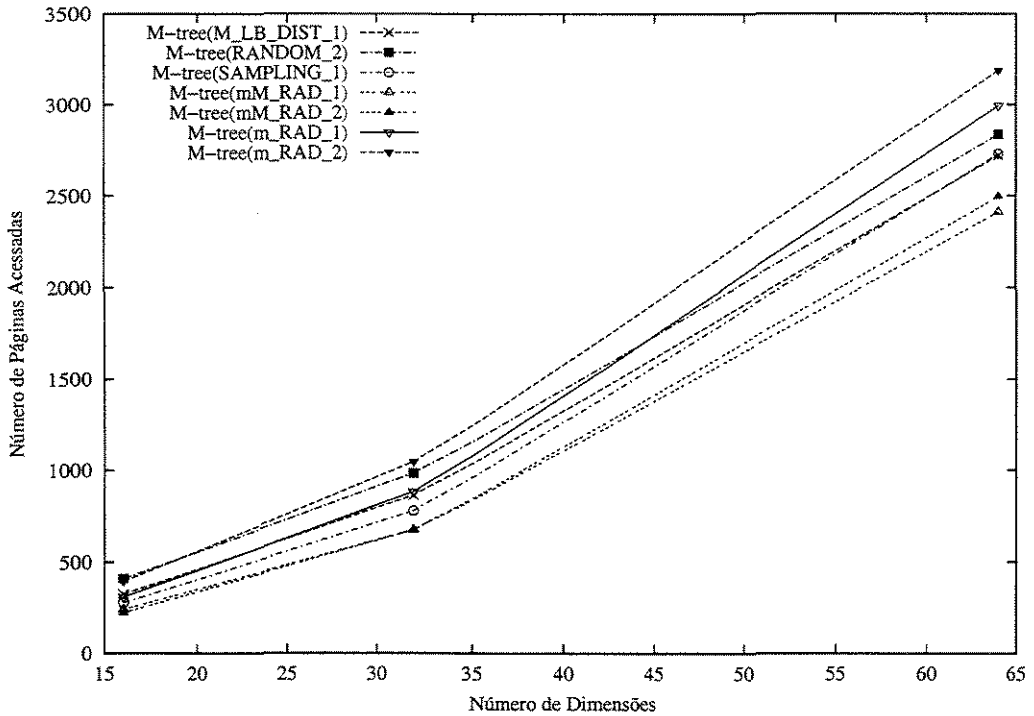


Figura 3.11: Páginas acessadas na consulta *vs.* número de dimensões

comparação com o tempo de consulta. Para a construção da tabela, cada aspecto foi analisado em relação aos resultados obtidos nas avaliações de 16, 32 e 64 dimensões para o conjunto de dados de imagens. O melhor resultado médio, sobre todas a dimensões, entre as técnicas foi representado com o valor 1, e os demais valores representam o custo em relação à melhor técnica. O custo de uma técnica B em relação a uma técnica A, calculado com os resultados em cada dimensão (representado por R_d), é apresentado na fórmula abaixo:

$$Custo(B,A) = \left(\frac{R_{16}(B)}{R_{16}(A)} + \frac{R_{32}(B)}{R_{32}(A)} + \frac{R_{64}(B)}{R_{64}(A)} \right) /3 \tag{3.1}$$

Através desta tabela, pode ser avaliado a quantidade percentual que uma técnica é inferior à melhor, considerando determinado aspecto. O cálculo do desempenho médio foi realizado pela média entre os três primeiros aspectos.

A partir dos dados da tabela e dos resultados mostrados nos gráficos desta seção, é realizado abaixo o comentário das técnicas de *split*:

- `RANDOM`: foi a técnica que mais se destacou nas avaliações de construção. Devido a sua natureza aleatória, o tempo gasto em *splits* é baixo. Além disso, os *splits* gerados por essa técnica são bastante balanceados, o que implica numa alta utilização do

	tempo construção	tamanho índice	tempo consulta	acesso consulta	desempenho médio
RANDOM_2	1	1	1.676217	1.474787	1.225406
SAMPLING_1	1.097783	1.345456	1.071615	1.163012	1.171618
M_LB_DIST_1	1.105606	1.431513	1.049164	1.274078	1.195428
m_RAD_1	1.305372	1.753759	1.036042	1.296824	1.365058
m_RAD_2	2.374598	2.125864	1.191207	1.530546	1.897223
mM_RAD_1	1.144338	1.076927	1	1.014877	1.073755
mM_RAD_2	1.647227	1.045599	1.009663	1	1.234163

Tabela 3.1: Comparação das técnicas de *split* originais da M-tree

espaço de armazenamento. O tempo baixo para a construção dos índices é devido a esses dois fatores. Entretanto, por não haver critério na escolha dos roteadores, são criados roteadores com raio de cobertura muito grande. Isto ocasiona um alto grau de sobreposição entre roteadores e causa o aumento do número de páginas visitadas, especialmente na variação RANDOM_2. Por esse motivo, as avaliações de tempo de consulta foram as piores.

- SAMPLING: o comportamento das variações desta técnica foi bastante regular, devido a um tempo de construção bom, e, em comparação com as demais, uma razoável utilização do espaço de armazenamento. Com um processo de escolha de roteadores mais refinado que a técnica RANDOM, da qual foi derivada, a técnica SAMPLING obteve melhores resultados chegando a ser apenas 7% pior, com a variação SAMPLING_1, que a melhor técnica na consulta (mM_RAD_1). No resultado final, medido pelo desempenho médio, as duas variações obtiveram boa classificação, ficando atrás apenas da técnica mais rápida, a mM_RAD_1.

- M_LB_DIST: a simplicidade na forma de escolher os roteadores deveria tornar esta técnica uma das mais rápidas na construção de índices, entretanto, devido a um *split* bastante desbalanceado a técnica cria índices grandes, o que justifica os resultados obtidos no tempo de construção. Já no tempo de consulta, as duas variações obtiveram bons resultados, mesmo tendo um número de acesso a páginas alto. Isto aconteceu devido à baixa utilização do espaço de armazenamento, que implicou, devido a existência de poucos elementos nos nós, num menor número de computações para a resolução das consultas. Outra característica desta técnica foi a semelhança dos resultados obtidos entre as duas variações. Isto ocorreu devido a escolha do segundo roteador da M_LB_DIST_2 ser o mais próximo, em termos de distância, do atual roteador.

- m_RAD: como uma das técnicas mais complexas, esta técnica obteve os piores resultados, mesmo quando comparada com uma técnica de igual complexidade, como a mM_RAD. Em relação ao espaço de armazenamento, os piores resultados são certamente da m_RAD. O motivo desses resultados ruins é ocasionado pelo *split* extremamente

desbalanceado. Isso ocorre porque o algoritmo minimiza a soma dos raios de cobertura dos novos roteadores, e, na maioria das vezes, essa soma inclui um raio muito pequeno, chegando a cobrir apenas 2 elementos. Nas avaliações de consulta, da mesma forma que na técnica M_LB_DIST, a baixa utilização do espaço de armazenamento ocasiona um alto número de acesso a páginas, com um tempo baixo de consulta.

- mM_RAD: por ser uma técnica de alta complexidade, os resultados de tempo de construção não foram muito bons em relação às técnicas mais simples, apesar do desempenho da variação mM_RAD_1 ter sido próximo a elas. Entretanto, esse custo computacional resultou em boa utilização do espaço de armazenamento, sendo próximo ao melhor resultado, da RANDOM_2. Nas avaliações da consulta, o número de páginas acessadas foi seguramente o menor, ficando as duas variações com resultados muito parecidos. No tempo da consulta, a técnica ainda obteve melhor desempenho, porém com resultados próximos a outras técnicas.

Considerando todos os aspectos mostrados na Tabela 3.1 e a análise realizada sobre as técnicas de *split* originais, a técnica que melhor se destacou foi a mM_RAD_1. Esta técnica será utilizada nas comparações com as demais estruturas no restante desta dissertação.

3.2.2 Comparações da M-tree— *Splits* Novos

A Seção 2.3.2 apresentou técnicas novas para realização de *splits* (OVERLAP, mM_MEAN e MM_RAD2), e uma nova forma de reorganização para a M-tree, a Adaptive M-tree. Estas propostas são avaliadas nesta seção. Para a avaliação da proposta da Adaptive M-tree foram utilizadas as técnicas mM_MEAN e mM_RAD para a realização dos *splits*. Para a avaliação das técnicas, foi utilizado o conjunto de dados de imagens. Novamente, nos gráficos em que a dimensão está sendo analisada, o tamanho de página foi mantido em 8 Kb. Nos casos em que o tamanho de página está sendo avaliado, o número de dimensões foi mantido em 32.

Em todas as técnicas foram avaliadas variações confirmando o pai, que receberam o sufixo _1, e variações não confirmando o pai, que receberam o sufixo _2. Nas avaliações da Adaptive M-tree, foi incluído um sufixo A na técnica. Nos gráficos apresentadas a seguir, foram omitidas as técnicas OVERLAP_2, mM_MEAN_2 e mM_MEAN_2A para facilitar a visualização dos resultados. Para efeito de comparação, a melhor técnica de *split* original, a mM_RAD_1, foi incluída nos gráficos.

Tempo de Construção do Índice

A Figura 3.12 mostra o tempo necessário pelas técnicas para a construção dos índices com o aumento do número de dimensões. Entre as novas propostas, a técnica OVERLAP,

seguramente a mais complexa, teve o pior desempenho. A variação OVERLAP_2 (não mostrada na figura), chega a ser 8 vezes, em média, mais lenta que a mM_RAD_1, a mais rápida. Nas avaliações da Adaptive M-tree, houve um aumento bastante pequeno no tempo de construção, chegando no máximo a 3%, entre a técnica mM_RAD_1 e mM_RAD_1A. A técnica mM_MEAN_2 (não mostrada), devido a sua complexidade apresentou comportamento decrescente em relação ao tempo, se beneficiando com o aumento do número de dimensões uma vez que o número máximo de elementos de um nó diminui. Apesar da técnica mM_RAD_1 ser a mais rápida na média, novas variações como a mM_MEAN_1 e a MM_RAD2_1 têm desempenho bastante próximo.

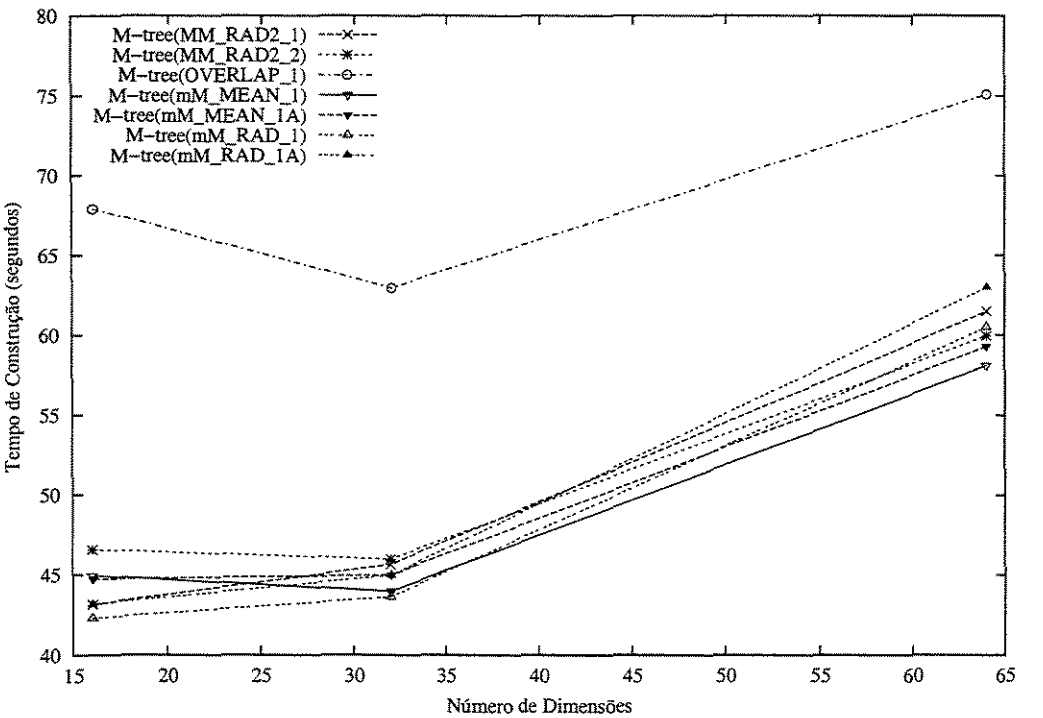


Figura 3.12: Tempo de construção do índice *vs.* número de dimensões

A influência causada pelo aumento do tamanho da página de disco (aumento do tamanho do nó) no tempo de construção dos índices é representado na Figura 3.13. De forma geral, o aumento do tamanho do nó traz também o aumento do número de computações realizadas durante o *split*, justificando o crescimento geral no tempo. Entretanto, algumas técnicas, especialmente as que não confirmam o pai, sofrem mais com esse aumento, sendo as técnicas mM_MEAN_2 e a OVERLAP_2 (não mostradas) os principais exemplos. A dificuldade de otimização, imposta pelo método do *split*, justifica os resultados ruins nessas técnicas.

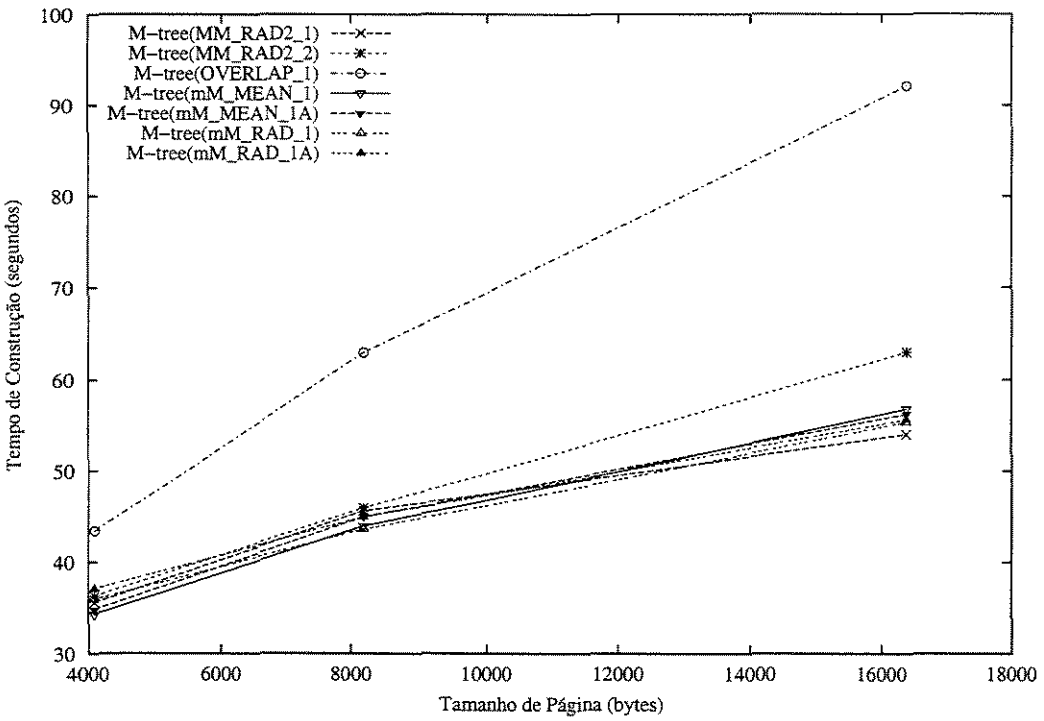


Figura 3.13: Tempo de construção do índice *vs.* tamanho da página de disco

Tamanho do Índice

O reflexo do aumento do número de dimensões no tamanho do índice é mostrado na Figura 3.14. As variações da técnica `OVERLAP` mostram uma utilização ruim do espaço de armazenamento, chegando gerar índices, com a `OVERLAP_2` (não mostrada), 140% maiores em média que a melhor técnica, a `MM_RAD2_2`. Isso mostra que os *splits* ocorrem de forma bastante desbalanceada. Outro fato observado é a diferença do tamanho do índice entre as variações da técnica `MM_RAD2`, sendo a variação `MM_RAD2_1` 45% maior que a `MM_RAD2_2`. Isso acontece porque como a maximização da distância entre os roteadores é um dos objetivos da técnica, e como a maioria das entradas está mais próxima do atual roteador, a técnica escolhe o segundo roteador como um dos mais distantes, que acaba tendo poucas entradas próximas a ele. Desta forma, o *split* é feito de forma desbalanceada, justificando o aumento do tamanho do índice. A técnica `mM_MEAN_1` apresentou desempenho bastante similar à `mM_RAD_2`, enquanto as variações que utilizaram a Adaptive M-tree também obtiveram melhora na utilização do espaço de armazenamento.

A Figura 3.15 mostra a influência gerada pela mudança do tamanho de página sobre o tamanho do índice. A partir desta figura, é possível verificar quais técnicas geram *splits* desbalanceados. Técnicas como a `OVERLAP` (duas variações) e a `MM_RAD2_1` são confirmadas como tendo *splits* desbalanceados pelo crescimento do índice demonstrado.

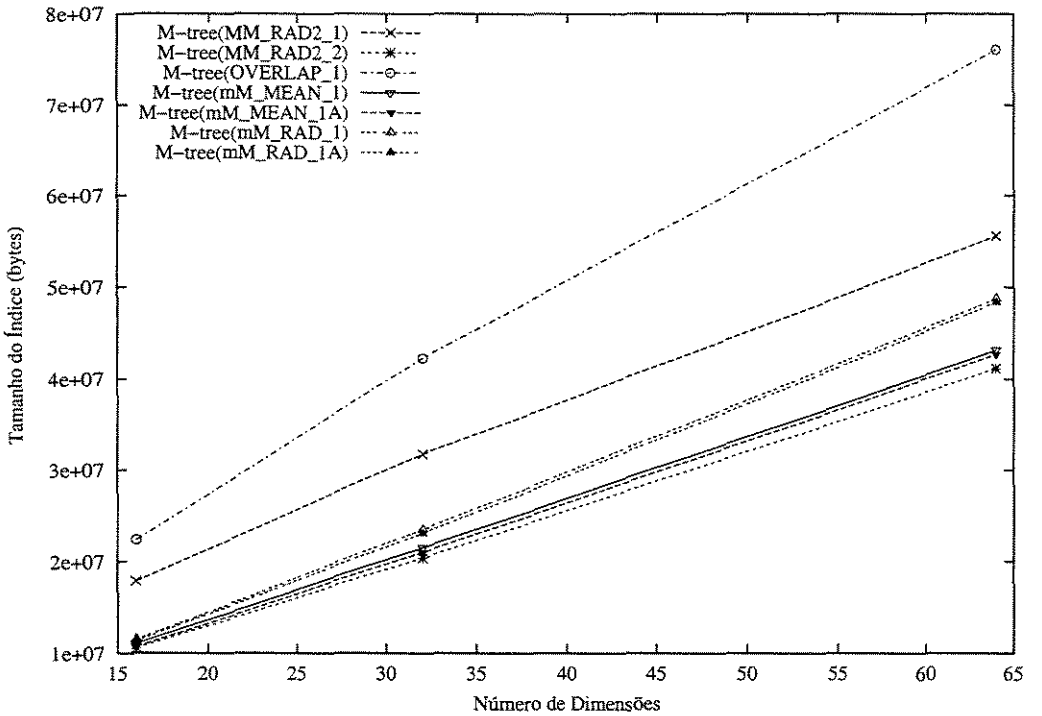


Figura 3.14: Tamanho do índice *vs.* número de dimensões

Entre as técnicas mais "econômicas", a técnica `MM_RAD2_2` mantém o melhor desempenho.

Tempo de Consulta

A Figura 3.16 mostra o tempo médio usado nas consultas com a variação da dimensão, tempos referentes às consultas de vizinho mais próximo retornando 21 imagens. Entre as melhores técnicas, a `mM_RAD_1` confirma pequena superioridade na média. Curiosamente, a técnica mais compacta, a `MM_RAD2_2`, foi uma das lentas técnicas. Isso acontece porque o método que procura criar roteadores afastados, para minimizar a sobreposição entre eles, acaba criando uma grande sobreposição entre os demais roteadores que compõem o nó superior ao *split*. Entre as novas propostas, a técnica `mM_MEAN_1` obteve o melhor resultado sendo cerca de 7% mais lenta na média. Entretanto, a `mM_MEAN_1` teve melhor desempenho de consulta com dados hiperdimensionais. Para verificar a eficiência da `mM_MEAN_1` sobre a `mM_RAD_1`, foram realizadas avaliações utilizando histogramas de cores com 128 cores, o que acabou confirmando a superioridade da `mM_MEAN_1`. Em 128 dimensões, a técnica `mM_RAD_1` é 10% mais lenta que a `mM_MEAN_1`.

O reflexo causado pelo aumento do tamanho da página sobre o tempo da consulta é mostrado na Figura 3.17. O aumento do tamanho da página é vantajoso para todas técnicas, tendo as técnicas com *split* mais desbalanceado as maiores quedas no tempo

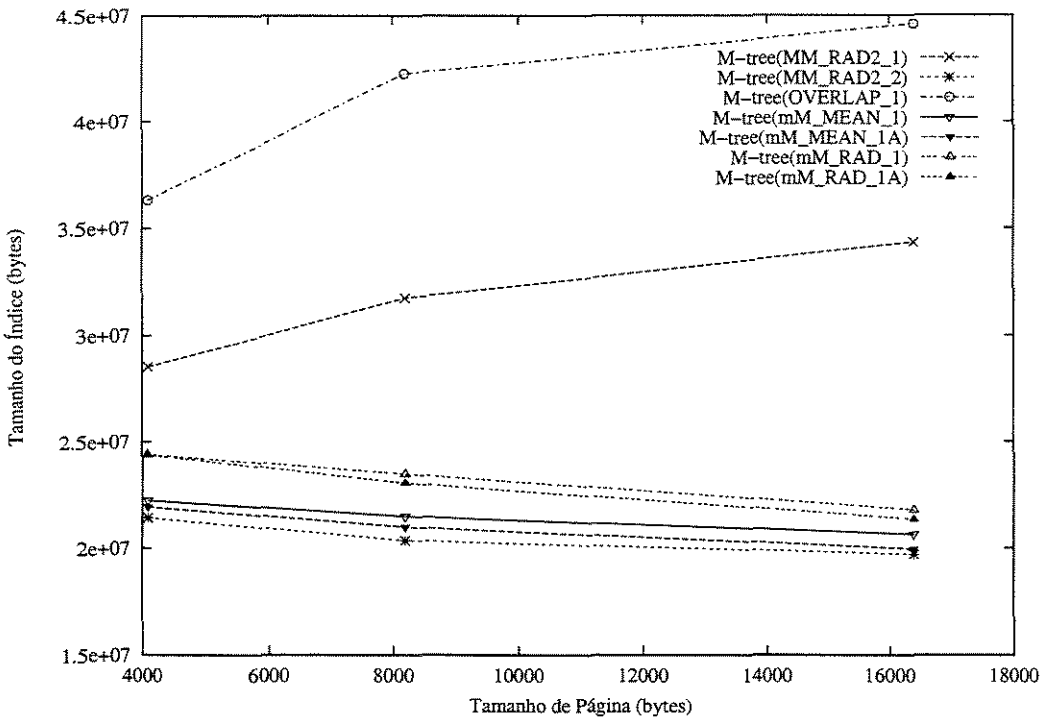


Figura 3.15: Tamanho do índice *vs.* tamanho da página de disco

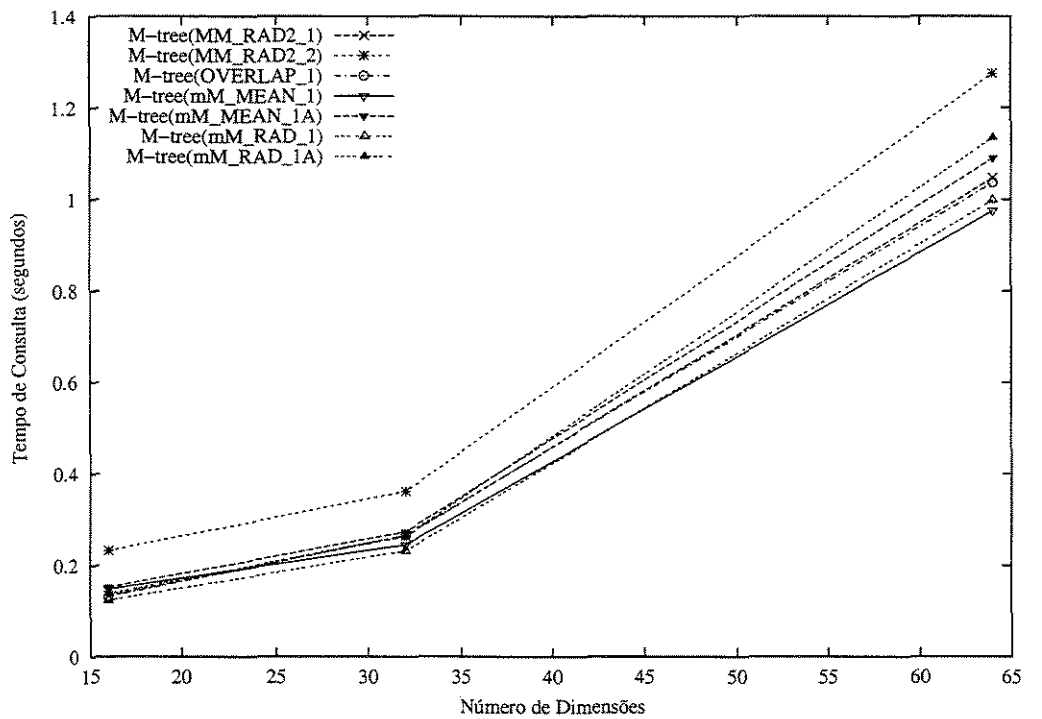


Figura 3.16: Tempo de consulta *vs.* número de dimensões

– OVERLAP (duas versões) e MM_RAD2_1. Isso acontece devido a baixa utilização do espaço de armazenamento, que deixa os nós com menos elementos, diminuindo o esforço computacional para escolher os caminhos.

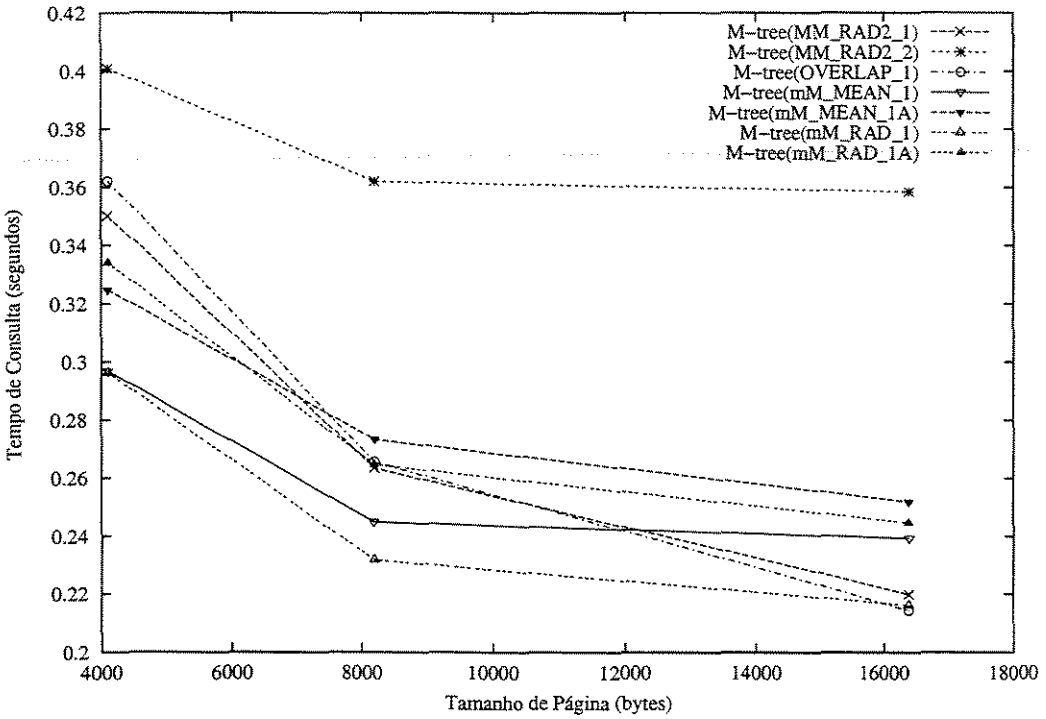


Figura 3.17: Tempo de consulta *vs.* tamanho da página de disco

A Figura 3.18 mostra o número de páginas acessadas na consulta de acordo com o número de dimensões. Novamente, o gráfico mostra a incapacidade dos resultados de acesso a páginas determinar as melhores técnicas. O principal exemplo disso ocorre entre as técnicas OVERLAP_2 (não mostrada) e MM_RAD2_2. A técnica OVERLAP_2 acessa 28% mais páginas que a MM_RAD2_2, entretanto a MM_RAD2_2, supostamente mais rápida, é na verdade 22% mais lenta que a OVERLAP_2. O principal motivo, novamente, é a baixa utilização do espaço de armazenamento feita pela técnica OVERLAP_2, que torna mais simples a tarefa de análise dos nós, diminuindo o esforço computacional e, consequentemente, o tempo real de consulta.

Conclusão — *Splits* Novos da M-tree

Novamente, não houve técnica que tenha se mostrado superior em todos os aspectos analisados. Para tentar determinar a técnica que melhor tenha se comportado, a Tabela 3.2 traz todas as novas propostas, incluindo a melhor técnica original. Da mesma forma que

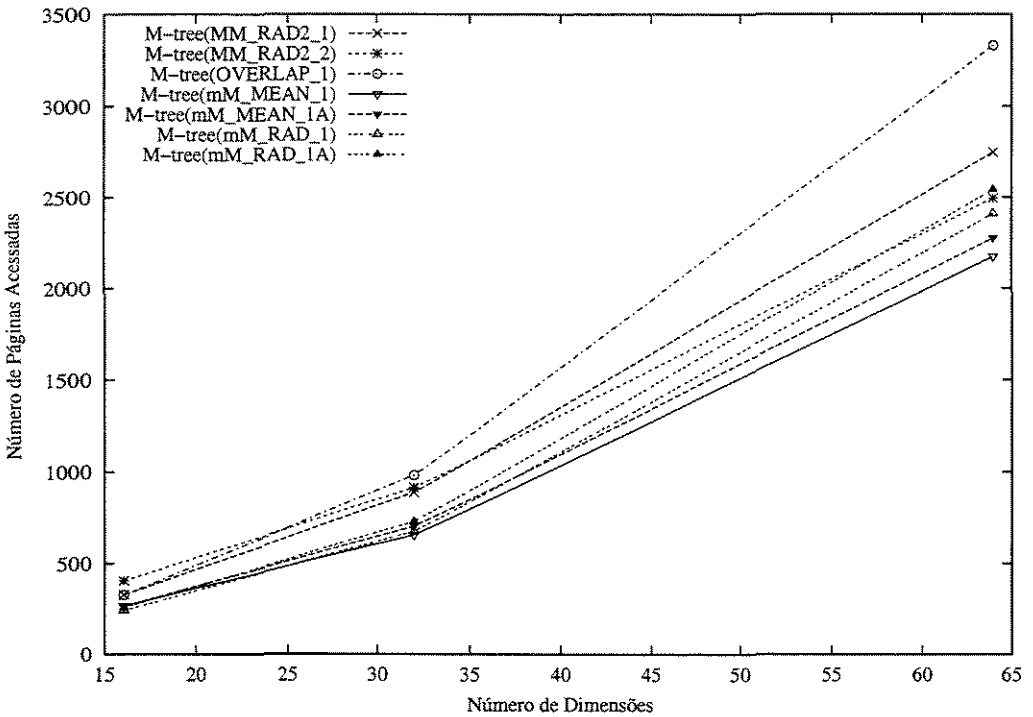


Figura 3.18: Páginas acessadas na consulta *vs.* número de dimensões

na Tabela 3.1, os valores na tabela representam o custo da técnica sobre a melhor técnica em determinado aspecto.

	tempo construção	tamanho índice	tempo consulta	acesso consulta	desempenho médio
mM_RAD_1	1	1.142503	1	1.014165	1.047501
OVERLAP_1	1.430145	1.916759	1.082916	1.416597	1.476607
mM_MEAN_1	1.010274	1.050702	1.077680	1	1.046219
MM_RAD2_1	1.027622	1.459879	1.090577	1.281532	1.192693
MM_RAD2_2	1.048689	1	1.574324	1.353772	1.207671
mM_MEAN_1A	1.023049	1.027793	1.168614	1.037930	1.073152
mM_RAD_1A	1.030909	1.127817	1.133089	1.080622	1.097272

Tabela 3.2: Comparação das técnicas de *split* novas da M-tree

De acordo com a comparação feita na tabela e os demais gráficos apresentados nessa seção, abaixo são realizados comentários sobre as novas propostas:

- OVERLAP: foi a técnica com pior desempenho no tempo de construção, especialmente com a OVERLAP_2. Isso ocorre porque durante a avaliação dos possíveis pares para novos roteadores, é necessário o cálculo do raio de cobertura, e não é possível descartar um par enquanto os raios não sejam calculados por completo. Em outras técnicas, como na mM_RAD, essa otimização pode ser realizada. Além do problema do custo de construção, os *splits* gerados são muito desbalanceados, levando a técnica

ao pior resultado de utilização do espaço de armazenamento. No tempo de consulta, apesar do alto número de páginas acessadas, a variação OVERLAP_1 obteve bom resultado no tempo de consulta.

- mM_MEAN: nesta técnica, como na OVERLAP, não é possível realizar otimizações. O resultado disso foi a variação que não confirma o pai (mM_MEAN_2) ser 2 vezes mais lenta, para a construção da árvore, que a variação mM_MEAN_1, que ficou entre as mais rápidas. O tamanho do índice foi reduzido em relação as variações da técnica mM_RAD, porém ainda foi maior que a técnica MM_RAD2_2. Na consulta, novamente tem-se bons resultados. Por uma pequena diferença, as variações da mM_MEAN acessaram menos páginas para responder às consultas. No tempo de execução das consultas, os resultados foram os melhores entre as novas propostas, mas não superaram os resultados da mM_RAD_1. No desempenho médio, a variação mM_MEAN_1 obteve o melhor resultado entre as novas propostas, e igual desempenho à técnica mM_RAD_1. Se considerado apenas os resultados dos dados hiperdimensionais, 64 e 128 (avaliação especial), a variação mM_MEAN_1 foi superior à técnica mM_RAD_1 em até 10% no tempo de consulta (128 dimensões).

- MM_RAD2: diferentemente das duas técnicas anteriores, foi possível realizar otimizações nesta técnica. O resultado disso foi que nas duas variações o tempo de construção do índice ficou entre os melhores. O resultado do tamanho do índice foi bastante diferente entre as duas variações. Enquanto a MM_RAD2_2 gerou os menores índices medidos para a M-tree, a variação que confirma o pai gerou índices 45% maiores devido a uma característica do *split* desta variação, discutida na análise da Figura 3.14. Entre os resultados das consultas, ambas variações tiveram alto número de acessos a páginas. Entretanto, com a variação MM_RAD2_1 foram alcançados bons resultados com o tempo de consulta.

- Adaptive M-tree: esta proposta de reorganização foi testada com duas técnicas de *split*, mM_MEAN e mM_RAD. O tempo de construção do índice, conforme esperado, aumentou nas duas técnicas, apesar do aumento ter sido pequeno. No tamanho do índice, os resultados foram melhores com as duas técnicas. Entretanto, os resultados nas consultas não foram bons. Tanto no número de páginas acessadas, quanto no tempo da consulta, houve a piora dos resultados.

Conforme as avaliações realizadas sobre as novas propostas para a M-tree, a técnica mM_MEAN_1 obteve o melhor desempenho médio, sendo este praticamente igual ao desempenho da melhor técnica de *split* original, a mM_RAD_1. A técnica mM_MEAN_1 será utilizada na comparação com as demais estruturas nas Seções 3.2.3 e 3.2.4.

3.2.3 Construção dos Índices

Após a análise e escolha de apenas duas versões de M-tree, a comparação entre as diferentes estruturas, apresentadas no Capítulo 2, M-tree, R-tree, R*-tree, SS-tree e SR-tree é realizada nesta seção quanto aos aspectos de construção dos índices, tempo e tamanho.

Conforme mostrado na Seção 3.1.3, as estruturas foram avaliadas em relação aos conjuntos de dados de imagens e uniforme. Nesta seção, esses resultados foram separados em duas partes, sendo a primeira a análise dos resultados sobre o conjunto de dados de imagens.

Durante a análise dos gráficos, enquanto a dimensão estiver sendo variada, o tamanho da página de disco é mantido em 8 Kb. Quanto o tamanho da página de disco é variada, a dimensão é mantida em 32.

Conjunto de Dados de Imagens

Através das Figuras 3.19 e 3.20 é possível avaliar o tempo gasto na construção dos índices. A primeira figura mostra claramente a dificuldade da R*-tree para gerar o índice. Em 64 dimensões, enquanto as variações da M-tree levam 1 minuto para a construção do índice, a R*-tree leva 22 minutos. A complexidade do *split* da R*-tree, diretamente ligado ao número de dimensões, justifica o comportamento apresentado. A R-tree, mesmo utilizando um *split* quadrático em relação ao número de entradas, foi bastante superior à R*-tree no tempo de construção, sendo cerca de 70% mais rápida na média.

Entre as estruturas com melhor tempo de construção, a M-tree teve o melhor desempenho, sendo na média a SS-tree 47% mais lenta, e a SR-tree 80% mais lenta. Em relação à SS-tree, a SR-tree, apesar de ser mais lenta, apresentou bom resultado, se considerado a complexidade do *split*, cerca de 20% mais lenta na média.

Quando comparado o impacto do aumento do tamanho da página de disco sobre o tempo de construção do índice, todas as estruturas levaram mais tempo quando o tamanho de página foi aumentado. Entretanto, enquanto a SR-tree demorou apenas 6% a mais, a SS-tree demorou 36% a mais, e a M-tree 65%, quando comparado o aumento da página de disco de 4 Kb para 16 Kb. Isso acontece porque a SR-tree gasta muito espaço para a representação de uma entrada em um nó, significando esse aumento a possibilidade de uma melhor distribuição das entradas. Entre as estruturas R-tree e R*-tree, o aumento do tamanho da página implicou no aumento acentuado do tempo de construção dos índices.

Durante a construção dos índices, houve problemas na construção de índices quando foi utilizado 64 dimensões e 4 Kb de tamanho de página de disco. O problema do *fanout*, discutido na Seção 2.1.3, chegou ao extremo neste caso, fazendo com que as estruturas R-tree, R*-tree e SR-tree não conseguissem gerar índices pela falta de espaço nos nós.

A Figura 3.21 mostra o crescimento do tamanho dos índices à medida que o número

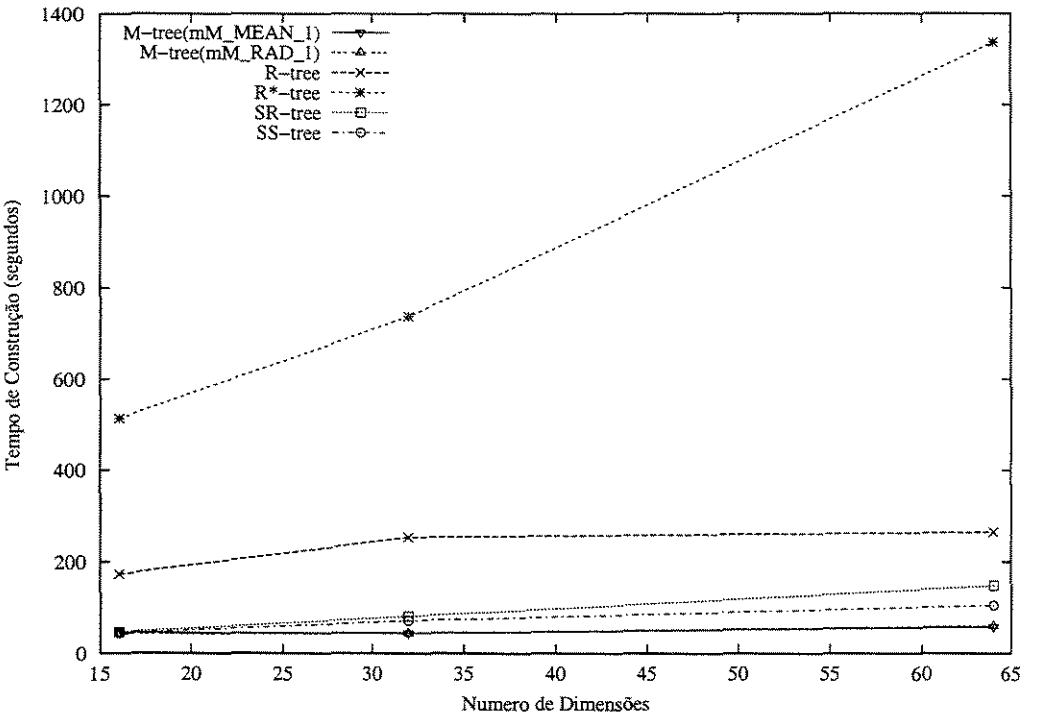


Figura 3.19: Tempo de construção do índice *vs.* número de dimensões

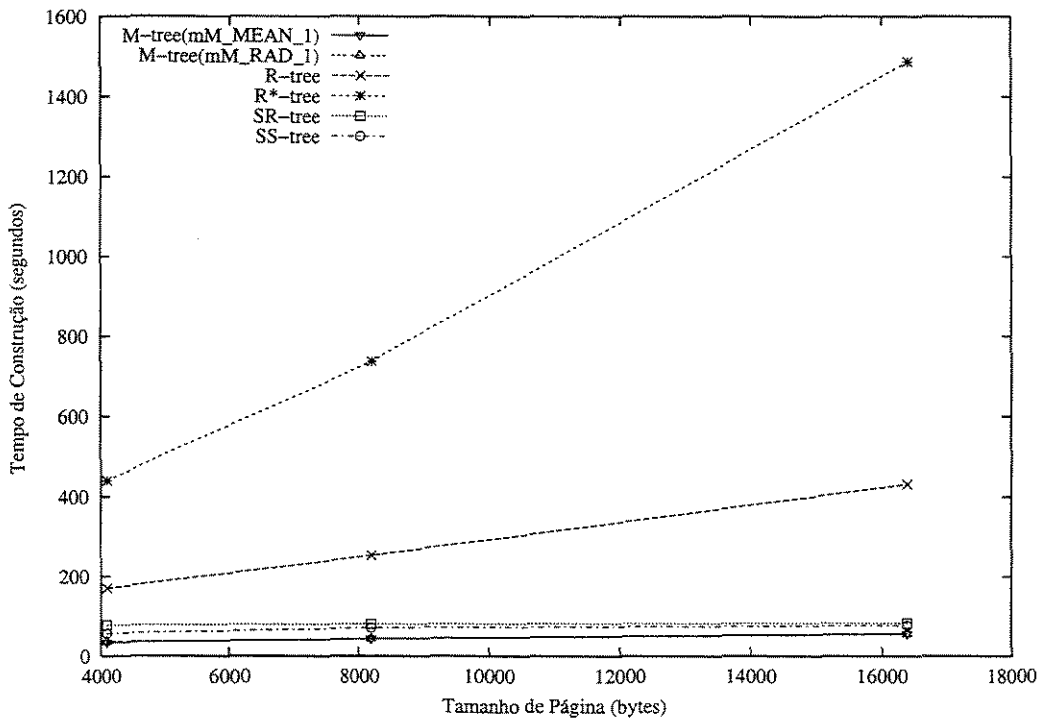


Figura 3.20: Tempo de construção do índice *vs.* tamanho da página de disco

de dimensões cresce. Entre as estruturas, a SS-tree seguramente é a mais "econômica". Isso acontece principalmente porque o espaço necessário para a representação de uma entrada em um nó é pequeno. Na M-tree, onde o tamanho da entrada também é pequeno, o mesmo comportamento não foi observado, sendo a M-tree a estrutura que gerou os maiores índices por não ter *splits* tão balanceados quanto os da SS-tree. Na SR-tree, é observado um crescimento mais acentuado, em relação às demais, a medida que a dimensão aumenta. Isso acontece devido ao tamanho da entrada, que é o maior de todos. Quando poucas entradas podem ser armazenadas em um nó, a altura da árvore cresce, aumentando o tamanho do índice.

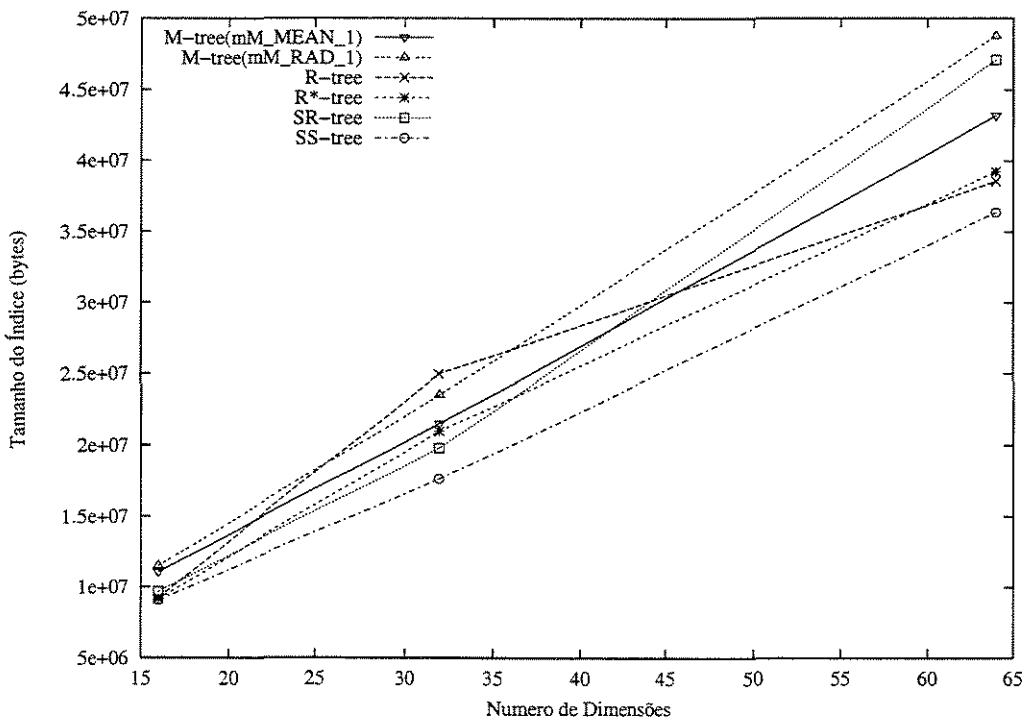


Figura 3.21: Tamanho do índice *vs.* número de dimensões

O reflexo causado pelo aumento do tamanho da página de disco sobre o tamanho do índice é mostrado na Figura 3.22. Apesar de todas as técnicas terem conseguido diminuir o tamanho do índice com o aumento da página, a SR-tree se destacou por conseguir reduzir em até 28%. Isso acontece porque a árvore ficou com menor altura, devido ao aumento do tamanho do nó, ocorrendo isso com maior ênfase na SR-tree por ela ter entradas muito grandes. A SS-tree confirma a sua superioridade como a mais compacta estrutura, apesar da SR-tree se aproximar quando o tamanho de página é igual a 16 Kb. Entre as demais estruturas, a M-tree, além de ter o pior desempenho, não conseguiu tirar proveito satisfatório do aumento do tamanho da página.

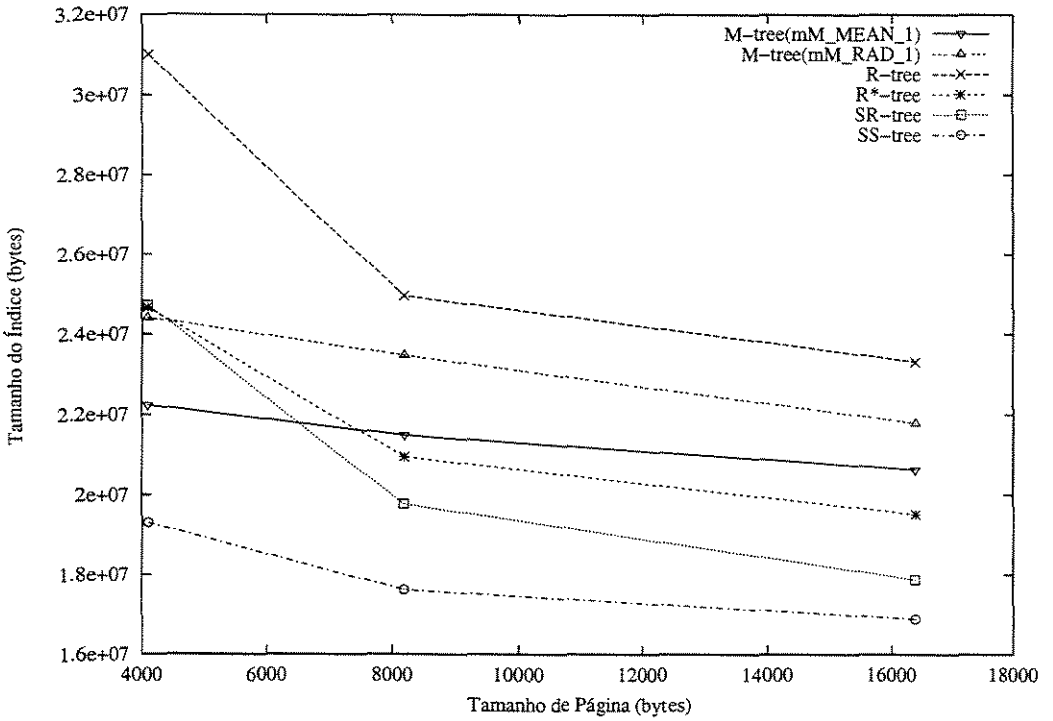


Figura 3.22: Tamanho do índice *vs.* tamanho da página de disco

Conjunto de Dados Uniforme

Além do conjunto de dados de imagens, foi utilizado o conjunto de dados uniforme, conforme descrito na Seção 3.1.1. Conjuntos de dados de aplicações reais dificilmente se assemelham ao conjunto de dados uniforme, entretanto, esse conjunto pode ser utilizado para a avaliação da robustez das estruturas estudadas.

Na Figura 3.23 mostra o tempo de construção levado para a construção dos índices em relação ao número de dimensões. Na figura é possível verificar o comportamento anômalo apresentado pelas estruturas SS-tree e SR-tree. Devido ao conjunto de dados, quando há a necessidade da realização de um *split*, especialmente nos nós folhas, o processo para a criação da esfera que contém os pontos se torna bastante complexo. Neste processo, onde os pontos são incluídos de forma incremental, quando é necessário o aumento do raio para cobrir o novo ponto incluído é realizado o cálculo de um novo centro e raio para a esfera. Este cálculo é realizado através da resolução de um sistema de equações. A complexidade para a resolução do sistema de equações depende principalmente do número de pontos envolvidos, além do número de dimensões. Notadamente, a utilização do conjunto de dados uniforme ocasionou no aumento excessivo da necessidade desse cálculo. Em 32 dimensões é possível armazenar mais pontos, para um tamanho fixo de página, do que em 64 dimensões, justificando o comportamento apresentado. Quando utilizado página

de disco de 16 Kb e 64 dimensões, o tempo levado para a construção das árvores chega a 5 horas e 40 minutos.

Na M-tree houve um aumento menor do tempo de construção do índice com relação aos resultados de dos conjuntos de imagens, cerca de 35%. Nas estruturas R-tree e R*-tree, o comportamento foi contrário às demais estruturas. Na R-tree houve uma redução de cerca de 7%, na média, com a utilização do conjunto de dados uniforme. Na R*-tree essa redução com relação ao conjunto de dados de imagens chegou a 12%.

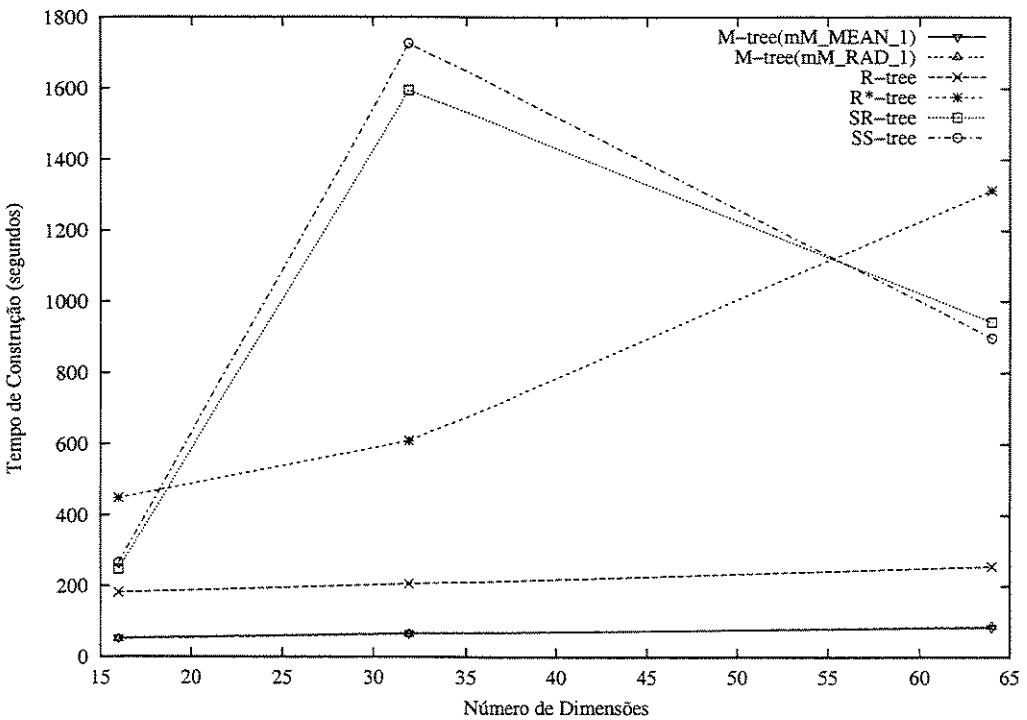


Figura 3.23: Tempo de construção do índice *vs.* número de dimensões (uniforme)

Em relação ao tamanho dos índices gerados para o conjunto de dados uniforme, houve algumas mudanças (Figura 3.24) em relação aos resultados obtidos com o conjunto de dados de imagens. As estruturas SS-tree e SR-tree mantiveram o mesmo tamanho. Porém, as estruturas R-tree e R*-tree reduziram o tamanho do índice gerado, sendo a R-tree a estrutura com os menores índices. Na M-tree, houve um aumento de cerca de 80% em média no tamanho dos índices (`mM_RAD_1`), sendo a M-tree com a técnica `mM_MEAN_1` melhor opção na M-tree.

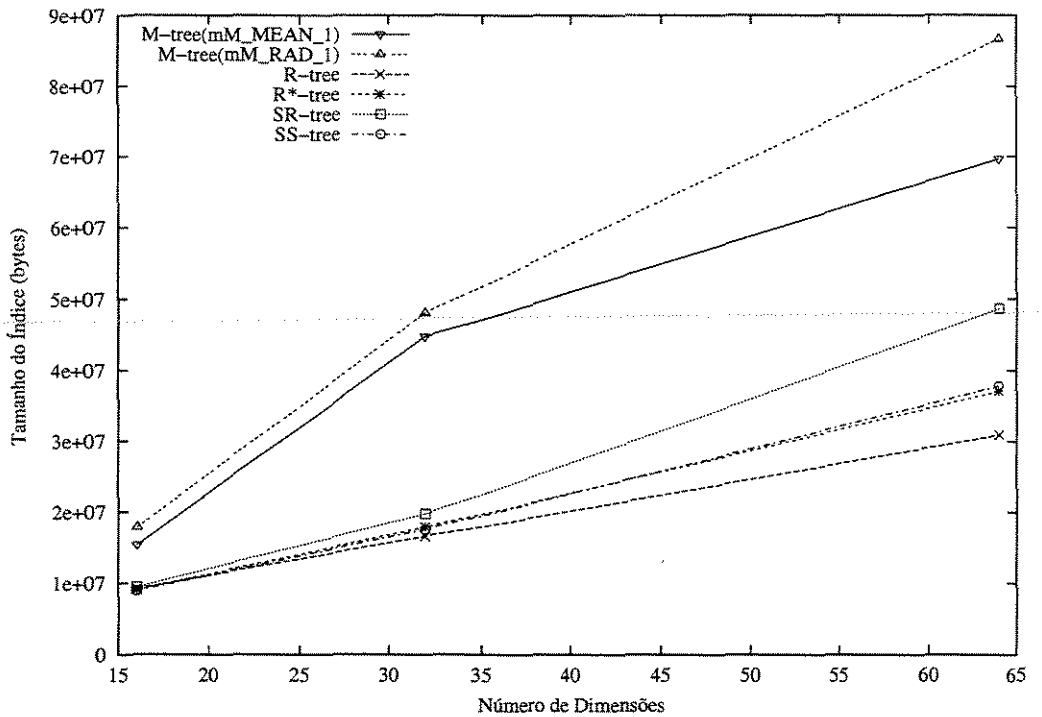


Figura 3.24: Tamanho do índice *vs.* número de dimensões (uniforme)

3.2.4 Consultas de Similaridade

As estruturas avaliadas quanto aos aspectos de construção na seção anterior, são, nesta seção, avaliadas quanto aos aspectos de consulta. Foram realizadas consultas de vizinho mais próximo, conforme descrição na Seção 3.1.3, para a avaliação das estruturas.

Da mesma forma que na seção anterior, os resultados obtidos utilizando-se os conjuntos de dados de imagens e uniforme são separados em duas partes.

Conjunto de Dados de Imagens

A Figura 3.25 mostra o tempo levado pelas estruturas para a resolução das consultas. Claramente, a estrutura com o melhor desempenho é a SR-tree. A M-tree teve desempenho bastante próximo à SS-tree, entretanto, em 64 dimensões a SS-tree mostra melhor desempenho, tornando-a a melhor das duas na média. As estruturas R-tree e R*-tree tiveram os piores resultados. Nelas, o rápido crescimento no tempo de consulta é explicado pelo alto grau de sobreposição que ocorre.

O reflexo causado pelo aumento do tamanho da página de disco é mostrado na Figura 3.26. Esse aumento beneficiou estruturas como a M-tree, que acaba superando a SS-tree. Na SR-tree é observado certa queda, porém com melhor desempenho em 8 Kb.

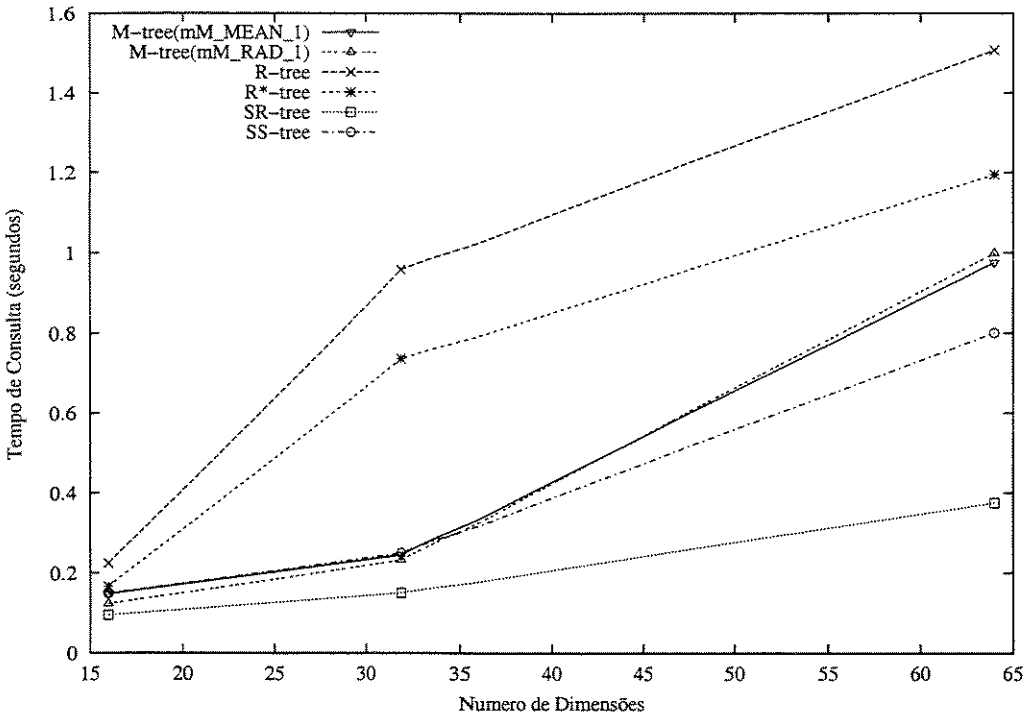


Figura 3.25: Tempo de consulta *vs.* número de dimensões

É importante notar que utilizando-se 64 dimensões, o aumento do tamanho de página traz o melhor desempenho em 16 Kb, ou seja, o aumento do tamanho da página é recomendado quando o número de dimensões cresce. Na R-tree e R*-tree, a queda no tempo de consulta foi maior, chegando a 25% com a R-tree. Isso acontece devido ao melhor agrupamento alcançado pela técnica de *split*.

Na Figura 3.27 é mostrado a variação do tempo da consulta em relação ao número de vizinhos retornados. Da mesma forma que no estudo da M-tree, as estruturas tiveram o mesmo comportamento qualitativo.

O número de páginas acessadas nas consultas de similaridades, em relação ao número de dimensões, é mostrado na Figura 3.28. Em 64 dimensões, a estrutura R*-tree apresenta desempenho similar a M-tree, entretanto a M-tree teve desempenho melhor no tempo de consulta.

Conjunto de Dados Uniforme

Na Figura 3.29 é mostrado o tempo de consulta em relação ao número de dimensões. Neste caso, a mudança do conjunto de dados ocasionou uma mudança radical dos resultados. Todas as estruturas sofreram os resultados do aumento excessivo da sobreposição, o que ocasionou um aumento geral no tempo da consulta. Entretanto, a R-tree teve o menor

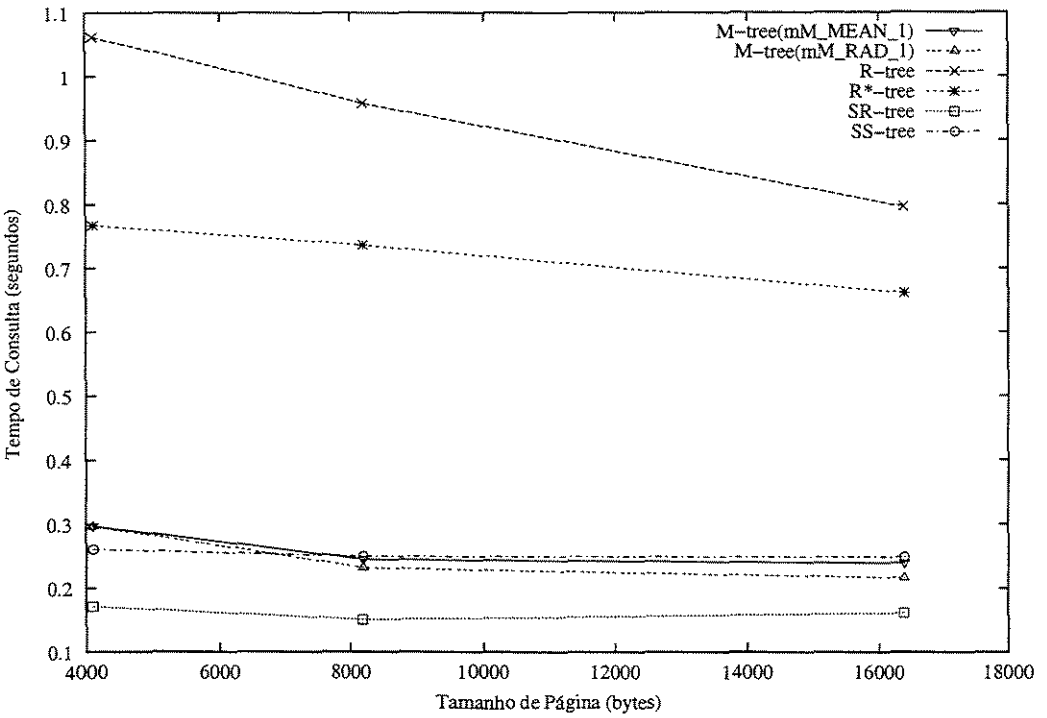


Figura 3.26: Tempo de consulta *vs.* tamanho da página de disco

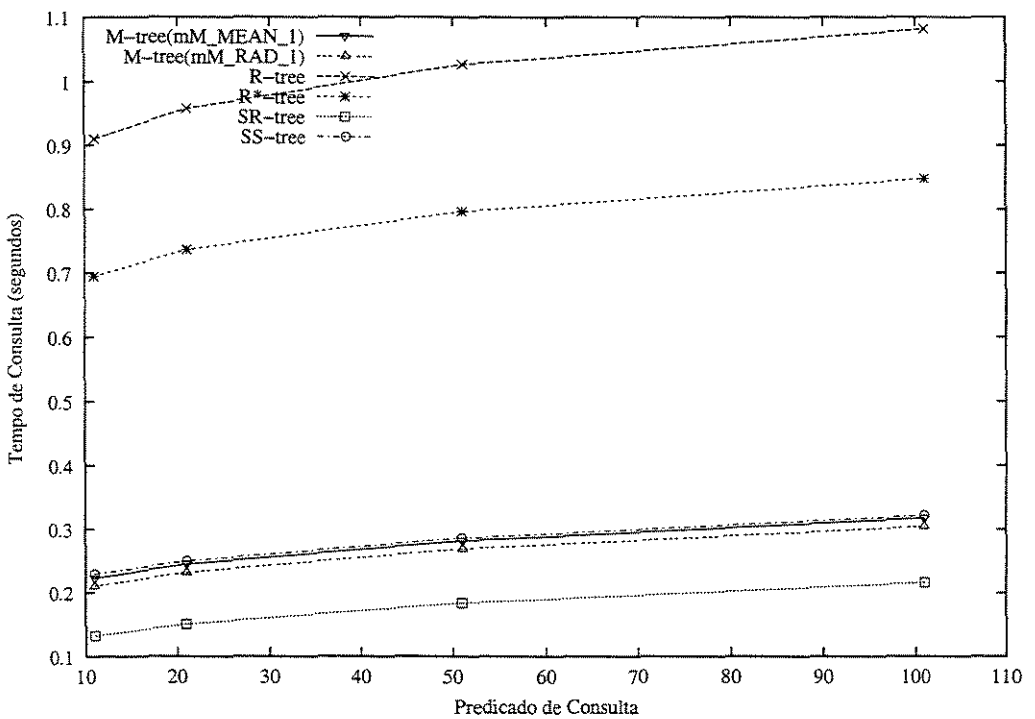


Figura 3.27: Tempo de consulta *vs.* número de vizinhos retornados

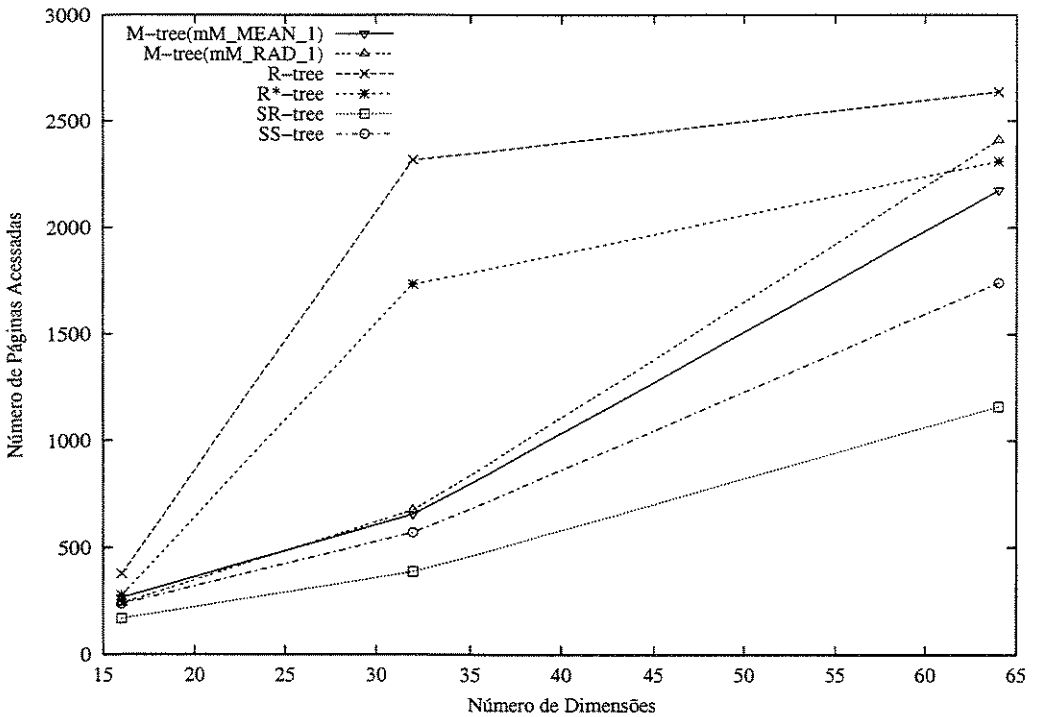


Figura 3.28: Páginas acessadas na consulta *vs.* número de dimensões

aumento, cerca de 110% na média, e tornou-se a estrutura mais rápida. A SR-tree teve o seu tempo aumentado cerca de 8 vezes na média. A M-tree, os efeitos do conjunto de dados também não foram bons, teve seu tempo aumentado cerca de 7 vezes.

3.2.5 Busca Linear de Vetores de Características

Em [BGRS99], Beyer, Goldstein, Ramakrishnan e Shaft estudam o efeito do número de dimensões sobre o problema do vizinho mais próximo. No trabalho é mostrado que sobre um amplo conjunto de condições, à medida que o número de dimensões cresce, a distância ao ponto mais próximo se aproxima da distância do ponto mais distante. Um efeito primário deste resultado é que as estruturas de acesso podem não ser tão eficientes quanto uma simples busca linear nos pontos.

Devido a este resultado, nesta seção é analisado o desempenho das estruturas hierárquicas, apresentadas no Capítulo 2, em relação a busca linear. Nela, os vetores de características são armazenados de forma linear, sem a criação de qualquer mecanismo para acelerar a busca de elementos. Para a resolução de uma consulta, uma varredura é realizada sobre todos os elementos no índice.

O custo para a criação do índice linear é extremamente baixo, uma vez que é necessário

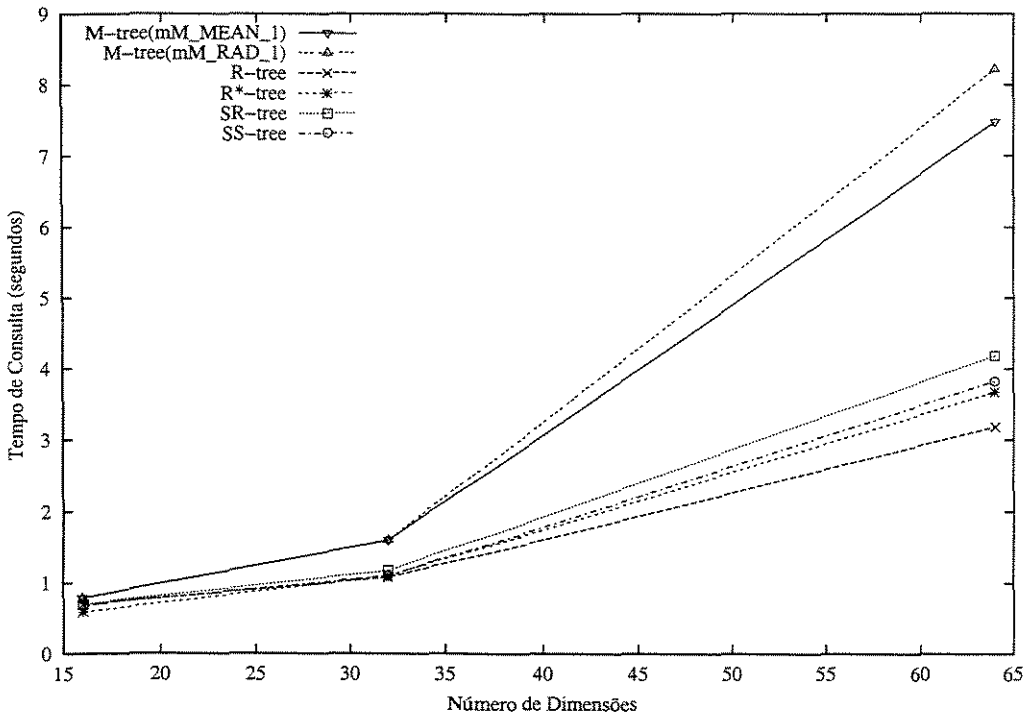


Figura 3.29: Tempo de consulta *vs.* número de dimensões (uniforme)

apenas o armazenamento dos elementos (vetores de características), sem a necessidade de qualquer computação sobre eles. Desta forma, a busca linear superou todas as estruturas hierárquicas, levando apenas 7% do tempo da estrutura hierárquica mais rápida, a M-tree. Variações do tamanho de página praticamente não influenciam o tempo levado na construção do índice.

Como na busca linear são apenas armazenados os elementos, diferente das estruturas hierárquicas onde árvores são utilizadas na organização dos elementos, o tamanho dos índices gerados é o menor possível. Isso significa que o índice gerado pela SS-tree, menor índice gerado entre as estruturas hierárquicas, é cerca de 66% maior que o índice gerado pela busca linear.

A avaliação de consulta realizada sobre a busca linear foi a mesma realizada sobre as demais estruturas. Para a resolução das consultas de vizinho mais próximo, uma varredura sobre o índice linear é necessária. Na Figura 3.30 é mostrado o número de páginas acessadas para a resolução das consultas. Com exceção das estruturas R-tree e R*-tree em 32 dimensões, todas as estruturas acessaram menos páginas para a resolução das consultas.

Na busca linear, praticamente não há complexidade na utilização do índice. Na varredura do índice, realizada durante uma consulta, a cada página lida, é necessário apenas

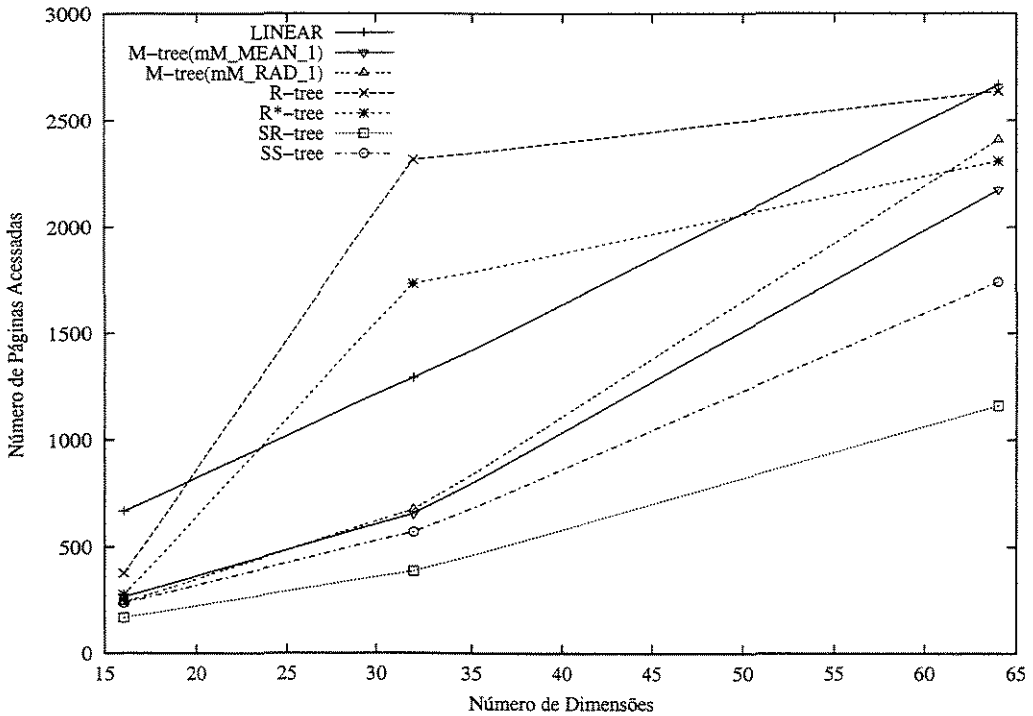


Figura 3.30: Páginas acessadas na consulta *vs.* número de dimensões

o cálculo da distância. Isso implica que no tempo total gasto na consulta, 85% do tempo é gasto com o cálculo de distância entre os elementos. Nas demais estruturas, onde é realizado, através do GiST, um maior controle sobre o índice, o tempo gasto com a computação de distâncias é de apenas 50%. O resultado dessa diferença é mostrado na Figura 3.31, onde o tempo de consulta da busca linear torna-se competitivo com as estruturas mais eficientes. A SR-tree continua sendo a mais rápida, entretanto a busca linear, chamada aqui de LINEAR, foi apenas 2% mais lenta que ela em 64 dimensões. Em 32 ou 16 dimensões, a LINEAR teve desempenho similar ou pior que as estruturas SS-tree e M-tree.

Devido aos resultados surpreendentes apresentados na busca linear, novas avaliações foram realizadas para verificar como o crescimento do conjunto de dados influencia no desempenho das estruturas mostradas. Para isso, a partir do conjunto de imagens foram escolhidos, aleatoriamente, subconjuntos de 5.000, 10.000 e 20.000 imagens do conjunto de imagens utilizado nas demais avaliações. Nos gráficos mostrados a seguir, o tamanho da página de disco foi mantido em 8 Kb. Nas avaliações de consulta, os gráficos correspondem ao tempo levado em consultas de vizinho mais próximo retornando 21 elementos.

A Figura 3.32 mostra o crescimento do número de páginas acessadas em relação ao número de elementos indexados. Na figura foram utilizados elementos de 64 dimensões,

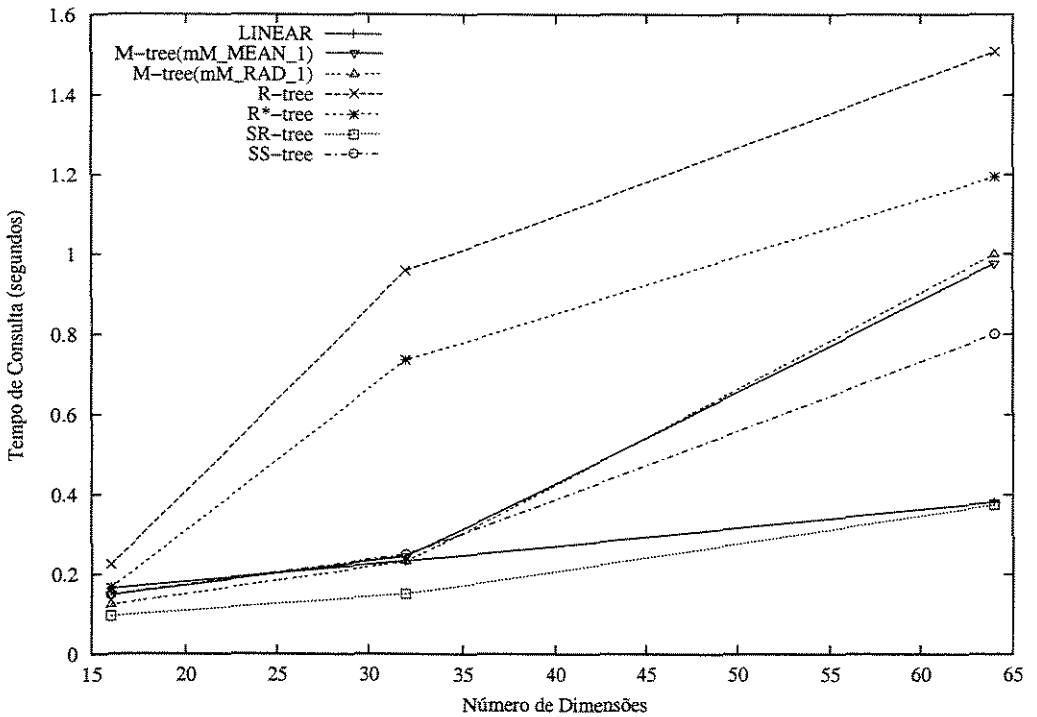


Figura 3.31: Tempo de consulta *vs.* número de dimensões

onde a busca linear apresentou melhor resultado. A curva da LINEAR tem um crescimento bem mais acentuado que as demais, se tornando em 40.000 a estrutura que mais acessa páginas. Esse comportamento de crescimento acelerado em relação às estruturas SR-tree, SS-tree e M-tree, também é verificado em 32 e 16 dimensões, sendo que nestas o número de acessos chega a ser 90% e 150% maior que a M-tree, respectivamente.

O tempo levado nas consultas, com relação ao número de elementos indexados é mostrado na Figura 3.33. No gráfico, gerado com elementos de 64 dimensões, a LINEAR apresenta um crescimento do tempo mais acelerado que a SR-tree, e quando é utilizado índices com 40.000 elementos já se torna melhor. Foi verificado que, como o problema do *fanout* pode prejudicar a SR-tree, se a página de disco é aumentada para 16 Kb, a SR-tree já é melhor que a LINEAR em índices de 20.000 elementos. Mantendo página de 8 Kb, e utilizando-se elementos de 32 e 16 dimensões, a SR-tree tem melhor desempenho que a LINEAR em índices de 12.000 e 11.000 elementos, respectivamente.

Conforme os resultados obtidos, a busca linear tem a sua eficiência relacionada com o número de elementos indexados e o número de dimensões utilizadas. A busca linear praticamente não enfrenta problemas com o aumento do número de dimensões, exceto pelo aumento da complexidade da computação de distâncias, que é comum a todas as estruturas. Isto é verificado com os bons resultados apresentados pela busca linear em

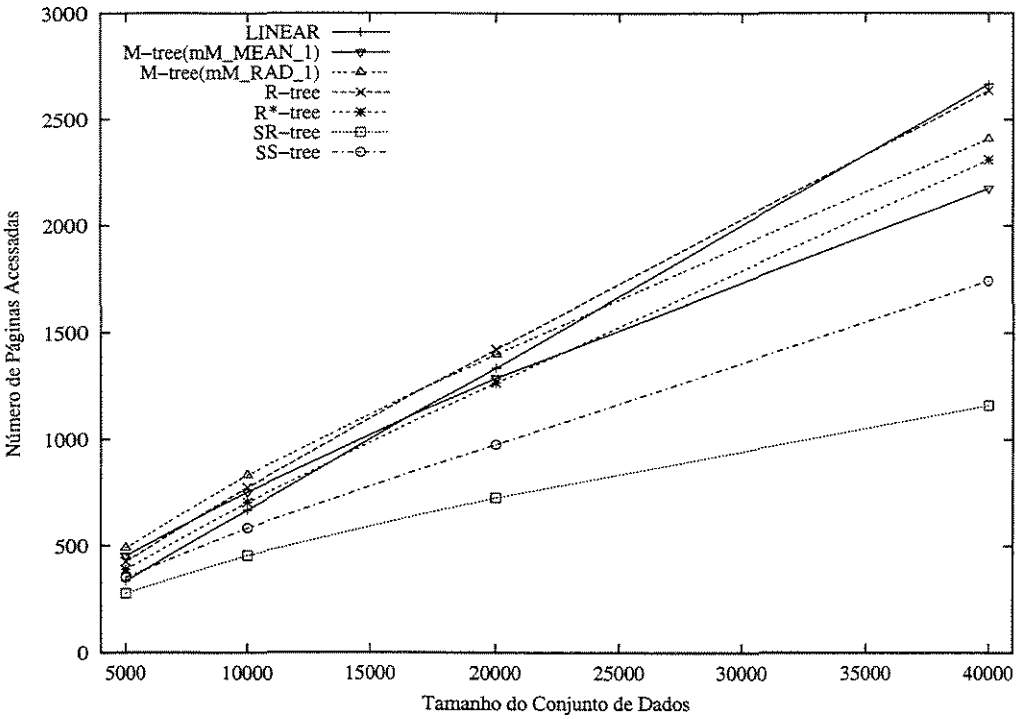


Figura 3.32: Páginas acessadas na consulta *vs.* tamanho do conjunto de dados (64d)

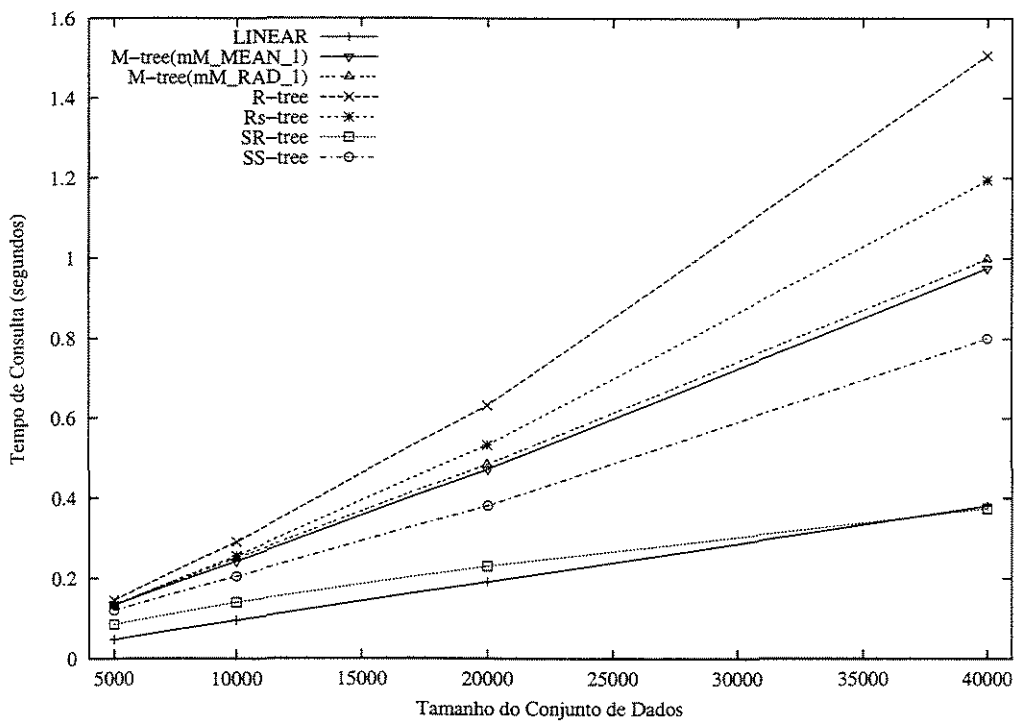


Figura 3.33: Tempo de consulta *vs.* tamanho do conjunto de dados (64d)

64 dimensões. Entretanto, o principal fator que determina a escolha de uma estrutura hierárquica de indexação em vez da busca linear é o número de elementos indexados. Em ambientes hiperdimensionais, p.ex. 64 dimensões, conjuntos de dados a partir de 40.000 elementos tornam estruturas como a SR-tree melhores em termos de tempo de consulta. Quando comparado o número de páginas acessadas, algumas estruturas hierárquicas, como a SS-tree e SR-tree, já são melhores com conjuntos de dados de 10.000 elementos.

3.3 Conclusão

Neste capítulo foi realizada a análise de desempenho das as estruturas apresentadas no Capítulo 2. Na Seção 3.1 foram apresentados os critérios utilizados na avaliação das estruturas, sendo eles: conjunto de dados, ambiente de análise e parâmetros de análise. Na análise das estruturas, apresentada na Seção 3.2, houve maior ênfase sobre a estrutura M-tree (Seções 3.2.1 e 3.2.2), sendo seguida pela comparação geral entre as estruturas. Finalmente, uma forma alternativa de indexação, a busca linear, foi avaliada na Seção 3.2.5.

A avaliação da estrutura M-tree incluiu novas políticas de *split* e nova reorganização da M-tree, apresentada na Seção 2.3.2. Dessas novas propostas, a técnica de *split* `mM_MEAN_1` teve desempenho igual ou superior a melhor técnica de *split* original, a `mM_RAD_1`. Notadamente, no ambiente hiperdimensional, comum nas aplicações de banco de imagens, os melhores resultados foram alcançados, entre todas as variações de M-tree, pela técnica proposta `mM_MEAN_1`.

Na comparação geral realizada entre as estruturas R-tree, R*-tree, SS-tree, SR-tree e M-tree (duas variações), apresentada nas Seções 3.2.3 e 3.2.4, foram utilizados os conjuntos de dados de imagens e uniforme. O comportamento apresentado nas avaliações com o conjunto de dados uniforme foi bastante diferente do apresentado com o conjunto de dados de imagens. Entretanto, os resultados oriundos de uma aplicação real, no caso banco de imagens, são mais representativo na definição das melhores estruturas. Desta forma, a Tabela 3.3 apresenta uma comparação geral entre as estruturas utilizando-se os resultados obtidos com o conjunto de dados de imagens. Da mesma forma que na Tabela 3.1, os cálculos realizados para cada aspecto apresentado foram feitos em relação ao número de dimensões. A média entre os resultados dos aspectos de tempo de construção, tamanho do índice e tempo de consulta é mostrada na coluna de desempenho médio. A busca linear não se encontra na tabela devido aos resultados serem totalmente dependentes do número de elementos indexados, o que poderia gerar falsa expectativa de desempenho.

A partir dos resultados da tabela e dos demais resultados apresentados ao longo deste capítulo, é realizada abaixo a avaliação final das estruturas de indexação:

- M-tree: o tempo levado na construção dos índices foi, sem dúvida, o melhor tempo

	tempo construção	tamanho índice	tempo consulta	acesso consulta	desempenho médio
M-tree (mM_RAD_1)	1	1.313623	1.834931	1.749552	1.382851
M-tree (mM_MEAN_1)	1.010274	1.208517	1.929809	1.715414	1.382867
R-tree	4.753460	1.165408	4.249186	3.497999	3.389351
R*-tree	17.050752	1.092545	3.277806	2.704458	7.140368
SS-tree	1.473636	1	1.789048	1.461011	1.420895
SR-tree	1.803180	1.161011	1	1	1.321397

Tabela 3.3: Comparação das estruturas de indexação

entre as estruturas hierárquicas. O segundo melhor tempo, da SS-tree, foi 47% mais lento. Já o uso do espaço de armazenamento pela M-tree não foi bom. A comparação com a SS-tree, que representa elementos com o mesmo espaço, mostra que os *splits* gerados na M-tree não são muito bons. O destaque aqui fica para a nova técnica `mM_MEAN_1`, que foi 8% menor e teve melhores resultados nos espaços de 64 e 128 dimensões (Seção 3.2.2) que a outra técnica, a `mM_RAD_1`. Nas avaliações de consulta, apesar de ter alto número de acessos nas consultas, a M-tree mostrou, com a `mM_RAD_1`, um tempo de consulta semelhante à SS-tree. No desempenho médio, a M-tree superou a SS-tree devido ao excelente resultado do tempo de construção dos índices. É importante salientar que, em um ambiente onde seja comum a inserção de novos elementos, o aspecto de tempo de construção, diretamente relacionado com a inserção de elementos, torna-se bastante importante.

- R-tree e R*-tree: os resultados de tempo de construção obtidos pelas estruturas da família R-tree foram os piores. Sobretudo com a R*-tree, que devido a complexidade do *split*, chega a ser 17 vezes mais lenta que a M-tree. O tamanho dos índices gerados é menor que os índices da M-tree, e menor que os índices da SR-tree com a R*-tree. Entretanto, nas avaliações de consulta é possível verificar o efeito causado pelo problema do *fanout* e principalmente pelo problema da sobreposição. A R-tree chega a ler mais de três vezes o número de páginas lidas pela SR-tree durante as consultas. Os resultados obtidos com o tempo de consulta foram da mesma forma os piores, evidenciando que a família R-tree não é escalável quanto ao número de dimensões.

- SS-tree: teve seu tempo de construção o segundo melhor, perdendo apenas para a M-tree devido a complexidade na geração dos centróides dos nós internos. Entretanto, a SS-tree tem *splits* bastante balanceados, resultando na alta, e melhor, utilização do espaço de armazenamento. Nas avaliações de consulta, a SS-tree teve um baixo número de páginas acessadas devido ao tamanho do índice. Mas o efeito causado pelo problema da sobreposição, ocasionado pelas esferas da SS-tree, não possibilitou melhores resultados em relação ao tempo de consulta. Ainda assim, a SS-tree teve o segundo melhor desempenho no tempo das consultas.

- SR-tree: entre todos os resultados, o único que não esteve perto dos melhores foi o tempo levado na construção dos índices. Na SR-tree, o tempo chegou a ser 80% maior que na M-tree, devido à necessidade de atualização de esferas e retângulos na hierarquia de organização dos elementos. Já no tamanho do índice gerado, a SR-tree foi apenas 16% maior que a SS-tree. Esse pode ser considerado um bom resultado, uma vez que a representação de um elemento na SR-tree é três vezes maior que na SS-tree. Nas consultas estão certamente os melhores resultados da SR-tree. O problema da sobreposição é reduzido com a utilização de esferas e retângulos, e o resultado direto disso é o número reduzido de páginas acessadas. Além disso, como um número menor de elementos é guardado em um nó, em relação às demais estruturas, o número de distâncias computadas é reduzido, implicando no melhor desempenho de tempo de consulta. A influência causada pelo problema do *fanout* foi pequena, quando comparada às demais estruturas, e sendo significativa no caso em que não foi possível construir os índices (64 dimensões e 4 Kb de página de disco).

Nos resultados obtidos, a SR-tree se mostrou a melhor estrutura com os conjuntos de dados reais. A M-tree mostrou poder alcançar bons resultados, dependendo da técnica de *split* utilizada.

Capítulo 4

Conclusão

Esta dissertação apresentou uma análise comparativa entre estruturas de indexação para dados hiperdimensionais. Conforme visto no Capítulo 1, a utilização de dados multidimensionais traz problemas para os bancos de dados tradicionais. Entre eles, a incapacidade das estruturas de indexação em organizar e recuperar adequadamente tais tipos de dados. A Seção 1.1.1 mostra a abordagem tomada na aplicação escolhida, banco de imagens. Nela, são tratados o espaço de cor utilizado (HSV), a forma de representação das imagens (histogramas de cores) e a forma de comparação das imagens (distância euclidiana—L_2).

No Capítulo 2 é mostrado como os vetores de características (representações das imagens) são utilizados na indexação espacial de dados, com a apresentação de estruturas como a R-tree, R*-tree, SS-tree e SR-tree. Uma forma diferente de processar os vetores de características é mostrado com a indexação adimensional de dados da M-tree. Nela, além das técnicas originais de *split*, foram sugeridas novas técnicas e também uma nova forma de reorganização.

O Capítulo 3 mostra os critérios utilizados na mensuração do desempenho das estruturas, sendo eles conjuntos de dados, ambiente de análise e parâmetros de análise. Além disso, é mostrada a análise realizada sobre os resultados experimentais, sendo ela feita com maior ênfase na estrutura M-tree devido à sua flexibilidade no uso de funções de (dis)similaridade.

Sem dúvida alguma a SR-tree obteve o melhor desempenho médio entre as estruturas hierárquicas. Entretanto, foi visto que quando o tamanho do conjunto de dados é pequeno, a busca linear de dados pode ser utilizada, obtendo um melhor desempenho em todos os aspectos em relação as demais estruturas hierárquicas.

É importante notar também o aspecto de flexibilidade das estruturas quanto à função de (dis)similaridade entre os vetores de características. As estruturas R-tree e R*-tree são de difícil adaptação [SK97]. Estruturas como a SS-tree e a SR-tree foram desenvolvidas para utilizar apenas funções relacionadas com a distância euclidiana ponderada. Já a

M-tree, a mais flexível de todas as estruturas, necessita apenas que alguns predicados sejam cumpridos. Uma vez que a comparação de vetores de características tem se tornado bastante complexa, estruturas com alta flexibilidade, como a M-tree, podem ser também consideradas importantes neste cenário de indexação de dados.

4.1 Contribuições

Dentre os resultados apresentados no Capítulo 3, podemos salientar as seguintes contribuições:

- Proposta das novas técnicas de *split* para a M-tree: `OVERLAP`, `mM_MEAN` e `MM_RAD2`. Sendo que a técnica `mM_MEAN_1` obteve os melhores resultados em dados hiperdimensionais entre as técnicas da M-tree.
- Proposta de uma nova forma de reorganização dos dados para a M-tree, chamada de Adaptive M-tree. Nos resultados, melhores utilizações do espaço de armazenamento foram alcançadas.
- Comparação extensiva sobre técnicas de *split* para a M-tree. Entre elas, as técnicas de *split* sugeridas como melhores pelos autores da estrutura e as técnicas propostas nesta dissertação.
- Comparação entre as estruturas R-tree, R*-tree, SS-tree, SR-tree e M-tree, com ênfase na avaliação do comportamento das estruturas com o crescimento do número de dimensões. O efeito do tamanho da página de disco sobre as estruturas também é avaliado nas comparações.
- Comparação de resultados de consulta mostrando que, em ambientes de alta complexidade computacional, o número de páginas acessadas pode ser falho em relação ao tempo real da consulta.
- Comparação da busca linear de dados em relação às estruturas hierárquicas. Na comparação é realizado um estudo das estruturas quanto à escalabilidade em relação ao tamanho do conjunto de dados.

4.2 Trabalhos Futuros

Seguindo a mesma linha de estudo realizada na presente dissertação, podem ser indicados os seguintes caminhos para trabalhos futuros:

- Entre as características que uma estrutura de indexação deve ter, a dinamicidade não foi estudada nesta dissertação. Neste possível trabalho é realizado a avaliação das estruturas após continuadas operações de inserção, alteração e remoção de elementos, verificando a organização e desempenho nas consultas.

- Devido à possibilidade das estruturas utilizarem conjuntos de dados da ordem de milhões de objetos, estudar possibilidades para a paralelização das operações sobre as estruturas apresentadas.

- Avaliar o comportamento das técnicas de *split* da M-tree com relação a outras métricas. Orientar o estudo para a avaliação da M-tree com métricas bastante complexas, bem como com técnicas simples, procurando traçar um comportamento das técnicas de *split* da M-tree.

- Avaliar estruturas mais recentes, como a Hybrid Tree [CM99], em relação às melhores estruturas apresentadas nesta dissertação.

- Avaliar as estruturas de indexação com relação a conjuntos de dados reais diferentes dos utilizados nesta dissertação.

Referências Bibliográficas

[BGRS99] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory*, pages 217–235, 1999.

[BKK96] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases*, pages 28–39, 1996.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, volume 19, 1990.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*, chapter 1. Addison Wesley Longman, 1999.

[CM99] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *Proceedings of the 15th International Conference on Data Engineering*. IEEE Computer Society, 1999.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, 1997.

[GG97] J. Gray and G. Graefe. The five-minute rule ten years later, and other computer storage rules of thumb. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, volume 26, pages 63–68, 1997.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–54, 1984.

[Hen98] A. Henrich. The LSD^h-tree: An access structure for feature vectors. In *Proceedings of the Fourteenth International Conference on Data Engineering*, pages 362–369. IEEE Computer Society, 1998.

[HNP95] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized search trees for databases systems. In *Proceedings of the 21st International Conference on Very Large Data Bases*, 1995.

[Knu72] D. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, 1972.

[KS97] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, volume 26, pages 369–380, 1997.

[LJF94] K.-I. Lin, H. V. Jagadish, and C. Faloutsos. The TV-tree: An index structure for high-dimensional data. *The International Journal on Very Large Data Bases*, 3(4):517–542, 1994.

[MG93] R. Mehrotra and J. E. Gary. Feature-based retrieval of similar shapes. In *Proceedings of the 9th International Conference on Data Engineering*, pages 108–115. IEEE Computer Society, 1993.

[NBE+93] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, and C. Faloutsos. The QBIC project: Querying images by content using color, texture, and shape. In *Storage and Retrieval for Image and Video Databases*, volume 1908 of *SPIE*, 1993.

[Nie90] H. Nieman. *Pattern Analysis and Understanding.* Springer Series in Information, 1990.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry.* Springer–Verlag, 1985.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79, 1995.

[SB91] M. J. Swain and D. H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[SBK92] B. K. Shoichet, D. L. Bodian, and I. D. Kuntz. Molecular docking using shape descriptors. *Journal of Computional Chemistry*, 13(3):380–397, 1992.

[SC96] J. R. Smith and S.-F. Chang. Visualseek: A fully automated content-based image query system. In *Proceedings of the ACM International Conference on Multimedia*, pages 87–98, 1996.
`http://www.ctr.columbia.edu/VisualSEEk`.

[SK97] T. Seidl and H.-P. Kriegel. Efficient user-adaptable similarity search in large multimidia databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 506–515, 1997.

[Smi97] J. R. Smith. *Integrated Spatial and Feature Image Systems: Retrieval Analysis and Compression.* PhD thesis, Graduate School of Arts and Sciences, Columbia University, 1997.

[SS94] M. Stricker and M. Swain. The capacity and the sensitivity of color histogram indexing. Technical Report 94-05, Departament of Computer Science, The University of Chicago, 1994.

[WJ96] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *Proceedings of the 12th International Conference on Data Engineering*, pages 516–523. IEEE Computer Society, 1996.

[WS82] G. Wyszecki and W. S. Stiles. *Color Science: concepts and methods, quantitative data and formulae.* John Wiley & Sons, 2nd edition, 1982.