Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Indexando Pontos em Espaços com Altas Dimensões Através dos Métodos de Indexação Espacial *K-D-B-Tree, R-Tree* e *TV-Tree*

Uma breve comparação analítica

Roberto Figueira Santos Filho
Agma Juci Machado Traina
Caetano Traina Júnior

Nº 86

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Abril/1999

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação e Estatística

Indexando Pontos em Espaços com Altas Dimensões Através dos Métodos de Indexação Espacial
K-D-B-Tree, R-Tree e *TV-Tree*
Uma breve comparação analítica[1]

Roberto Figueira Santos Filho
Agma Juci M. Traina
Caetano Traina Júnior

Av. Dr. Carlos Botelho, 1465
CEP 13.560-970 – São Carlos – São Paulo – Brasil

ICMC-USP, São Carlos

Abril de 1999

[1] Este trabalho teve apoio financeiro da FAPESP.

RESUMO

A elaboração do presente texto foi motivada pelo interesse em estudar a indexação de vetores de características, extraídos automaticamente de imagens (imagens médicas, em princípio), observada a peculiaridade de que cada classe de vetores (ou características) possua uma quantidade específica de elementos, podendo atingir valores superiores a 100. Os elementos destes vetores podem ser vistos como coordenadas de um ponto no espaço, cuja dimensão é determinada pela quantidade dos mesmos. Para cada classe de características é criada uma estrutura de índices para auxiliar no armazenamento e recuperação das imagens em um Sistema Gerenciador de Banco de Dados (SGBD).

Com este objetivo, apresentam-se os seguintes métodos de indexação espacial (MIE): *K-D-B-Tree*, *R-Tree* e *TV-B-Tree*. A estrutura de dados e um resumo do funcionamento destes métodos são relatados, bem como uma crítica à sua utilização no sentido de indexar pontos situados em espaços de altas dimensões (podendo ser superior a 100, como dito anteriormente). Estão relatadas, ainda: simulações com MIE *R-Tree* sobre uma base de dados composta apenas por vetores, variando a quantidade de elementos destes vetores; os resultados obtidos com estas simulações e as conclusões em torno destes resultados.

Sumário

1. INTRODUÇÃO 1
   1.1 Considerações Iniciais 2
   1.2 Objetivo 3
   1.3 Organização do Trabalho 5
2. K-D-B-TREE 6
   2.1 Introdução 7
   2.2 Estrutura de Dados 7
   2.3 Consulta 9
   2.4 Inserção 10
   2.5 Remoção 13
   2.6 Conclusões Parciais 14
3. R-TREE 17
   3.1 Introdução 18
   3.2 Estrutura de dados 18
   3.3 Consulta 20
   3.4 Inserção 21
   3.5 Remoção 22
   3.6 Particionamento 23
   3.7 Conclusões Parciais 26
4. TV-TREE 28
   4.1 Introdução 29
   4.2 Estrutura de Dados 30
   4.3 Consulta 32
   4.4 Inserção 33
   4.5 Particionamento 35
   4.6 Remoção 36
   4.7 Conclusões Parciais 36
5. TESTANDO A R-TREE EM VÁRIAS DIMENSÕES 37
   5.1 Dados Espaciais 38
   5.2 Material 39
   5.3 Método 40
   5.4 Resultados e Discussão 41
   5.5 Conclusões Parciais 49
6. CONCLUSÃO GERAL 50
   6.1 Resumo e Considerações Finais 51
   6.2 Proposta de Trabalhos Futuros 52

REFERÊNCIAS BIBLIOGRÁFICAS 53

APÊNDICE 56

Lista de Figuras

Figura 1: Exemplo de uma *2-D-B-Tree*. 9
Figura 2: Uma folha sendo particionada. 11
Figura 3: Um nó interno sendo particionado. 11
Figura 4: Exemplo de uma *B-Tree*. 13
Figura 5: A região A não pode ser unida a nenhuma outra. 14
Figura 6: Particionando uma região em função do hiperplano principal. 15
Figura 7: Exemplo de uma *R-Tree*. a) Representação em memória; b) Representação espacial com os *MBR*s. 19
Figura 8: Exemplo de um particionamento. a) Bom particionamento; b) Mau particionamento. 24
Figura 9: Exemplos de *TMBR*s (em forma de Diamantes e Esferas). a) $\alpha = 1$; b) $\alpha = 2$; c) $\alpha = 2$. 32
Figura 10: Exemplo de uma *TV*-1 (*MBRs* em forma de diamantes). 33
Figura 11: Exemplo de uma *TV*-2 (*MBRs* em forma de esferas). 33
Figura 12: Exemplo de inserção. O *TMBR D*1 é ajustado para incluir o ponto *P*. 33
Figura 13: Altura da árvore após 10.000 inserções. 42
Figura 14: Tempo total de inserção para1.000 pontos. 44
Figura 15: Nós Acessados/Consulta vs. Dimensão. 46
Figura 16: Nós Acessados/Remoção vs. Dimensão. 48

Abreviaturas

MBR	*Minimum Bounding Rectangle* (mínimo retângulo externo a um objeto espacial)
MIE	Método de Indexação Espacial
PACS	*Picture Archiving and Communication System* (Sistema de Armazenamento e Comunicação de Imagens)
PAM	*Point Access Methods* (Métodos de Acesso Pontual)
ms	mili segundo
kB	kilo *Byte*
UCP	Unidade Central de Processamento
RM	Ressonância Magnética
SAM	*Spatial Access Methods* (Métodos de Acesso Espacial)

1. INTRODUÇÃO

1. INTRODUÇÃO

1.1 Considerações Iniciais

O presente texto diz respeito ao estudo e à comparação de estruturas de dados que suportam a armazenagem e a recuperação de dados multidimensionais, tendo como aplicação inicial a indexação de características extraídas automaticamente de imagens em bancos de dados. Mais especificamente, estes dados correspondem a pontos situados no espaço, representando uma determinada característica extraída das imagens.

A dimensão do espaço de representação é determinada pelo número de elementos que compõem uma determinada característica, podendo assumir valores que superam a ordem de 200 elementos. Por exemplo, o histograma de cores obtido de uma imagem com *pixels* de 8 *bits* de profundidade corresponde a um vetor com 256 elementos, um para cada nível de intensidade de cinza da imagem.

Vem sendo bastante estudada a recuperação de imagens armazenadas em uma base de dados fundamentada em informações extraídas do seu conteúdo [TamuraH_84] [TrainaC_97] [WhiteD_95] [PetrakisE_97]. Estas informações, conhecidas por características das imagens (*features*), são estruturáveis e, portanto, podem ser catalogadas. Valendo-se da criação de múltiplas estruturas de índices sobre estas características, pode-se restringir drasticamente a coleção de imagens que fazem parte do *response set* de uma consulta [TrainaC_97]. Assim, apenas algumas (relativamente) poucas imagens precisam ser, de fato, comparadas para se obter o resultado de uma consulta baseada no conteúdo pictórico das mesmas. Cada estrutura indexa uma determinada característica, que tende a ser independente das demais, composta por um conjunto de elementos extraídos das imagens.

A quantidade de características que podem ser extraídas das imagens, bem como a combinação de diversos algoritmos de busca em uma mesma consulta pode ser elevado. Os elementos que compõem estas características, por sua vez, podem corresponder a um conjunto de valores numéricos. Em outras palavras, estes elementos podem ser *n-uplas* com valores numéricos, onde

n varia de acordo com a respectiva característica, apresentando um comportamento bastante imprevisível e podendo atingir a ordem de centenas. Embora a técnica de combinação de múltiplas estruturas de índices simultaneamente passe a ser interessante quando a seletividade aumenta, deve-se notar a complexidade de tal operação.

Para indexar estas *n-uplas*, as mesmas são tratadas como vetores de números que representam coordenadas de um ponto em um espaço *n*-dimensional, onde *n* corresponde à quantidade de números (elementos) que compõem estas *n-uplas*. Sobre estes pontos, é, então, utilizado algum método de indexação espacial[2] (MIE) para facilitar as consultas e o acesso às imagens.

1.2 Objetivo

Este texto é uma continuação do trabalho de mestrado iniciado em [SantosFilhoR_99] e apresentado ao ICMC-USP como exame de qualificação do curso de Mestrado. O seu objetivo é o estudo de métodos de indexação espacial (MIE) para auxiliar o armazenamento e a recuperação de imagens médicas em um Sistema de Armazenamento e Comunicação de Imagens (*Picture Archiving and Communication System* - PACS [DuttonG_90] [AlcocerP_96] [SclabassiR_96]). Pretende-se, porém, que os resultados deste trabalho não se restrinjam ao armazenamento de imagens.

Para tanto, apresenta-se a descrição e a análise do funcionamento de três dos diversos MIEs encontrados na literatura, procurando identificar um que melhor se adapte à aplicação proposta anteriormente. Serão levados em consideração: sua utilização para indexar pontos em dimensões variadas, podendo atingir valores superiores a 100, e sua integração a gerenciadores de bases de dados.

Em outras palavras, estes métodos de indexação serão avaliados em função de sua capacidade de adaptação a dimensões variadas e de como estas adaptações afetam os gerenciadores de base de dados, ou a si próprios.

[2] Em [SantosFilhoR_99] pode ser encontrada uma explicação dos termos que serão utilizados ao longo trabalho.

Observa-se, ainda, que a dimensão e a distribuição espacial dos pontos a serem indexados são dependentes do algoritmo utilizado para extrair as características das imagens. Portanto, nenhuma presunção visando melhorar o desempenho do MIE a ser utilizado pode ser feita a este respeito. Ou seja, o MIE a ser escolhido não deve depender da distribuição dos pontos no espaço e deve sofrer o menor impacto possível quando utilizado em altas dimensões[3].

Os MIEs que serão tratados aqui, bem como a motivação para sua inclusão neste trabalho, são apresentados a seguir:

- *K-D-B-Tree* [RobinsonJ_81]: é um método de acesso pontual (*Point Access Method*, ou *PAM*), sendo, em princípio, uma adaptação mesclando as características do MIE *K-D-Tree* [GaedeV_97] [SametH_90] com as da estrutura de indexação *B-Tree* [ComerD_79], tornando possível o armazenamento secundário.
- *R-Tree* [GuttmanA_84]: dentre os métodos de acesso espacial (*Spatial Access Method*, ou *SAM*), este é o mais referenciado na literatura. Apesar de ser voltado para dados espaciais, suas características não impedem o armazenamento de dados pontuais. Foi incluído neste estudo por sua importância na área de dados espaciais e, principalmente, por sua simplicidade. Estas qualidades o torna muito interessante como objeto de aprendizado, no que diz respeito a implementação e subseqüentes simulações.
- *TV-Tree* [LinK_94]: outro *PAM*, voltado para altas dimensões, com a peculiaridade de usar subconjuntos das coordenadas do ponto em níveis diferentes da árvore, ao invés de todas as coordenadas. Ou seja, a indexação é feita sobre apenas algumas coordenadas, adicionando-se outras quando o poder discriminatório que possuem for realmente necessário.

[3] A faixa de valores atribuída ao conceito de alta dimensão varia de acordo com o domínio de aplicação [SantosFilhoR_99]. Aqui, esta faixa corresponderá a valores acima de 100.

1.3 Organização do Trabalho

Este trabalho apresenta a seguinte organização:

- **capítulo 1**: são apresentadas as idéias que motivaram o desenvolvimento do trabalho e sua organização;
- **capítulo 2**: será apresentada a *K-D-B-Tree*;
- **capítulo 3**: será apresentada a *R-Tree*;
- **capítulo 4**: será apresentada a *TV-Tree*;
- **capítulo 5**: serão apresentados resultados de simulações com o MIE *R-Tree* utilizando uma base de dados composta por vetores representando pontos no espaço e as conclusões em torno destes resultados;
- **capítulo 6**: serão apresentadas as conclusões finais;
- **Referências Bibliográficas**.

2. K-D-B-TREE

2. K-D-B-TREE

Segue-se uma descrição do funcionamento e da estrutura de dados do MIE *K-D-B-Tree*. Ao final, será feita uma análise crítica visando sua aplicação em altas dimensões. Este capítulo foi baseado em [RobinsonJ_81], do mesmo autor da estrutura.

2.1 Introdução

Assim como a *B-Tree*, a *K-D-B-Tree* é uma árvore multivias com nós de tamanhos fixos e balanceada em função de sua altura, ou seja, todos os nós folha se encontram no mesmo nível. Cada nó corresponde a uma página de disco, tornando possível o armazenamento secundário. Ao contrário do que ocorre com a *B-Tree*, a *K-D-B-Tree* não garante a ocupação mínima de 50%. No entanto, na prática, espera-se que o número de nós com menos de 50% de ocupação seja relativamente pequeno, quando comparado com o número total de nós no índice [RobinsonJ_81].

A *K-D-B-Tree* particiona o espaço de forma semelhante à *K-D-Tree* [GaedeV_97] [SametH_90] [SantosFilhoR_99]: o espaço é particionado em duas regiões a partir de hiperplanos iso-orientados, sendo estas regiões disjuntas e com particionamento completo [GaedeV_97]. A letra *K*, no nome, indica a dimensão do espaço indexado pela estrutura. Assim, uma 3-*D-B-Tree* corresponde a uma *K-D-B-Tree* indexando pontos em um espaço de dimensão 3.

2.2 Estrutura de Dados

Seja p um ponto no espaço k-dimensional E: $domínio_0$ X $domínio_1$ X ... X $domínio_{k-1}$, seja r uma região neste espaço definida pelo conjunto de todos os pontos (x_0, x_1, ..., x_{k-1}) que satisfazem:

$$min_i \leq x_i < max_i, \qquad 0 \leq i < k\text{-}1,$$

para alguma coleção de min_i, $max_i \in domínio_i$. São armazenados, então, as coordenadas x_i dos pontos e os intervalos min_i e max_i das regiões (que passarão a ser tratados respectivamente por x_i,

min_i e max_i). Considerem-se definidos, também, os elementos ∞_i e $-\infty_i$ como sendo, respectivamente, o maior e o menor valores possíveis de um determinado $domínio_i$.

A *K-D-B-Tree* possui as seguintes características:

- todos os pontos indexados se encontram em nós folhas (ou simplesmente, folhas);
- cada folha contém entradas do tipo (*p*, *RId*), em que *RId* se refere (pode ser um ponteiro) a uma tupla na base de dados contendo os atributos não espaciais do objeto. E *p* corresponde às coordenadas do ponto;
- cada nó interno contém entradas do tipo (*r*, *CId*), em que *CId* (*Child Id*) é um ponteiro para um nó filho (ou simplesmente, filho). Ao passo que *r* corresponde à região que engloba as regiões ou os pontos existentes na página indicada por *CId*;
- *M*, *N* correspondem, respectivamente, ao número máximo de entradas em uma folha e ao número máximo de entradas em um nó interno.

As propriedades a seguir definem o comportamento da estrutura de dados da *K-D-B-Tree*:

- nenhum nó interno possui entradas com *Cid*s nulos e nenhum nó interno está vazio;
- o tamanho do caminho, em números de nós, desde o nó raiz (ou simplesmente, raiz) até uma folha, é o mesmo para todas as folhas. Ou seja, a *K-D-B-Tree* é balanceada em função da altura;
- em cada nó interno, as regiões r_i são disjuntas e sua união define outra região;
- se a raiz é um nó interno (não é uma folha), a união de suas r_n ($1 \leq n \leq N$) regiões corresponde a *E*: $domínio_0$ X $domínio_1$ X ... X $domínio_{k-1}$;
- considere-se uma entrada (r_n, CId_n) em um nó interno e considere-se, ainda, que seja outro nó interno o filho indicado por CId_n: então, a união de todas as regiões deste filho é, exatamente, r_n;
- ainda no caso anterior, se o filho indicado por CId_n for uma folha, todos os seus p_m ($1 \leq m \leq M$) pontos estão, necessariamente, contidos em r_n.

A Figura 1 exemplifica uma *K-D-B-Tree* para o espaço bidimensional.

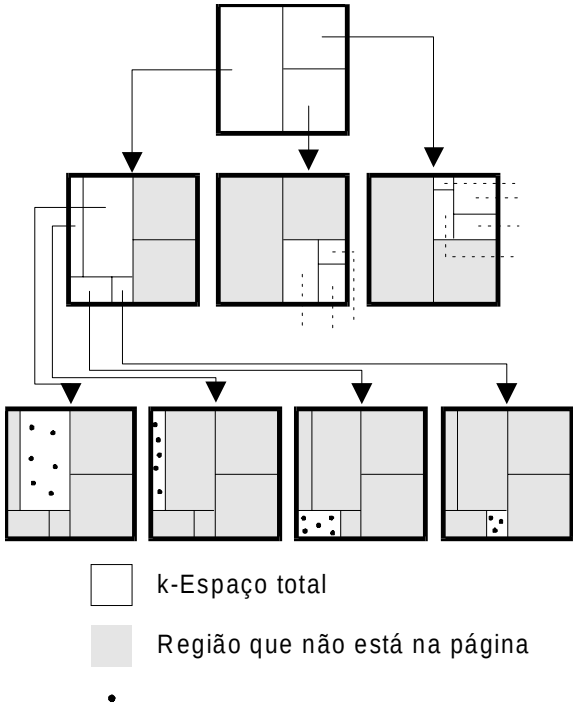


Figura 1: Exemplo de uma ***2-D-B-Tree***.

2.3 Consulta

Um exemplo de consulta em uma *K-D-B-Tree* pode ser: encontrar todos os pontos internos à região (que será tratada por **região de consulta** - *query region*) definida pelo produto cartesiano I_0 X I_1 X ... X I_{k-1}, onde I_i ($0 \leq i < k$-1) corresponde a um intervalo [min_i, max_i] ($0 \leq i < k$-1). Se alguns destes intervalos correspondem ao intervalo total daquele domínio, em outras palavras, se I_i corresponde a *domínio*$_i$, a consulta é tida como: **consulta por limites parciais** (*partial range query*). Se alguns dos intervalos são pontos, ou seja, $min_i = max_i$, a consulta é tida como: **consulta por coincidência parcial** (*partial match query*). Finalmente, se todos os intervalos são pontos a consulta é tido como: **consulta por coincidência exata** (*exact match query*).

A seguir, o algoritmo para obter todos os registros correspondentes a pontos situados internamente à região de consulta r_c, considere-se *nó*$_{raiz}$ a raiz e *nó*$_a$ o nó atual de busca:

Algoritmo 1: Consulta

1. Se *nó*$_{raiz}$ tem valor nulo, encerre a consulta;
2. Se *nó*$_a$ é uma folha:
   - 2.1. então, para cada entrada (p_m, RId_m), com p interno a r_c, recupere da base de dados o registro endereçado (apontado) por RId_m;
   - 2.2. senão, para cada entrada (r_n, CId_n), onde a interseção entre r_n e r_c é não vazio, faça *nó*$_a$ ser o nó endereçado (apontado) por CId_n e retorne ao passo 2.

2.4 Inserção

A inserção na *K-D-B-Tree* é similar à feita na *B-Tree*. Novos pontos são inseridos em folhas. Quando uma folha atinge seu número máximo de entradas (M), a mesma é particionada, propagando a inclusão para o seu nó pai. Se o mesmo ocorre com este pai, o particionamento é propagado recursivamente até atingir um nó com espaço para receber mais uma entrada, ou até que a raiz seja atingida. Se a raiz também está cheia, uma nova raiz é criada com duas entradas: cada uma recebe como filho um dos nós gerados pelo particionamento da antiga raiz.

O particionamento em uma *K-D-B-Tree* é definido da seguinte maneira:

- Seja x_i um elemento do $dominio_i$ sobre o qual o particionamento será feito;
- Seja r uma região qualquer, definida pelo produto cartesiano I_0 X I_1 X ... X I_{k-1};
- Se $x_i \notin I_i$:
    - então, a região r não é afetada pelo particionamento;
    - senão, considere-se $I_i = [min_i, max_i)$. Particionar a região r ao longo de x_i gera duas novas regiões:

        1. I_0 X ... X $[min_i, x_i)$ X ... X I_{k-1};
        2. I_0 X ... X $[x_i, max_i)$ X ... X I_{k-1}.

A região (1) é dita região esquerda, ao passo que a região (2) é dita região direita.

Seja x_i um elemento qualquer de um $domínio_i$. E seja $I_i = [min_i, max_i)$ o intervalo de uma região r qualquer neste mesmo domínio: diz-se que a região r está à esquerda (ou, que é região esquerda) de x_i se, e somente se, $x_i \geq max_i$; e diz-se que a região r está à direita (ou, que é região direita) de x_i se, e somente se, $x_i < max_i$. Considere-se, ainda, o ponto $y = (y_0, y_1, ..., y_{k-1})$: y diz-se estar à esquerda de x_i se, e somente se, $y_i < x_i$; e diz-se estar à direita de x_i, caso contrário.

Uma folha $nó_f$ é particionada ao longo de x_i gerando duas novas folhas: a $nó_{fE}$ (folha esquerda) e $nó_{fD}$ (folha direita). Em seguida, todos os pontos que estão à esquerda de x_i, são transferidos para

o nó $nó_{fE}$. Os pontos restantes, que estão à direita de x_i, são transferidos para o nó $nó_{fD}$. Por fim, o nó $nó_f$ é removido. A Figura 2 exemplifica o particionamento de uma folha.

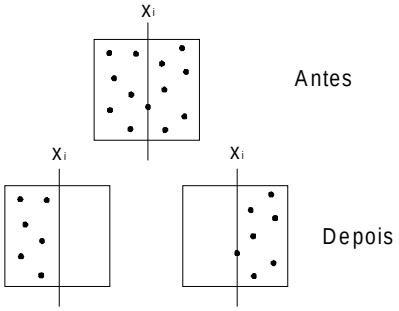


Figura 2: Uma folha sendo particionada.

Um nó interno $nó_{int}$ é particionado ao longo de x_i gerando dois novos nós internos: os nós $nó_{intE}$ (nó interno esquerdo) e $nó_{intD}$ (nó interno direito). Em seguida, eles são preenchidos com as regiões geradas a partir daquelas existentes no nó original, o qual será removido ao final deste processo. Sendo assim, o particionamento de $nó_{int}$ ao longo de x_i pode ser descrito da seguinte forma, considerando-se as entradas existentes neste nó representadas pelo par (r_n, CId_n):

Algoritmo 2: Particionamento

1. Se r_n está à esquerda de x_i, insira a entrada (r_n, CId_n) no $nó_{intE}$;
2. Se r_n está à direita de x_i, insira a entrada (r_n, CId_n) no $nó_{intD}$;
3. Caso contrário, ou seja, r_n é "cortada" por x_i:
   - 3.1. particione o filho endereçado por CId_n ao longo de x_i, resultando em dois novos filhos endereçados por CId_e e CId_d;
   - 3.2. particione a região r_n, resultando nas regiões r_e e r_d;
   - 1.1. adicione a entrada (r_e, CId_e) ao nó $nó_{intE}$ e a entrada (r_d, CId_d) ao nó $nó_{intD}$.

Observa-se que este procedimento é recursivo, vide o que ocorre no passo 3.1 do algoritmo anterior. A Figura 3 exemplifica o particionamento de um nó interno.

O algoritmo a seguir descreve os passos necessários para a inclusão do registro (p, RId), considere-se $nó_r$ como sendo a raiz e $nó_a$ como sendo o nó atual:

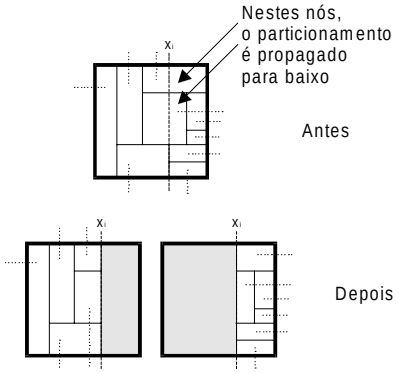


Figura 3: Um nó interno sendo particionado.

Algoritmo 3: Inserção

1. Se $nó_r$ é nulo, crie uma nova folha contendo a entrada (p, RId). Considere este nó como sendo $nó_r$ e encerre a inclusão;

2. Caso contrário, faça uma consulta do tipo coincidência exata sobre o ponto *p* obtendo a folha $nó_f$ onde o ponto deve ser inserido[4].

3. Consiedere $nó_f$ como sendo $nó_{atual}$. Se $nó_f$ ainda não atingiu o número máximo de entradas, insira a entrada (*p*, *TId*) em $nó_f$ e encerre;

4. Senão, escolha um domínio, $domínio_i$, e um elemento x_i deste domínio de tal forma que, particionando $nó_{atual}$ ao longo de x_i, dois novos nós contendo espaços livres para novas entradas serão gerados. Desde que o número de pontos ou regiões em um nó precisa ser reduzido de apenas uma unidade para gerar espaço para novas entradas, é fácil perceber que gerar estes nós é sempre possível. Particione $nó_{atual}$ ao longo de x_i, gerando os nós $nó_e$ e $nó_d$;

5. Se $nó_{atual}$ era a raiz, vá para o passo 6. Caso contrário, considere $nó_{pai}$ como sendo o pai de $nó_{atual}$ (o qual foi obtido durante a consulta feita no passo 2). Substitua a entrada (r_{atual}, CId_{atual}) referente a $nó_{atual}$ em $nó_{pai}$, onde r_{atual} é a região que engloba todo o nó $nó_{atual}$ e CId_{atual} é o seu endereço, pelas entradas (r_e, CId_e) e (r_d, CId_d), onde r_e e r_d são obtidas com o particionamento da região r_{atual} ao longo de x_i; e CId_e e CId_d são os endereços dos nós $nó_e$ e $nó_d$, respectivamente. Se este passo provocar uma super lotação do nó $nó_{pai}$, repita o passo 4 com $nó_{atual}$ sendo o $nó_{pai}$;

6. Crie um novo nó $nó_{raiz}$, contendo as seguintes entradas: (r_e, CId_e) e (r_d, CId_d), onde r_e e r_d são obtidas com o particionamento da região r_{atual} ao longo de x_i; e CId_e e CId_d são os endereços dos nós $nó_e$ e $nó_d$, respectivamente.

A escolha de x_i e $domínio_i$ pode ser feita ciclicamente, semelhante à como é feita na *K-D-Tree*. ou seja, em cada nó é armazenada uma variável *dp* (domínio de particionamento), a qual recebe, inicialmente, o valor zero, quando da criação do primeiro nó da *K-D-B-Tree*. Quando um nó é particionado, um elemento do domínio $domínio_{dp}$ é escolhido e as novas páginas passam a ter o

[4] Samet em [SametH_90] comenta que é comum a ocorrência de colisões em dados espaciais. Caso ocorra uma colisão, a mesma deve ser tratada de alguma forma: emissão de mensagem de erro, criação de uma lista encadeada

valor de *dp* igual a (*dp* + 1) módulo *k*. A Figura 3, antes do particionamento, pode ser vista como um exemplo de uma *K-D-B-Tree* gerada por este critério de escolha de $domínio_i$. Porém, o passo 3 do Algoritmo 2 (*Particionamento*) pode provocar algumas exceções (vide nó direito da Figura 3, após ter sido particionada).

Outras alternativas para o critério de escolha de x_i e $domínio_i$, bem como suas implicações, são discutidas em [RobinsonJ_81].

2.5 Remoção

A estrutura de dados da *K-D-B-Tree*, da forma como foi definida na seção 2.2, não exclui a possibilidade de existência de folhas vazias. Além disto, não faz nenhuma imposição acerca de ocupação mínima. Portanto, o algoritmo de remoção pode ser simplificado para: encontrar o registro a ser removido através de uma consulta por coincidência exata e removê-lo do nó em que se encontra.

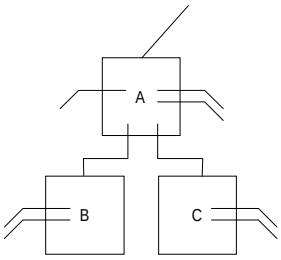


Figura 4: Exemplo de uma ***B-Tree***.

Porém, este algoritmo pode levar a baixos índices de ocupação. Para resolver este problema, a *K-D-B-Tree* utiliza técnicas de reorganização semelhantes às utilizadas em *B-Trees*. Considerando a subárvore de uma *B-Tree* na Figura 4, se os ponteiros no nó *A* para os nós *B* e *C* são adjacentes, dois tipos de reorganização são possíveis: concatenação, onde os nós *B* e *C* são fundidos em apenas um; e redistribuição, onde as entradas dos nós *B* e *C* são redistribuídas entre eles. Estes tipos de reorganização são tidos como locais, visto que envolvem apenas o nó *A* e seus filhos.

Quase o mesmo tipo de reorganização local pode ser aplicada a uma *K-D-B-Tree*, desde que a união das regiões correspondentes aos nós *B* e *C* seja, também, uma região (segundo as propriedades apresentadas na seção 2.2). Porém, se os nós B e C não forem folhas, a redistribuição entre eles pode levar ao passo 3 do Algoritmo 2.

contendo as colisões, etc..

Outro problema é que nem sempre a união de apenas duas regiões gera outra região. A região A na Figura 5 ilustra este problema. A solução pode ser unir mais do que duas regiões (A, B e C, no exemplo citado) e particionar a nova região em duas ou mais, se desejável.

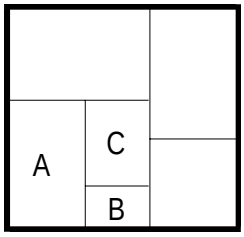


Figura 5: A região A não pode ser unida a nenhuma outra.

Segundo Robinson [RobinsonJ_81], até a data da citada publicação, nenhum tipo de reorganização havia sido implementado e o problema ainda estava sendo estudado.

2.6 Conclusões Parciais

Uma variação que pode ser feita com uma *K-D-Tree* é, a cada nó, armazenar apenas a coordenada referente à dimensão que está sendo utilizada para particionar o espaço àquele nível da árvore. As outras coordenadas, que poderiam ser consideradas como atributos não espaciais do ponto [SametH_90] poderiam estar, por exemplo, armazenadas em disco.

Como conseqüência, a estrutura de um nó seria independente da dimensão do espaço a ser indexado eo número de comparações feitas a cada nível seria constante e igual a 1. Ou seja, a *K-D-Tree* não deteriora em função da dimensão, mantendo o espaço necessário para armazenar um nó, e o número de comparações feitas para decidir que caminho da árvore deve ser percorrido, constantes e independentes da dimensão de sua aplicação.

Até então, os únicos problemas que a *K-D-Tree* apresentava eram: não ser balanceada, podendo degenerar em uma lista encadeada a depender da ordem em que os dados fossem inseridos; e não suportar armazenamento secundário. Antes da elaboração do presente texto, acreditava-se que a *K-D-B-Tree* solucionava estes problemas, balanceamento e armazenamento secundário, mantendo as características relativas à dimensão, citadas anteriormente. Foi esta, alias, a principal motivação para o seu estudo.

No entanto, como pode ser observado na seção 2.2, a *K-D-B-Tree* apresenta uma dependência em relação à dimensão, visto que o tamanho do nó e o número de comparações feitas a cada instante

variam linearmente em função da dimensão. Esta variação ocorre porque são armazenados, para cada dimensão, os limitantes inferior e superior de cada uma das regiões que compõem um nó.

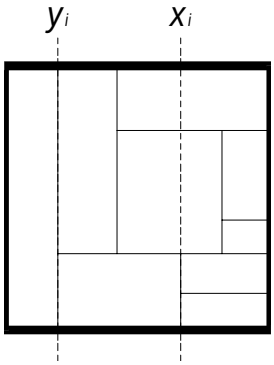


Figura 6: Particionando uma região em função do hiperplano principal.

Para justificar o armazenamento destes intervalos os seguintes pontos devem ser observados:

- Tendo em vista que a *K-D-Tree* não é balanceada, assim como à *B-Tree*, o que proporciona um balanceamento à *K-D-B-Tree* é a forma como é feito o particionamento de um nó, quando este atinge a quantidade máxima de entradas.
- Uma opção seria, supondo uma *K-D-B-Tree* gerada ciclicamente, escolher o hiperplano principal de uma região, ou seja, aquele que particiona o espaço delimitado por seus intervalos em duas outras regiões distintas, sem a necessidade de propagar o particionamento árvore abaixo. Usando como exemplo a Figura 6, o particionamento poderia ser feito pelo hiperplano y_i e não pelo x_i.
- Porém, escolher sempre o hiperplano principal não garante o balanceamento da árvore, forçando, por vezes, o particionamento da região por outros hiperplanos, tais como x_i.
- Visto que as regiões podem ser particionadas por hiperplanos diferentes do principal, torna-se necessário o armazenamento dos intervalos limitantes em cada dimensão. Isto ocorre para que seja possível delimitar as regiões geradas por este tipo de particionamento (vide região direita apresentada na Figura 3 após ter sido particionada por x_i)

Outra justificativa para armazenar estes limitantes é que, ao contrário da *K-D-Tree*, a *K-D-B-Tree* cresce de baixo para cima e, como conseqüência dos pontos colocados anteriormente, para que o particionamento de um nó seja feito fora do hiperplano principal, há a necessidade de se conhecer os limites de toda a região em questão. Pois:

- Supondo que seja particionado um nó interno de uma *20-D-B-Tree*, onde o número máximo de entradas por nó é 5, e que a dimensão de particionamento indicada pela variável *dp* corresponda à $dimensão_6$, só será possível definir o valor do elemento x_6 se

forem conhecidos os limites da região sob esta dimensão. Caso contrário, uma região definida em um nó superior pode ser invadida. Além disso, em geral, x_6 é escolhido como sendo a mediana do número de entradas ou de pontos existentes no nó.

O problema gerado pela necessidade de armazenar todos os limites de um região é que, para um nó de tamanho fixo, correspondente a uma página de disco, aumentar a dimensão implica aumentar o espaço ocupado por uma entrada, o que leva a diminuir a quantidade de entradas por nó. Com isso, além de aumentar a altura da árvore, pode-se chegar ao ponto em que apenas uma entrada ocupe o nó inteiro, degenerando a *K-D-B-Tree* para uma lista encadeada. Pode acontecer, ainda, que nenhuma entrada caiba em um nó, tornando necessários ajustes na estrutura de dados.

Além dos problemas encontrados em função da variação da dimensão, a *K-D-B-Tree* apresenta problemas quanto a: a ausência de uma política bem definida de reorganização, a necessidade de propagar o particionamento árvore abaixo e a possibilidade de que a taxa de ocupação caia a valores indesejáveis.

3. R-TREE

3. R-TREE

Este capítulo foi baseado em [GuttmanA_84] e apresenta o funcionamento e a estrutura de dados do MIE *R-Tree*. Ao final será feita uma análise crítica, visando sua aplicação em altas dimensões. A notação utilizada pelo autor será mantida.

3.1 Introdução

O MIE *R-Tree* é, sem sombra de dúvidas, o *SAM* mais referenciado na literatura, sendo o mais importante entre os que aplicam o conceito de *MBR* (*Minimum Bounding Rectangle*, ou retângulo limítrofe mínimo). Consiste basicamente de um *SAM* que faz decomposição espacial irregular recursiva, organizada em diretórios hierárquicos.

3.2 Estrutura de dados

A *R-Tree* é uma árvore balanceada em função da altura, sendo que as folhas contém ponteiros para os atributos não espaciais dos objetos. Os nós correspondem a páginas de disco, com as seguintes características (ver Figura 7, obtida em [GuttmanA_84]):

- cada folha contém entradas do tipo (*I*, *Tid*), em que *Tid* se refere (pode ser um ponteiro) a uma tupla na base de dados, contendo os atributos não espaciais do objeto. E *I* corresponde ao *MBR* deste objeto;
- cada nó interno contém entradas do tipo (*I*, *Cid*), em que *Cid* é um ponteiro para algum filho. Ao passo que *I* corresponde ao *MBR* que cobre todas as regiões descritas pelos *MBR* dos nós inferiores;
- Considere-se M o número máximo de entradas em um nó. Assuma-se, então, $m \leq M/2$ o parâmetro que especifica o número mínimo de entradas em um nó.

A *R-Tree* apresenta as seguintes propriedades:

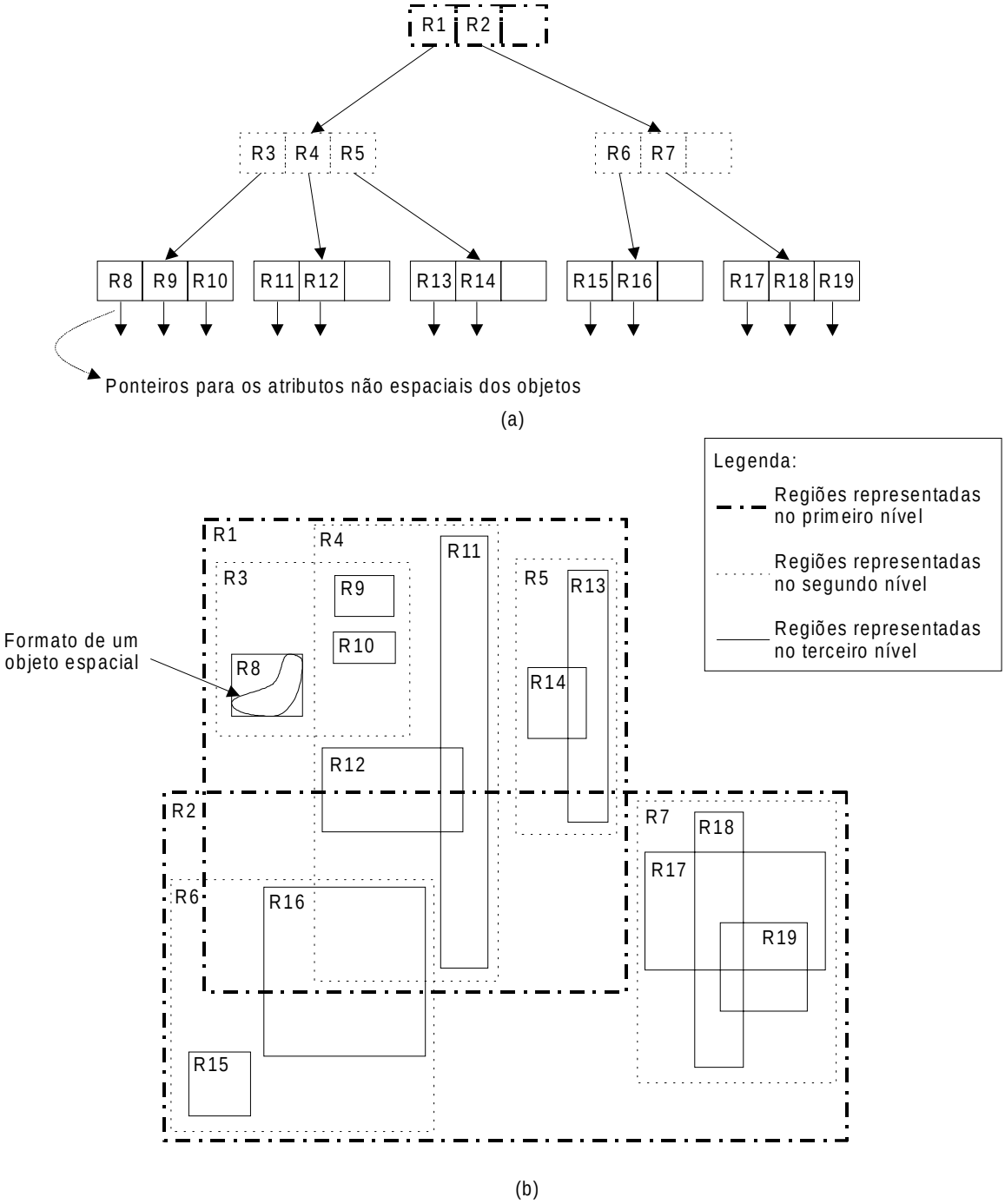


Figura 7: Exemplo de uma ***R-Tree****. a) Representação em memória; b) Representação espacial com os MBRs.*

- Toda folha contém entre *m* e *M* entradas, exceto a raiz;
- Para cada entrada do tipo (*I*, *Tid*) em uma folha, *I* é o *MBR* que contém o objeto descrito por *Tid*;
- Todo nó interno tem entre *m* e *M* filhos, a menos que seja a raiz;

- Para cada entrada do tipo (*I*, *Cid*) em um nó interno, *I* é o *MBR* que engloba todos os *MBRs* de seus filhos;
- A raiz tem no mínimo dois filhos, a menos que seja uma folha;
- Todas as folhas aparecem no mesmo nível.

A altura de uma *R-Tree* com *N* registros é no máximo $|\log_m N|$-1, porque o número mínimo de entradas por nó é *m*. O número máximo de nós é $[N/m] + [N/m^2]+...+1$. A pior taxa de ocupação para todos os nós, exceto para raiz, é *m*/*M*. Os nós tendem a ter mais do que *m* entradas, o que irá diminuir a altura da árvore e melhorar a taxa de ocupação.

3.3 Consulta

A consulta na *R-Tree* é similar à feita na *B-Tree*. Pode ser necessário percorrer mais do que uma subárvore de um determinado nó, aumentando o custo da consulta. Os algoritmos de inserção e particionamento devem ser projetados de forma a minimizar este aumento.

Considere-se *E.I* como sendo o *MBR* referente à entrada *E*, e *E.p* como sendo um *Tid* ou *Cid*. O seguinte algoritmo descreve uma consulta:

Algoritmo 4: Consulta

Encontra todas as entradas cujos *MBR*s coincidem com um intervalo espacial de consulta *S*, a partir da *R-Tree* cuja raiz é indicada por *T*:

1. Se *T* é um nó interno, percorra todas as suas entradas *E* para verificar se *E.I* faz interseção com *S*. Para todas as entradas que fazem interseção, execute este passo novamente, considerando *T* como sendo a subárvore apontada por *E.p*;
2. Se *T* é uma folha, percorra todas as entradas *E* para verificar se *E.I* faz interseção com *S*. Se as entradas fazem interseção, então compõem o conjunto resposta.

3.4 Inserção

Inserção na *R-Tree* é similar à feita na *B-Tree*, sendo que novas entradas são acrescentadas nas folhas, e os nós que ultrapassam a capacidade máxima de entradas, *M*, são particionados. O particionamento é, então, propagado para os pais.

Os seguintes algoritmos descrevem a inserção:

Algoritmo 5: Inserção

Insere uma nova entrada *E* em uma *R-Tree*:

1. Use *EscolheFolha* para obter a folha *L* onde *E* deve ser incluído;
2. Se *L* tem espaço para mais uma entrada, insira *E*. Caso contrário, use *ParticionaNo* para obter *L* e *LL* contendo *E* e as antigas entradas de *L*;
3. Use *AjustarÁrvore* passando *L* como parâmetro para propagar as mudanças necessárias árvore acima. Se *L* foi particionado, passe *LL* também;
4. Se a propagação do particionamento ao longo da árvore provocou o particionamento da raiz, crie uma nova raiz tendo os dois nós gerados como filhos.

Algoritmo 6: EscolheFolha

Seleciona a folha em que *E* será inserida:

1. Faça *N* corresponder à raiz;
2. Se *N* é uma folha, retorne *N*;
3. Senão, faça *F* corresponder à entrada de *N* cujo *MBR F.I* necessita de menor expansão para englobar *E.I*. No caso de empate, escolha a entrada com *MBR* de menor área;
4. Faça *N* corresponder ao filho de *F* referenciado por *F.p*. Retorne ao passo 2.

Algoritmo 7: AjustarÁrvore

Percorre árvore acima, a partir de uma folha, ajustando os *MBR*s e propagando os particionamentos:

1. Faça $N = L$. Se L foi particionado anteriormente, faça NN corresponder ao nó resultante do particionamento;
2. Se N é a raiz, encerre;
3. Considere P como sendo o pai de N e E_N como sendo a entrada de P referente a N. Ajuste $E_N.I$ de forma a englobar todos os *MBR*s de N;
4. Se NN foi gerado, crie uma nova entrada E_{NN}, com $E_{NN}.p$ referenciando NN e $E_{NN}.I$ englobando todos os *MBR*s de NN. Se houver espaço em P, insira E_{NN}. Caso contrário, use *ParticionaNo* para obter P e PP contendo E_{NN} e as antigas entradas de P;
5. Faça $N = P$. Se N foi particionado, faça $NN = PP$. Retorne ao passo 2.

O algoritmo *ParticionaNo* será descrito na seção 3.6.

3.5 Remoção

Os seguintes passos são executados para se remover a entrada E:

Algoritmo 8: Remove

Remove uma entrada E de uma *R-Tree*:

1. Use *EncontraFolha* para obter a folha L que contém E. Se a entrada E não foi encontrada, encerre;
2. Remova E de L;
3. Execute *CondensaÁrvore*, passando L como parâmetro;
4. Se a raiz ficou com apenas um filho após o passo anterior, transforme este filho na nova raiz.

Algoritmo 9: EncontraFolha

Encontra a folha que contém a entrada E, a partir da *R-Tree* cuja raiz é indicada por T:

1. Se T é um nó interno, percorra todas as suas entradas F para verificar se $F.I$ faz interseção com $F.I$. Para cada entrada que faz interseção, execute este passo

novamente considerando *T* como sendo a subárvore apontada por *F.p*, até que *E* seja encontrado ou que todas as subárvores tenham sido checadas;

2. Se *T* é uma folha, percorra cada entrada para verificar se corresponde à entrada *E*. Se *E* for encontrado, retorne T.

Algoritmo 10: CondensaÁrvore

Dada uma folha *L*, da qual uma entrada foi removida, eliminar esta folha caso tenha um número de entradas menor do que o mínimo *m*, e reinserir suas entradas. Propagar a eliminação desta folha, se necessário. Ajustar todos os *MBR*s árvore acima:

1. Faça $N = L$. Considere *Q* como sendo um conjunto que irá conter os nós eliminados, inicialmente vazio;
2. Se *N* é a raiz, vá para o passo 6. Caso contrário, considere *P* como sendo o pai de *N* e E_N como sendo a entrada a entrada de *P* referente a *N*.
3. Se *N* tem menos do que *m* entradas, elimine E_N de *P* e insira em *Q*;
4. Se *N* não foi eliminado, ajuste $E_N.I$ de forma a englobar todos os *MBR*s de *N*;
5. Faça $N = P$. Retorne ao passo 2.
6. Reinsira todas as entradas existentes em *Q*. As entradas que faziam parte de folhas, são reinseridas em folhas, da mesma forma que no Algoritmo 5 (*Inserção*). Mas, entradas que faziam parte de nós internos têm que ser reinseridas no mesmo nível em que se encontravam, de forma que as folhas das subárvores referentes às suas entradas fiquem no mesmo nível que as outras folhas da árvore principal.

3.6 Particionamento

O particionamento deve ser feito de forma a evitar que os dois nós gerados tenham sempre de ser percorridos quando da execução de uma consulta. Um nó deve ou não ser percorrido, conforme o seu *MBR* total faça ou não interseção com o intervalo de consulta. Para tanto, a área total dos *MBR*s de cada um dos nós gerados a partir do particionamento, deve ser a menor possível. Como pode ser observado, as áreas dos *MBR*s (linhas tracejadas) na Figura 7a, são maiores do que as dos *MBR*s (linhas tracejadas) da Figura 7b.

A seguir, serão resumidas três propostas para executar o particionamento de um nó, com *M*+1 entradas, de forma a gerar dois outros nós com *MBR*s de áreas mínimas:

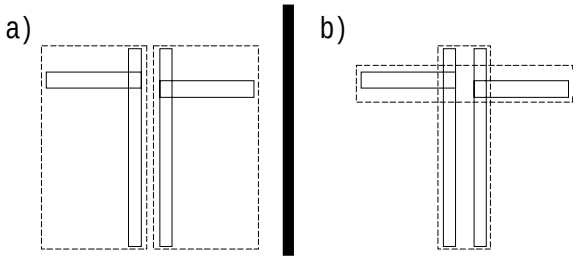


Figura 8: Exemplo de um particionamento. a) Bom particionamento; b) Mau particionamento.

3.6.1 Algoritmo Exaustivo

A forma mais direta de se encontrar os dois nós de áreas mínimas é testar todas as possibilidades de agrupamento das *M*+1 entradas e escolher a melhor. Porém, o número de possibilidades é aproximadamente 2^{M-1}. Desde que um valor razoável para *M* é 50 [GuttmanA_84], o número de agrupamentos possíveis é consideravelmente grande.

3.6.2 Algoritmo Quadrático

A proposta aqui é encontrar uma combinação de entradas com área **pequena**. Não se garante, porém, que seja esta **a melhor** combinação possível. O custo é quadrático em relação a *M* e linear em relação ao número de dimensões.

O primeiro passo consiste em obter duas entradas, dentre as *M*+1, como sementes (primeira entrada) para compor os dois grupos que serão gerados após o particionamento. Estas sementes correspondem às entradas que iriam gerar a maior área, quando comparadas duas a duas, caso fossem colocadas no mesmo nó. Ou seja, subtraindo a área do *MBR* que engloba estas duas entradas da soma das áreas das próprias entradas, o valor seria maior do que o resultante do mesmo cálculo aplicado aos outros pares de entradas. As entradas restantes são escolhidas e inseridas como descrito a seguir:

Algoritmo 11: ParticionamentoQuadrático

Dividir um grupo de *M*+1 entradas em dois grupos:

1. Execute *ObtémSementes* para escolher as duas primeiras entradas e inclua uma em cada grupo;

2. Se todas as entradas já foram inseridas, encerre. Se um dos grupos possui tão poucas entradas, de forma que todas as outras tenham que ser nele inseridas afim de que passe a ter o número mínimo *m* de entradas, insira todas e encerre;

3. Execute *ObtémPróximaEntrada* para obter a próxima entrada a ser inserida. Insira esta entrada no grupo que cujo *MBR* total terá que ser menos expandido após a inserção. Na ocorrência de empate, a inserção é feita naquele de menor área total. Depois, naquele com menor número de entradas e por fim, no que apresenta os dois casos. Retorne ao passo 2.

Algoritmo 12: ObtémSementes

Seleciona duas entradas para serem as primeiras de cada grupo:

1. Para cada par de entradas, E_1 e E_2, componha um *MBR J* que engloba E_1.I e E_2.I. Calcule: d = área(J) - área(E_1.I) - área(E_2.I);

2. Escolha o nó que apresenta o maior valor de d.

Algoritmo 13: ObtémPróximaEntrada

Escolhe uma entrada, dentre as que restam, para ser testada em um grupo:

1. Para cada entrada *E* que ainda não foi inserida em um dos grupos, calcule d_1 = aumento necessário à área do *MBR* total do primeiro grupo para incluir *E.I*. Considerando, agora, o segundo grupo, calcule d_2 da mesma forma;

2. Escolha qualquer entrada que apresente a máxima diferença entre d_1 e d_2.

3.6.3 Algoritmo Linear

Este algoritmo apresenta custo linear em relação a *M* e ao número de dimensões. É basicamente idêntico ao anterior, mas usa uma versão diferente do Algoritmo 12 (*ObtémSementes*). Além disto, o Algoritmo 13 (*ObtémPróximaEntrada*), escolhe qualquer entrada dentre as que ainda restam.

Algoritmo 14: ObtémSementesLinear

Seleciona duas entradas para serem as primeiras de cada grupo:

1. Ao longo de cada dimensão, encontre a entrada cujo *MBR* tem o lado inferior mais alto e a que tem o lado superior mais baixo. Armazene a distância entre eles;
2. Normalize as distâncias dividindo pelo tamanho de todo o conjunto ao longo da dimensão correspondente;
3. Escolha o par com a maior distância normalizada, dentre todas as dimensões.

3.7 Conclusões Parciais

Por serem os dados organizados através de *MBR*s, a *R-Tree* apresenta a mesma dependência em função da variação da dimensão encontrada na *K-D-B-Tree*. Conseqüentemente, cabe considerar o que foi dito na seção 2.6 quanto aos problemas da quantidade de entradas por nó.

Além deste problema, como os *MBR*s de uma *R-Tree* não são disjuntos, mais do que uma subárvore virá a ser percorrida durante uma consulta. Ocorre, ainda, que nem todas as subárvores percorridas possuem objetos que satisfaçam a consulta, diminuindo o índice de eficiência baseado na relação objetos encontrados vs. nós visitados.

Em contrapartida, pode-se destacar que: a *R-Tree* apresenta uma política bem definida de reorganização de dados, o particionamento de um nó interno não precisa ser propagado árvore abaixo, uma taxa de ocupação mínima é garantida e podem ser armazenados dados pontuais e espaciais.

Existem duas variações da *R-Tree* que procuram minimizar o problema da sobreposição dos *MBR*s: a R^*-*Tree* [BeckmannN_90] e a R^+-*Tree* [GaedeV_97] [SametH_90]. A primeira apresenta modificações no algoritmo de particionamento de um nó de forma a diminuir a incidência de sobreposições, mas não elimina totalmente este problema. Assinala Berchtold [Berchtold_96] que o aumento da dimensão resulta em maior número de sobreposições, tanto na *R-Tree*, quanto na R^*-*Tree*.

Na R^+-*Tree*, por sua vez, quando há a necessidade de sobreposição dos *MBR*s, os objetos que forçam esta sobreposição são “cortados”, gerando *MBR*s disjuntos. No entanto, o

particionamento também é propagado árvore abaixo e a inserção e remoção são mais custosas [GaedeV_97].

4. TV-TREE

4. TV-TREE

Este capítulo foi baseado em [LinK_94] e apresenta o funcionamento e a estrutura de dados do MIE *TV-Tree*. Ao final será feita uma análise crítica, visando sua aplicação em altas dimensões. A notação utilizada pelo autor será mantida.

4.1 Introdução

A *TV-Tree* é um *PAM* que trabalha com o conceito de *MBR* armazenando os dados apenas em nós folhas e os seus respectivos *MBR*s em nós internos de uma árvore. Esta estrutura propõe contrair e estender dinamicamente o vetor de características (coordenadas do ponto). Em outras palavras, ao invés de usar um número fixo de coordenadas do ponto, a *TV-Tree* procura usar, em níveis mais altos da árvore, uma quantidade menor de coordenadas, tidas como mais relevantes.

Quando novos objetos são inseridos na árvore, pode vir a ser necessário o uso de outras coordenadas para discriminar melhor estes objetos. A idéia principal é acrescentar novas coordenadas apenas quando são realmente necessárias, suavizando, assim, os efeitos relativos à dimensionalidade do espaço. Um dos benefícios adquiridos é o de tornar possível uma quantidade maior de entradas nos nós situados em níveis mais altos da estrutura.

A sigla *TV* vem de *Telescopic-Vector tree* (ou, árvore de vetores telescópicos), fazendo uma analogia ao movimento de contrair e estender um telescópio. Ainda com esta idéia, a *TV-Tree* chama de "**função telescópica**" a função utilizada para contrair e estender as coordenadas dos pontos.

Esta função é definida da seguinte maneira:

- dado um vetor $\vec{x}$, n X 1, e uma matriz A_m, m X n (m ≤ n), chamada de matriz de contração, o vetor $A_m \vec{x}$, m X 1, é uma m-contração de $\vec{x}$. Uma seqüência de tais matrizes, com m = 1, 2, ..., descreve uma função telescópica, desde que a seguinte condição seja satisfeita: se as m_1-contrações de dois vetores, $\vec{x}$ e $\vec{y}$, são iguais, então, também são iguais as respectivas m_2-contracões, para qualquer $m_2 \leq m_1$.

O exemplo mais simples para esta função é a que utiliza a seqüência das A_m matrizes diagonais, que simplesmente trunca o vetor. Esta função foi usada para gerar os exemplos descritos neste capítulo. Em [LinK_94] podem ser encontrados outros exemplos.

Para a *TV-Tree* o termo *MBR* não se refere a retângulos, mas sim a regiões do espaço, pois podem ser usadas regiões como: (hiper)esferas, (hiper)retângulos, (hiper)cubos, regiões limitadas em algumas dimensões e em outras não, etc. O termo *MBR* fica, então, com o seguinte significado: *MBR* - *Minimum Bounding Region* (ou, região limítrofe mínima).

Dentre os formatos para estas regiões o de representação mais fácil é a esfera, pois exige que apenas o centro e o raio sejam armazenados. Uma esfera de raio *r* é definida como o conjunto de pontos cuja distância Euclideana ao centro da esfera é menor ou igual a *r*. Vale notar que a distância Euclideana é um caso especial da métrica L_p, com $p = 2$:

$$L_p(\vec{x},\vec{y}) = \left[\sum_i (x_i - y_i)^p\right]^{1/p} \tag{1}$$

Para a L_1 (ou distância *Manhattan*, ou distância *city-block*), o formato gerado eqüivale a um diamante; para L_∞, o formato gerado eqüivale a um cubo.

Uma esfera L_p de centro $\vec{c}$ e raio *r*, é definida pelo conjunto de pontos cuja a distância L_p ao centro $\vec{c}$ é menor ou igual a r. Os algoritmos apresentados adiante suportam qualquer métrica L_p, sem necessitar alterações. O único algoritmo que depende do formato utilizado é o que calcula o *MBR* de um conjunto de pontos.

4.2 Estrutura de Dados

O nó da *TV-Tree* representa o *MBR* dos nós descendentes. Cada região é representada por um centro, que é definido por um vetor telescópico (gerado pela função telescópica), e um valor escalar correspondente ao raio. O termo *TMBR* (*Telescopic Minimum Bounding Region*) será usado para designar um *MBR* cujo centro é um vetor telescópico.

Uma esfera telescópica L_p com centro $\vec{c}$, raio r e dimensionalidade d, das quais α estão ativas, é definida pelo conjunto de pontos $\vec{y}$, tais que:

$$c_i = y_i,\ i = 1, ..., d - \alpha \tag{2}$$

e

$$r^p \geq \sum_{i=d-\alpha+1}^{d} \left(c_i - y_i\right)^p \tag{3}$$

Na Figura 9a, por exemplo, D2 tem a primeira dimensão inativa e a segunda ativa. Apenas a primeira dimensão de D1 está ativa. A dimensionalidade de D1 é 1 (apenas a primeira dimensão foi levada em consideração para determinar D1) e a dimensionalidade de D2 é 2 (as duas dimensões foram consideradas).

O conceito de dimensões **inativas** tem como base a seguinte idéia:

- à medida em que a árvore aumentar em altura, algumas folhas serão compostas por pontos que possuem, eventualmente, as mesmas k primeiras coordenadas. Caso isto ocorra, os *TMBR* iniciais consideram que as k primeiras dimensões destes pontos são inativas, pois não trazem nenhuma contribuição no sentido de discriminá-los.

Como conseqüência, os algoritmos são definidos de forma que as dimensões ativas sejam sempre as últimas. Além disso, o valor de α pode ser ajustado garantindo que todos os primeiros *TMBR*s de uma árvore possuam este mesmo valor. O valor de α é, na verdade, um parâmetro da *TV-Tree*. Logo, o número de dimensões ativas (α) de uma *TV-Tree*, é o número de dimensões ativas de todos os seus *TMBR*s.

A notação *TV*-1 indica uma *TV-Tree* com α=1. A Figura 9 exemplifica uma TV-1 (Figura 9a) e uma TV-2 (Figura 9b, c). O valor de α pode ser visto como o poder discriminatório da árvore. À medida que um maior poder discriminatório se faz necessário, novas dimensões são introduzidas.

A estrutura de dados de um *TMBR* é composta por: um vetor telescópico, contendo o número de dimensões (quantidade de coordenadas) e uma lista de coordenadas; e o valor do raio. A árvore gerada é semelhante a da *R-Tree*, e apresenta as seguintes características:

- cada nó contém um conjunto de entradas que correspondem aos *TMBR*s;

- todos os descendentes de uma entrada estão englobados por seu *TMBR*;
- os *TMBR*s podem ser sobrepostos;
- cada nó eqüivale a uma página de disco.

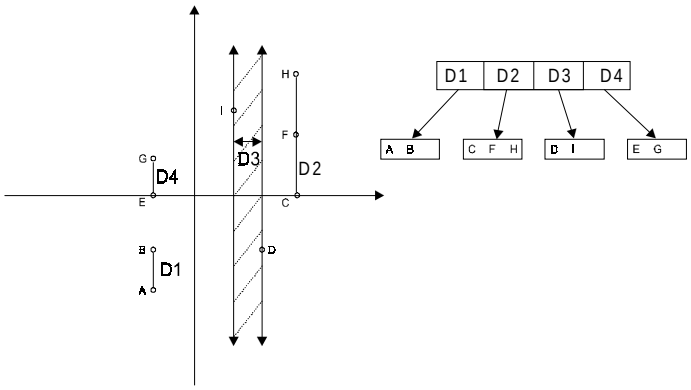
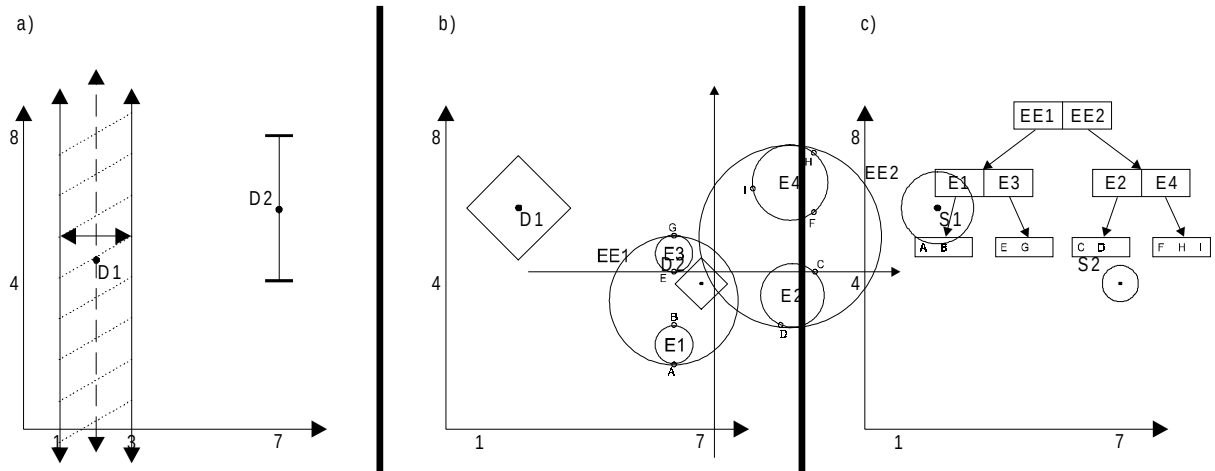


Figura 10: Exemplo de uma **TV-1** (**MBRs** em forma de diamantes).



D1: centro (2), raio 1
D2: centro (7,6), raio 2
Número de dimensões ativas: 1

D1: centro (2,6), raio 2
D2: centro (7,4), raio 1
Número de dimensões ativas: 2

D1: centro (2,6), raio 2
D2: centro (7,4), raio 1
Número de dimensões ativas: 2

Figura 11: Exemplo de uma **TV-2** (**MBRs** em forma de esferas).

• Centro do TMBR
↔ Indica que se extende ao infinito ao longo de tal dimensão

Figura 9: Exemplos de *TMBR*s (em forma de Diamantes e Esferas). a) $\alpha = 1$; b) $\alpha = 2$; c) $\alpha = 2$.

- como o tamanho de uma entrada varia de acordo com a dimensionalidade do vetor telescópico, a quantidade total de entradas que um nó pode ter também varia. Esta quantidade é definida pelo tamanho (em *bytes*) de um nó e pelo tamanho dos objetos (pontos e *TMBR*s) nele inseridos.

A Figura 10 apresenta uma *TV*-1 e a Figura 11 apresenta uma *TV*-2. A faixa A-I corresponde a pontos e as faixas D1-D4, SS1-SS2 e S1-S4 correspondem a nós internos. Apenas as duas primeiras dimensões são mostradas.

4.3 Consulta

A consulta na *TV-Tree* é similar à feita na *R-Tree*. A partir da raiz, cada uma das entradas é examinada e a subárvore referente àquelas cujo *TMBR* faz interseção com a região de busca é percorrida recursivamente. Pode ser necessário percorrer mais do que uma subárvore de um determinado nó, pois os *TMBR*s não são, necessariamente, disjuntos.

4.4 Inserção

A inserção é iniciada por uma busca percorrendo as subárvores mais adequadas a receberem o novo ponto (os critérios para indicar esta adequação são apresentados a seguir). Ao alcançar uma folha, a inserção é feita. Se a quantidade máxima de entradas de um nó foi superada, ou ele é particionado, ou algumas de suas entradas são reinseridas. Após a inserção, os ajustes necessários nos *TMBR*s são propagados árvore acima.

Um *TMBR* pode ser ajustado de duas formas: aumentando o seu raio ou decrementando sua dimensionalidade (contraindo o vetor telescópico do seu centro) de forma a acomodar o novo ponto. A Figura 12 ilustra o ajuste no *TMBR D*1, de uma *TV*-1, para incluir o ponto *P*.

A escolha da subárvore a ser percorrida para incluir um elemento *e* (ponto ou *TMBR*), baseia-se nos seguintes critérios:

- Aquela em que o ajuste no *TMBR* gera a menor quantidade de regiões sobrepostas;
- Aquela cujo *TMBR* necessita de menor ajuste quanto à sua dimensionalidade. Ou seja, se há a necessidade de diminuir a dimensionalidade do *TMBR*, escolha aquele que sofrerá a menor redução;
- Aquela cujo *TMBR* necessita de menor aumento em seu raio;
- Aquela onde a distância entre o ponto a ser inserido e o centro do seu *TMBR* é mínima.

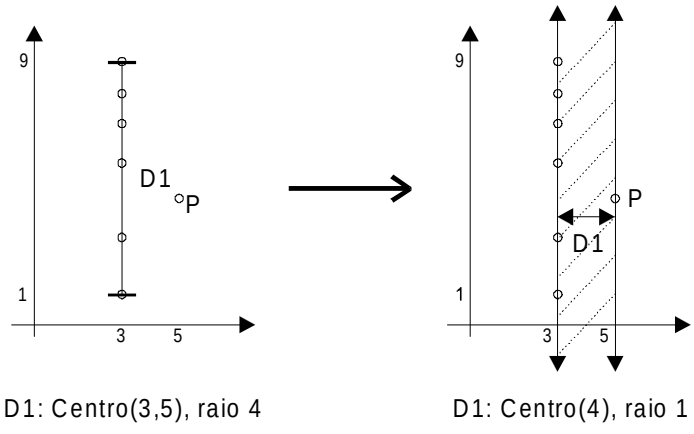


Figura 12: Exemplo de inserção. O TMBR D1 é ajustado para incluir o ponto P.

Durante a inserção, um nó pode ter a

quantidade máxima de entradas ultrapassada em duas situações: por estar recebendo mais uma entrada ou porque alguma das entradas já existentes teve o seu *TMBR* modificado (uma tentativa de estender o vetor telescópico aumentaria o espaço necessário para armazená-lo, por exemplo). A solução mais simples seria particionar o nó. Porém, outra opção apresentada pela *TV-Tree* é reinserir alguns elementos a partir da raiz. A vantagem com esta última opção é que aumenta-se a chance de colocar em um mesmo nó elementos que se encontram mais próximos.

Quando o número de entradas em um nó é superado, os seguintes critérios são utilizados para decidir o que deve ser feito:

- se for uma folha, uma porcentagem (p_{ri}) pré-determinada de suas entradas serão reinseridas, se esta for a primeira vez que o espaço do nó é esgotado durante a inserção corrente. Caso contrário, a folha será particionada. Políticas diferentes podem ser utilizadas para decidir que entradas devem ser reinseridas. Uma opção é escolher a que se encontra mais distante do centro da região;
- se o nó for interno será sempre particionado e o resultado será propagado árvore acima.

Algoritmo 15: Inserção

Inserir o elemento *e* na árvore cuja raiz é indicada por *N*:

1. Use *EscolherEntrada* para determinar a subárvore a ser percorrida, até uma folha ser alcançada. Faça *L* conter o endereço desta folha;
2. Insira *e* em *L*;
3. se o número de entradas de *L* foi superado:
    - 3.1. então, se esta é a primeira vez durante a inserção:
        - 3.1.1. então, obtenha os p_{ri} elementos mais distantes do centro de *L* e reinsira-os desde a raiz;
    - 3.2. senão, particione *L*;
4. Atualize os *TMBR*s necessários. Particione os nós internos que superarem a quantidade máxima de entradas.

4.5 Particionamento

O particionamento da *TV-Tree* pode ser feito de várias formas. Em [LinK_94], duas são destacadas:

- pelo uso de alguma técnica de agrupamento visando manter em um nó os objetos (pontos e *TMBR*s) que se encontram mais próximos entre si, por exemplo: dois objetos, os que apresentam a maior distância entre si, são escolhidos como sementes para encabeçar os dois grupos que serão gerados; a partir daí, os outros objetos são inseridos de forma a manter os mais próximos no mesmo grupo. Ou seja, cada objeto é comparado com as sementes e inserido no grupo encabeçado por aquela de menor distância ao mesmo.
- pelo uso de alguma técnica de ordenação (*space-filling curves* [GaedeV_97] [SametH_90], por exemplo). Os vetores são ordenados e em seguida o nó é particionado segundo algum critério, por exemplo: manter em um mesmo nó os objetos que, juntos, somam o menor raio.

4.6 Remoção

A remoção é semelhante à feita na *R-Tree*. A árvore é percorrida até que a folha onde se encontra o objeto a ser removido seja atingida. Em seguida, o mesmo é removido e as alterações necessárias (ajustes nos *TMBR*s dos nós superiores) são propagadas árvore acima. No caso de um nó ficar com menos do que a quantidade mínima de entradas, o mesmo é removido e suas entradas são reinseridas.

4.7 Conclusões Parciais

A proposta de trabalhar com um subconjunto das coordenadas dos pontos a cada instante acaba diminuindo os problemas de ocupação dos nós (quantidade de entradas por nó) encontrados nas duas outras estruturas aqui apresentadas. Mas, mesmo assim, pode haver alguma queda de desempenho para dimensões mais elevadas.

Com a *R-Tree*, o número máximo de entradas por nó diminui drasticamente à medida em que a dimensão aumenta. Porém, como a *TV-Tree* trabalha com um subconjunto de coordenadas, esta diminuição é amenizada, tornando sua aplicação para altas dimensões mais apropriada. Mas, mesmo assim, este recurso não garante que seja possível armazenar os *TMBR*s dos nós internos quando indexando pontos em altas dimensões, pois pode ocorrer que um *TMBR* venha a ser estendido de tal forma que não seja possível armazená-lo em um único nó. Em outras palavras, esta estrutura apresenta um comportamento em função do aumento da dimensão melhor que o apresentado pela *R-Tree*, mas não resolve de todo os problemas relativos à dimensionalidade do espaço.

Outro problema encontrado é que os *TMBR*s não são disjuntos, o que pode levar a uma queda no desempenho das consultas provocada pela necessidade de visitar uma maior quantidade de nós.

Finalizando, a *TV-Tree* apresenta uma política bem definida de reorganização de dados, o particionamento de um nó interno não precisa ser propagado árvore abaixo e uma taxa de ocupação mínima pode ser mantida.

5. TESTANDO A R-TREE EM VÁRIAS DIMENSÕES

5. TESTANDO A R-TREE EM VÁRIAS DIMENSÕES

Este capítulo apresenta resultados de simulações feitas com o MIE *R-Tree* para dimensões variadas. Inicialmente será descrito como os dados de simulação foram gerados e como foi feita a simulação. Em seguida, serão apresentados os resultados destas simulações e as conclusões desta fase do trabalho.

5.1 Dados Espaciais

O universo de estudo desta simulação constituiu-se de 12.000 vetores de 256 elementos representando 12.000 pontos no espaço de dimensão 256. Para variar as dimensões foram considerados os *k* (sendo *k* a dimensão em teste) primeiros elementos destes vetores. O procedimento utilizado para a obtenção dos vetores é descrito a seguir.

A intensão inicial era testar o funcionamento da *R-Tree* em dimensões variadas indexando vetores de características gerados a partir de imagens médicas. No entanto, um número razoável de vetores seria em torno de 10.000, ou seja, seriam necessárias 10.000 imagens médicas, em meio digital, para gerar estes vetores. Como esta quantidade é muito alta, dificultando a obtenção das imagens para teste, o conjunto de dados foi gerado da seguinte forma:

- o algoritmo que seria utilizado para extrair as características das imagens gerava um vetor contendo o histograma de cores para variações de tons de cinza em uma escala com 8 *bits* de profundidade (256 tons diferentes). Ou seja, para cada tom da escala de cinza, a partir de uma imagem, seriam contados os pontos (*pixels*) com aquele tom. Com isso, pretendia-se criar vetores com 256 elementos.
- para substituir as imagens, foi montado um arquivo contendo dados binários. Na literatura, diversas idéias têm sido aplicadas para gerar dados amostrais: indexar retângulos obtidos no esquema de um circuito integrado de computador [GuttmanA_84], processar palavras de um dicionário extraindo vetores de características [LinK_94], gerar dados aleatoriamente [Berchtold_96], etc. Procurando estar próximo ao que seria feito com as imagens médicas e para que fosse possível gerar vários vetores com 256 elementos, trilhas de músicas de um CD foram copiadas

digitalmente para o disco rígido e concatenadas, gerando um arquivo de tamanho equivalente ao do número de imagens que se desejava indexar.

- pretendia-se utilizar 12.000 imagens do tipo 220 X 220 X 8 (linhas X colunas X profundidade em *bits*), totalizando 48.400 *bytes* por imagem. Foi, então, gerado um arquivo com 580.800.000 *bytes*.
- a partir daí, os vetores foram montados em outro arquivo. Foram armazenados os 256 valores encontrados por vetor e a posição no arquivo de trilhas musicais do primeiro *byte* de cada subconjunto de 48.400 *bytes*. Foi feito, então, um programa que percorria o arquivo gerando (simulando) o histograma de cores a partir de blocos de 48.400 *bytes*.

Os vetores assim formados apresentam uma dependência linear entre seus elementos onde o somatório destes, para qualquer vetor, é igual a 48.400. Com isto, os pontos representados pelos vetores estarão distribuídos de maneira não uniforme, ao longo de uma "hiper-superfície" discreta, no espaço. Observa-se, ainda, que os dados gerados podem ser coincidentes. Ou seja, não é garantida a unicidade destes vetores (podem existir vários vetores com os mesmos elementos, indicando o mesmo ponto no espaço).

5.2 Material

Para a simulação foi utilizado o seguinte sistema computacional: microcomputador Pentiun II 300 MHz com 256 MB de memória principal, 3 GB de disco rígido e sistema operacional Linux Kernel 2.0.35 distribuição Red Hat.

Os nós das *R-Tree*s usadas na simulação tinham 4.096 *bytes*. Foram feitos testes com os três algoritmos de particionamento descritos na seção 3.6. A quantidade máxima de entradas por nó foi determinada pela dimensão em teste, sendo a quantidade mínima definida como metade da máxima. A Tabela 1 apresenta a relação de quantidades máxima e mínima de entradas por nó versus a dimensão.

O código da *R-Tree* foi obtido em [GuttmanA_97] e adaptado para atender as necessidades geradas por esta simulação (tamanho do nó, variação na dimensão, seqüência de teste, etc.). Este código gera a árvore totalmente residente em memória principal.

Tabela 1: Quantidades máxima e mínima de entradas por nó da *R-Tree* em função da dimensão em teste.

Dimensão	2	4	8	16	32	64	128	255
Quantidade Máxima de Entradas	254	127	63	31	15	7	3	2
Quantidade Mínima de Entradas	102	56	30	15	7	3	1	1

O nó da *R-Tree* é composto pelos seguintes dados, com seus respectivos tipos (em Linux, os tipos inteiro e ponteiro ocupam 4 *bytes*):

- nível da árvore, tipo inteiro;
- quantidade de entradas, tipo inteiro;
- conjunto de entradas. Cada entrada é composta por:
    - endereço do filho, para nós internos. Ou, endereço do objeto no arquivo de dados, referente à entrada. Ambos com tipo ponteiro;
    - vetor de intervalos. Cada intervalo, um por dimensão, possui os limitantes inferior e superior do *MBR* na dimensão correspondente. O tipo de cada limitante é inteiro.

5.3 Método

O método utilizado nesta simulação foi baseado no que Guttman apresentou em [GuttmanA_84].

Cada simulação foi composta por 3 etapas:

1. Inserção: 10.000 vetores foram inseridos, um a um, na mesma ordem em que foram gerados. Para efeito de análise de desempenho, apenas os 1.000 últimos foram contabilizados;
2. Consulta: foram feitas 1.000 consultas por coincidência exata (*exact match query*). Considerando o conjunto dos 10.000 registros já inseridos, foi feito um programa que escolhia, aleatoriamente, 1.000 registros deste conjunto montando uma seqüência de

busca. O programa foi feito de forma a garantir que o mesmo registro só seria consultado uma vez.

3. Remoção: foram feitas 1.000 remoções. Considerando o conjunto dos 10.000 registros já inseridos, o mesmo programa utilizado na etapa anterior foi usado para selecionar os registros a serem removidos.

Foram realizadas um total de 21 simulações combinando as diferentes dimensões com os 3 algoritmos de particionamento descritos na seção 3.6. Destas, duas apresentaram problemas de segmentação de memória e foram descartadas dos resultados, a saber: algoritmo Quadrático aplicado às dimensões 128 e 255. Não foram feitas simulações com o algoritmo Exaustivo nas dimensões 2, 4 e 8, pois o tempo de execução seria muito grande (ver seção 3.6.1, sobre quantidade de combinações possíveis). A Tabela 2 mostra com um X as combinações que foram avaliadas, perfazendo um total de 21 simulações.

Tabela 2: Quantidade de entradas por nó da R-Tree em função da dimensão em teste.

	Dimensão								
Particionamento	***2***	***4***	***8***	***16***	***32***	***64***	***128***	***255***	***Total***
Exaustivo				X	X	X	X	X	5
Quadrático	X	X	X	X	X	X			6
Linear	X	X	X	X	X	X	X	X	8
Total	2	2	2	3	3	3	2	2	19

Finalizando, a última dimensão testada foi 255. A dimensão 256, que resultaria da seqüência de potências de 2, somente permitiria armazenar um *MBR* com 256 intervalos em nós com 4096 *bytes*. Com 255 intervalos, foi possível armazenar 2 *MBR*s.

5.4 Resultados e Discussão

Na presente seção serão apresentados e discutidos, sucessivamente, os resultados obtidos na inserção, consulta e remoção.

Inserção

A Tabela 3 apresenta a configuração das árvores após a fase de Inserção. Como pode ser observado, a altura da árvore (item I) cresce com o aumento da dimensão. Isto ocorre devido à diminuição da quantidade máxima de entradas por nó. O mesmo comportamento é apresentado pela quantidade total de nós e pelo tamanho do índice (itens II e III, respectivamente).

Tabela 3: Configuração da árvore após a inserção de 10.000 pontos.

Dimensão	2	4	8	16	32	64	128	255
I. Altura da Árvore								
i. Exaustivo				3	4	6	9	14
ii. Quadrático	2	3	3	3	4	8		
iii. Linear	2	3	3	3	4	7	13	17
II. Quantidade Total de Nós								
i. Exaustivo				469	1.038	2.499	5.005	10.005
ii. Quadrático	71	139	238	467	1.025	4.131		
iii. Linear	77	140	239	470	1.010	3.320	9.995	14.328
III. Tamanho do Índice (kB)								
i. Exaustivo				1.876	4.152	9.996	20.020	40.020
ii. Quadrático	284	556	952	1.868	4.100	16.524		
iii. Linear	308	560	956	1.880	4.040	13.280	39.980	57.312
IV. Taxa de Ocupação								
i. Exaustivo				74%	70%	71%	99%	99%
ii. Quadrático	69%	64%	71%	74%	71%	48%		
iii. Linear	64%	64%	71%	74%	72%	57%	66%	84%

É fácil entender que, diante de uma capacidade menor de entradas por nó, um número maior de nós seja necessário para armazenar a mesma quantidade de pontos. Isto aumenta a altura da árvore e o tamanho do índice.

Como os itens I, II e III da Tabela 3 mostram, em função da variação da dimensão, o mesmo comportamento, somente o gráfico de um deles (item I) é apresentado (Figura 13). Ao observar este gráfico, torna-se evidente uma diferenciação no desempenho dos algoritmos por volta da dimensão 64.

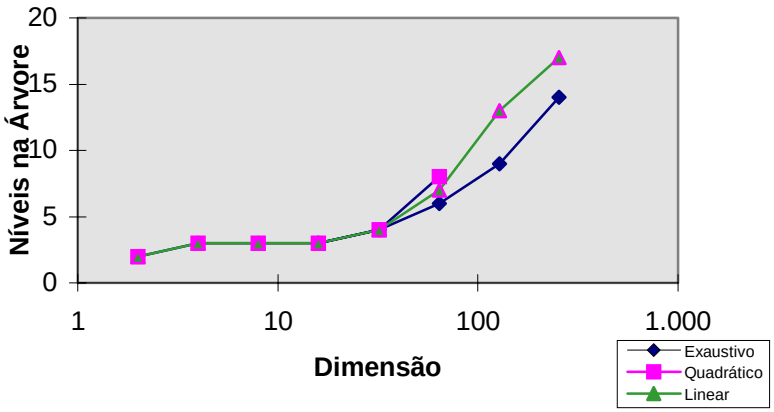


Figura 13: Altura da árvore após 10.000 inserções.

Sabe-se que os pontos utilizados nas simulações não estão uniformemente distribuídos no espaço, pois existe uma dependência linear entre as coordenadas. A cada dimensão apenas uma parte destas coordenadas é utilizada. Aparentemente, as coordenadas relativas às dimensões acima de 32 afetam esta distribuição, afetando de alguma forma o comportamento (desempenho) da árvore.

A taxa de ocupação se manteve entre 64% e 84% nas dimensões abaixo de 32, inclusive. A faixa obtida se encontra dentro do esperado para árvores semelhantes à *B-Tree*. Os algoritmos Quadrático e Linear apresentam uma queda brusca nos valores da taxa de ocupação entre as dimensões 32 e 64 (de 71% para 48% e de 72% para 57%, respectivamente). Novamente, observa-se que a distribuição espacial dos pontos possa ter influenciado estes resultados.

Outro resultado a ser observado é que o valor encontrado na dimensão 64 para a taxa obtida com o algoritmo Quadrático é inferior á ocupação mínima por nó (50%). Isso pode ser explicado da seguinte forma: como não são feitas restrições quanto à ocupação mínima da raiz, este valor indica que todas folhas e nós internos apresentam valores próximos a 50% de ocupação, ao passo que a raiz apresenta um valor inferior a este.

As taxas de ocupação encontradas para as dimensões 128 e 255 são, respectivamente: 66% e 84% para o algoritmo Linear e 99%, nos dois casos, para o algoritmo Exaustivo. Como o número mínimo e máximo nestas dimensões são bastante próximos, a árvore tende a apresentar uma taxa de ocupação mais próxima dos 100%.

Os resultados da Tabela 4 mostram o comportamento da *R-Tree* durante a inserção dos últimos 1.000 pontos, dentre os 10.000 inseridos. O número de nós particionados por ponto inserido apresenta um comportamento semelhante ao dos itens I, II e III da Tabela 3, discutidos anteriormente. Este número aumenta gradualmente até a dimensão 32, inclusive.

A dimensão 64 da Tabela 4 mostra um aumento considerável no item I (número de nós particionados por inserção) para os três algoritmos: de 0,11 para 0,25 com o Exaustivo; de 0,10 para 0,50 com o Quadrático; e de 0,09 para 0,34 com o Linear. Este salto pode confirmar a suspeita da heterogeneidade na distribuição dos pontos. A diminuição da quantidade de entradas

por nó, por si só, não explica o aumento brusco do número de particionamentos, principalmente por ser este aumento comum a todos os algoritmos.

Ora, supondo uma maior concentração de pontos em determinada região era de esperar que estes

Tabela 4: Desempenho da inserção. Foram considerados apenas os últimos 1.000 pontos.

Dimensão	2	4	8	16	32	64	128	255
I. Nós Particionados/Inserção								
i. Exaustivo				0,05	0,11	0,25	0,50	1,00
ii. Quadrático	0,01	0,02	0,03	0,04	0,10	0,50		
iii. Linear	0,01	0,01	0,03	0,06	0,09	0,34	1,00	1,41
II. Nós Acessados/Inserção								
i. Exaustivo				3,05	4,11	6,25	9,50	15,00
ii. Quadrático	2,01	3,02	3,03	3,04	4,10	8,50		
iii. Linear	2,01	3,02	3,02	3,06	4,09	7,33	14,00	18,41
III. Tempo Total de Inserção (s)								
i. Exaustivo				0,33	0,50	0,77	1,85	4,18
ii. Quadrático	0,65	0,62	0,65	0,63	0,80	2,02		
iii. Linear	0,08	0,22	0,23	0,32	0,50	1,12	2,78	5,58

apresentassem uma tendência a ocupar o mesmo nó da *R-Tree*, pois, assim, a área obtida seria menor. Conseqüentemente, este nó tenderia a ser particionado um número maior de vezes, elevando a quantidade de particionamentos por ponto inserido.

O número de nós acessados por inserção (item II da Tabela 4) está próximo à altura da árvore (item I da Tabela 3), como era esperado. O tempo total de inserção (item III da Tabela 4) apresenta um comportamento ao longo da variação da dimensão semelhante ao dos itens I, II e III da Tabela 3.

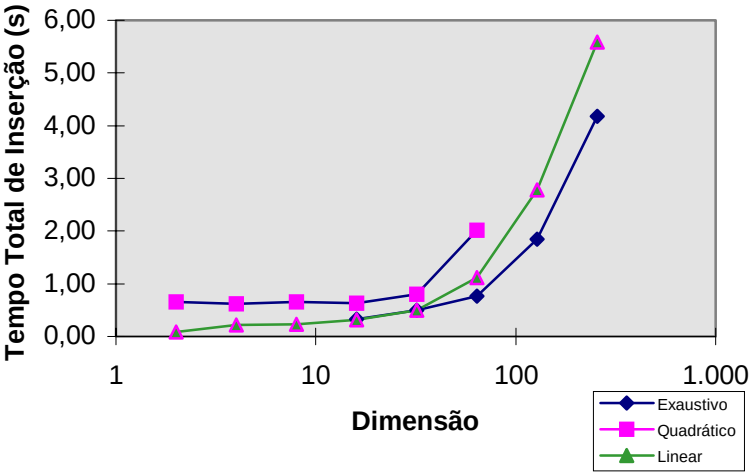


Figura 14: Tempo total de inserção para 1.000 pontos.

O gráfico da Figura 14 apresenta a evolução do tempo total de inserção ao longo da dimensão. A partir deste gráfico conclui-se que o algoritmo Exaustivo apresenta melhor desempenho para dimensões acima de 32, considerando o conjunto de pontos em uso. Entre as dimensões 16 e

32, os algoritmos Exaustivo e Linear apresentam o mesmo desempenho. Abaixo de 16, o Linear apresenta um desempenho melhor.

Consulta

Os resultados da Tabela 5 mostram o comportamento da *R-Tree* durante 1.000 consultas do tipo coincidência exata. Foram encontrados 1.018 pontos para todas as simulações com o algoritmo

Tabela 5: Desempenho obtido com 1.000 consultas por coincidência exata.

Dimensão	2	4	8	16	32	64	128	255
I. Pontos Encontrados								
i. Exaustivo				1.018	1.018	1.018	1.018	1.018
ii. Quadrático	1.025	1.018	1.018	1.018	990	1.013		
iii. Linear	1.025	1.018	1.018	1.018	1.018	1.018	1.018	1.018
II. Nós Acessados/Consulta								
i. Exaustivo				225,67	157,59	43,36	43,60	49,27
ii. Quadrático	3,58	129,12	187,10	225,33	159,50	84,57		
iii. Linear	8,22	130,16	188,03	224,97	161,48	74,26	98,77	91,78
III. Tempo Total de Consulta (s)								
i. Exaustivo				4,60	2,07	0,40	0,60	1,17
ii. Quadrático	0,10	5,47	7,45	4,73	2,05	0,75		
iii. Linear	0,38	5,53	7,53	4,75	2,15	0,78	1,37	2,33
IV. Tempo Total/Nós Acessados/ Consulta								
i. Exaustivo				0,020	0,013	0,009	0,014	0,024
ii. Quadrático	0,028	0,042	0,040	0,021	0,013	0,009		
iii. Linear	0,047	0,043	0,040	0,021	0,013	0,011	0,014	0,025

Exaustivo; com o algoritmo Quadrático foram encontrados 1.018 pontos para as simulações nas dimensões 4, 8 e 16 e, respectivamente, 1.025, 990 e 1.013 nas dimensões 2, 32 e 64; e para o algoritmo Linear, 1.025 pontos para a simulação na dimensão 2 e 1.018 nas demais dimensões.

Destes resultados conclui-se que, dos vetores gerados como descrito na seção 5.1, dentre os 1.000 selecionados para consulta por coincidência exata, 5% são coincidentes na dimensão 2 e apenas 3,6% nas demais dimensões, apresentando uma baixa taxa de colisões apesar da dependência linear entre as coordenadas.

Até o término deste trabalho, não se descobriu o porquê dos resultados apresentados nas dimensões 32 e 64 do algoritmo Quadrático. Era esperado o mesmo resultado obtido com os outros algoritmos. A conclusão imediata seria que este algoritmo gera os chamados falsos

descartes (*false dismissals*), mas serão necessários mais alguns testes e análises para confirmar esta suposição. Fica aqui, apenas a suspeita de haver problemas com este algoritmo.

O item II da Tabela 5 apresenta o número de nós acessados por consulta. De forma geral, considerando que se trata de busca de pontos por coincidência exata era esperado que os resultados ficassem próximos à altura da árvore, com uma pequena variação devido à ocorrência de *MBR*s sobrepostos. Porém, os resultados obtidos mostram um desempenho muito inferior ao esperado, pois em 10 dos 19 casos são acessados um número de nós em torno de 35 vezes a altura da árvore e no restante uma média de 5 vezes.

Este resultado indica que existe um número realmente grande de *MBR*s sobrepostos em nós internos da árvore, apesar do baixo índice de colisões. Porém, ao contrário do que se imaginava, não foi identificada nenhuma dependência destes resultados com a variação da dimensão. Em outras palavras, para este conjunto de dados, não pode ser dito que, com o aumento da dimensão o número de *MBR*s sobrepostos aumenta, levando a uma diminuição no desempenho de uma consulta.

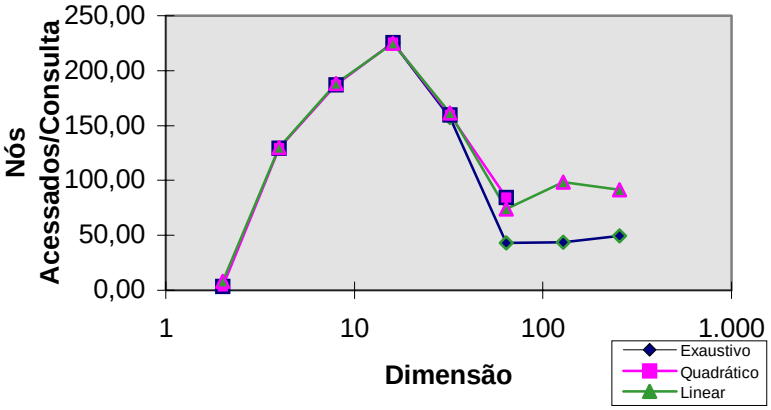


Figura 15: Nós Acessados/Consulta vs. Dimensão.

Observa-se, ainda, que há uma diferenciação nos resultados da dimensão 64 com relação ao número de nós acessados por consulta (Figura 15). Acredita-se que a distribuição espacial dos dados esteja afetando, também, este resultado. Como ocorreu nesta dimensão um número maior de particionamentos, é provável que os *MBR*s resultantes se encontrem menos sobrepostos.

O tempo total de consulta reflete, em parte, o comportamento do número de nós acessados por consulta. No entanto, uma observação deve ser feita: o número de nós acessados por consulta aumenta entre as dimensões 8 e 16 (de 187,1 para 225,33 com o algoritmo Quadrático e de 188,03 para 161,48 para o algoritmo Linear), enquanto que o tempo total de consulta diminui

neste mesmo intervalo (de 7,45 para 4,73 com o algoritmo Quadrático e de 7,53 para 4,75 para o algoritmo Linear), “quebrando” a tendência esperada de que estes valores tivessem a mesma evolução.

Para entender este comportamento, foi calculado o tempo médio gasto em cada nó consultado (tempo total / nós acessados / consulta) e os resultados se encontram no item IV da Tabela 5. Através destes resultados, conclui–se que o tempo médio de processamento dos nós durante uma consulta decresce à medida que a dimensão aumenta. Isto ocorre porque a quantidade de entradas por nó diminui com o aumento da dimensão.

Deve, então, existir um valor ótimo para a quantidade de entradas por nó entre as dimensões 8 e 32, pois é neste intervalo que o decréscimo do tempo médio de processamento dos nós é mais acentuado.

Remoção

A Tabela 6 apresenta os resultados da remoção de 2.000 pontos. Como os pontos a serem removidos foram escolhidos dentre os inseridos, o resultado correto seria 2.000 remoções para todas as simulações feitas. No entanto, para algumas simulações com o algoritmo Quadrático, este número não foi obtido, ou seja, foram removidos: 1.999 pontos na dimensão 16; 1.940 pontos na dimensão 32 e 1.987 pontos na dimensão 64. Novamente não foi possível identificar o que está causando estes resultados.

A quantidade de nós reinseridos por ponto removido (item III da Tabela 6) decresce com o aumento da dimensão, pois diminui a quantidade mínima de entradas por nó. Esta quantidade chega a zero para as dimensões 128 e 255 onde a quantidade mínima de entradas é 1, levando a uma quantidade maior de nós eliminados por ponto removido (item II da Tabela 6).

Ainda sobre a quantidade de nós reinseridos por ponto removido, observa-se que o algoritmo Quadrático apresenta um valor bastante alto para a dimensão 64. Isto se justifica pela baixa taxa de ocupação (48%, item IV da Tabela 3) gerada por este algoritmo. Como conseqüência há, também, uma elevação do número de nós eliminados por ponto removido.

Os itens IV e V da Tabela 6, número de nós acessados por remoção (Figura 16) e tempo total de remoção, respectivamente, apresentam comportamentos semelhantes aos encontrados na etapa de consulta, valendo aqui os mesmos comentários feitos quanto ao número de *MBR*s sobrepostos e à distribuição dos pontos no espaço. Afinal, a remoção envolve uma fase de consulta.

Tabela 6: Desempenho da remoção de 2.000 pontos.

Dimensão	2	4	8	16	32	64	128	255
I. Pontos Removidos								
i. Exaustivo				2.000	2.000	2.000	2.000	2.000
ii. Quadrático	2.000	2.000	2.000	1.999	1.940	1.987		
iii. Linear	2.000	2.000	2.000	2.000	2.000	2.000	2.000	2.000
II. Nós Eliminados/Remoção								
i. Exaustivo				0,05	0,10	0,05	0,01	0,10
ii. Quadrático	0,02	0,05	0,03	0,05	0,10	3,45		
iii. Linear	0,02	0,04	0,03	0,05	0,08	0,28	0,10	0,35
III. Nós Reinseridos/Remoção								
i. Exaustivo				0,68	0,62	0,11	0,00	0,00
ii. Quadrático	1,57	2,48	0,71	0,66	0,59	6,89		
iii. Linear	1,57	2,06	0,86	0,67	0,50	0,57	0,00	0,00
IV. Nós Acessados/Remoção								
i. Exaustivo				106,85	73,78	24,94	27,30	30,23
ii. Quadrático	5,87	70,18	91,69	106,87	83,75	99,77		
iii. Linear	9,05	67,57	91,62	107,06	80,23	43,95	54,71	47,14
V. Tempo Total de Remoção (s)								
i. Exaustivo				2,45	2,50	1,68	2,52	4,92
ii. Quadrático	1,90	5,32	3,27	2,62	2,78	22,30		
iii. Linear	0,78	2,92	3,13	2,38	2,43	2,92	3,08	6,08

Comparando-se, ainda, a remoção e a consulta, a remoção apresenta uma quantidade de nós acessados por remoção e um tempo total de processamento menores do que aqueles apresentados pela consulta. Isto pode ser visto como conseqüência da redução do número de pontos existentes na árvore. Curiosamente, apesar de serem removidos 2.000 pontos, contra 1.000 consultas (metade dos pontos), o tempo total de remoção é menor do que o de consulta.

Outra diferença se dá nos resultados obtidos para a dimensão 64 do algoritmo Quadrático onde ocorre um aumento dos acessos e do tempo total. Isto se justifica, novamente, pela baixa taxa de ocupação gerada por este algoritmo.

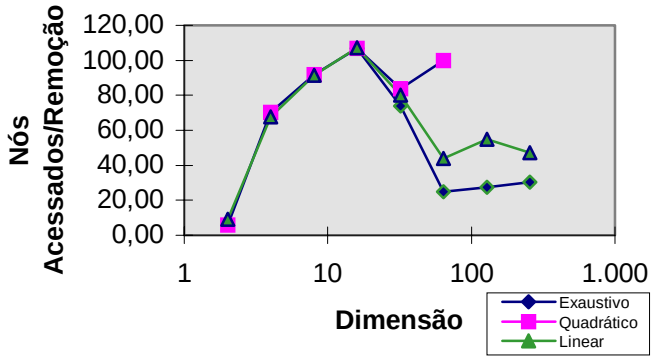


Figura 16: Nós Acessados/Remoção vs. Dimensão.

Por fim, em relação à remoção o algoritmo Exaustivo apresenta melhor desempenho para dimensões acima de 32, praticamente empatando com o algoritmo Linear entre as dimensões 16 e 32. Abaixo de 16, o Linear apresenta melhor desempenho.

5.5 Conclusões Parciais

O mais significativo dentre os resultados obtidos, considerando a aplicação da *R-Tree* para indexar pontos no espaço, é o número de nós acessados por consulta. Como já foi dito, esperava-se que este número fosse proporcional à altura da árvore. Porém, ao contrário do que se imaginava, aparentemente a variação da dimensão não afeta o desempenho desta estrutura, mas sim a distribuição dos pontos no espaço.

Os resultados obtidos na consulta e na remoção com o algoritmo Quadrático em relação aos falsos descartes precisam ainda ser checados. No entanto, por este motivo e pelos resultados obtidos, este algoritmo não é aconselhado para se trabalhar com dimensões variadas.

Fica como proposta para trabalhos futuros a elaboração desta simulação com outros conjuntos de vetores. No mínimo, um conjunto que não apresente qualquer dependência entre os elementos de seus vetores e outro gerado aleatoriamente. Não se deve deixar de lado, é claro, um conjunto gerado a partir de imagens reais. Para este último conjunto, vale a pena, também, efetuar consultas por regiões geradas a partir de pontos conhecidos e comparar as imagens obtidas em busca de semelhanças.

Por fim, tendo em vista os resultados apresentados pelos algoritmos Linear e Exaustivo, como a aplicação em questão necessita uma estrutura que se ajuste à variação da dimensão, uma outra proposta seria implementar uma *R-Tree* que trabalhe com os dois algoritmos. Para as aplicações com dimensões inferiores a um certo valor, a depender do tamanho do nó em uso, o algoritmo Linear seria utilizado. Para as demais, o Exaustivo seria utilizado.

6. CONCLUSÃO GERAL

6. CONCLUSÃO GERAL

6.1 Resumo e Considerações Finais

Este trabalho apresentou a descrição de três métodos de indexação espacial: *K-D-B-Tree*, *R-Tree* e *TV-Tree*. Foi feita para cada método uma análise crítica de seus algoritmos e da estrutura de dados, visando sua utilização para indexar dados estritamente pontuais em aplicações onde a dimensão do espaço não é conhecida a priori, podendo atingir valores realmente altos. Foram apresentados, ainda, os resultados de simulações feitas com o método *R-Tree*.

Ao contrário do que se imaginava antes da elaboração deste trabalho, o método *K-D-B-Tree* apresenta uma degradação com o aumento da dimensão, pois o número de entradas por nó diminui, elevando a altura da árvore. Além disso, ao contrário da *K-D-Tree*, o número de comparações feitas para identificar a subárvore a ser percorrida cresce em função da dimensão. Outro problema é que pode ser necessário propagar o particionamento de nós internos, árvore abaixo, o que viria a gerar taxas de ocupação indesejadas.

Os resultados obtidos através das simulações com a *R-Tree* indicam que a distribuição espacial dos dados utilizados afeta o desempenho mais do que a dimensão em uso. Em todo caso, para aplicações que envolvem estritamente pontos este método não é aconselhável, pois não apresenta um bom desempenho em consultas, mesmo para dimensões baixas. Um grande número de nós é acessado devido à sobreposição dos *MBR*s. As simulações discutidas aqui apresentam resultados em que para se encontrar um único ponto (consulta por coincidência exata) são acessados 120 nós, em média, para árvores com altura máxima 10.

A *TV-Tree* apresenta uma proposta de contrair e estender o vetor de coordenadas dos pontos indexados. A indexação é feita sobre apenas algumas coordenadas, adicionando outras quando o poder discriminatório que possuem for realmente necessário. Com isso, é obtido um número maior de entradas por nó em níveis mais próximos à raiz, melhorando o seu desempenho. Como este método já trabalha com tamanhos variados de vetores de características, torna-se bastante indicado para a aplicação em questão.

Os gerenciadores de base de dados, em geral, utilizam um tamanho fixo de registro de dados. Como os MIEs *K-D-B-Tree* e *R-Tree* aumentam o espaço utilizado pelas entradas nos nós linearmente em função da dimensão, seriam necessários ajustes extras tanto nos gerenciadores quanto nos MIEs para suportar aplicações em dimensões muito altas.

Dentre os três métodos de indexação espacial ora apresentados, o *TV-Tree* atende melhor às especificações propostas (utilização em altas dimensões, aplicação do mesmo método a dimensões diferentes e exigência do mínimo de alterações nos gerenciadores de dados).

6.2 Proposta de Trabalhos Futuros

Os testes realizados com a *R-Tree*, além de proporcionar um melhor entendimento do seu funcionamento, geraram um conjunto de valores para comparação e um conjunto de dados para simulação. De posse destes dados e valores, seria interessante a execução do mesmo processo de simulação para as estruturas *K-D-B-Tree* e *TV-Tree*. Com os resultados que vierem a ser obtidos será possível fazer uma análise do desempenho destas estruturas, baseada em resultados empíricos.

É interessante trabalhar no desenvolvimento de uma estrutura híbrida entre a *K-D-B-Tree* e a *TV-Tree*. A *K-D-B-Tree* gera regiões disjuntas do espaço, o que deve melhorar o desempenho da consultas (desconsiderando os problemas gerados pelo particionamento e falta de uma política de reorganização). A *TV-Tree* resolve o problema relacionado com o tamanho de um nó, contraindo e estendendo os vetores de coordenadas dos pontos. Pode-se depreender dessas considerações que pode ser bastante interessante uma solução mesclando estas duas características.

Finalizando, fica como proposta estudar formas de mensurar a distribuição espacial dos dados, com o intuito de gerar subsídios para confirmar as suspeitas em torno do desempenho da *R-Tree*.

Referências Bibliográficas

REFERÊNCIAS BIBLIOGRÁFICAS

[AlcocerP_96] Alcocer, P.R.C.; Melo, C.P.; Furuie, S.S.; Bertozzo Jr., N.; Parzianello, L.C.; Rebelo, M.: "*O Projeto PACS: Um Sistema para Visualisação Dinâmica e Armazenamento de Imagens de Angiografia Digital no INCOR*", in Anais do 3º Fórum Nacional de Ciência e Técnologia em Saúde, Campos do Jordão - SP, Nro.2, pp. 695-696, Outubro 1996.

[BeckmannN_90] Beckmann, N.; Kriegel, H.-P.; Schneider, R.; Seeger, B.: "*R*-tree: An Efficient and Robust Access Method for Points and Rectangles*", Proceedings of the ACM SIGMOD International Conference on the Management of Data, pg. 322-331, Maio 1990.

[Berchtold_96] Berchtold, S.; Keim, D.A.; Kriegel, H.-P.: "*The X-tree: An Index Structure for High-Dimensional Data*", Proceedings of the 23rd VLDB Conference, Mumbai (Bombay), India, 1996.

[ComerD_79] Comer, D.: "*The ubiquitous B-tree*", ACM Computing Surveys, Vol. 11, Nro. 2, pp. 121-137, Junho 1979.

[DuttonG_90] Dutton, G.F.: "*The Current State of PACS*", Applied Radiology, pp 15-19, Agosto 1990.

[GaedeV_97] Gaede, V.; Günther, O.: "*Multidimensional Acess Methods*", a ser publicado em ACM Computing Surveys - http://www.wiwi.hu-berlin.de/~gaede/survey.ver.ps.Z, 1997.

[GuttmanA_84] Guttman, A.: "*R-trees: A Dynamic Index Structure For Spacial Searching*", Proceedings of the ACM SIGMOD International Conference on the Management of Data, pp. 47-57, Junho 1984.

[GuttmanA_97] Guttman, A.: Implementação da estrutura de índices *R-Tree*. http://emerald.ucsc.edu/~tonig/rtrees/scr.tar, Janeiro 1997.

[LinK_94] Lin, K.-I.; Jagadish, H.V.; Faloutsos, C.: "*The TV-tree - an index structure for high-dimensional data*", VLDB Journal, Vol. 3, pp. 517-542, Outubro 1994.

[PetrakisE_97] Petrakis, E.G.M.; Faloutsos, C.: "*Similarity Searching in Medical Image Databases*", IEEE Trans. On Knowledge and Data Engineering, Vol. 9, Nro. 3, pp. 435-447, Maio/Junho 1997.

[RobinsonJ_81] Robinson, J.T.: "*The K-D-B Tree: A Search Structure for Large Multidimensional Dynamic Indexes*", Proceedings of the ACM SIGMOD International Conference on the Management of Data, pg. 10-18, Abril 1981.

[SametH_90] Samet, H.: "*The Design and Analysis of Spatial Data Structures*", Addison-Wesley, Reading, ISBN 0-201-50255-0, MA, 1990.

[SantosFilhoR_99] Santos Filho, R.F.; Traina, A.J.M.; Traina Jr., C.- *"Algoritmos Para Indexação em Bases de Dados Através de Estruturas Multidimensionais"*, São Carlos, ICMC-USP, Abril 1999 (Relatório Técnico nº. 85). [Disponível na Internet em <http://www.icmc.sc.usp.br/~biblio>]

[SclabassiR_96] Sclabassi, R.J.; Krieger, D.; Simon, R.; Lofink, R.; Gross, G.; DelLauder, D.M.: "*NeuroNet: Collaborative Itraoperative Guidance and Control*", IEEE Computer Graphics and Applications, (16) Nro. 1, pp. 39-45, 1996.

[TamuraH_84] Tamura, H.; Yokoya, N.: "*Image Database Systems: A Survey*", Pattern Recognition, Vol. 17, Nro. 1, pp. 29-49, 1984.

[TrainaC_97] Traina Jr., C.; Traina, A.J.M.; Santos, R. R.; Senzako, E. Y.: "*Content-Based Medical Image Retrieval in Object Oriented Databases*", in Journal of Informatica Medica Slovenica, Vol. 3, Nro. 12, pp. 67-72, Junho 1997.

[WhiteD_95] White, D.A.; Jain, R.: "*Similarity Indexing with the SS-tree*", IEEE 1995.

Apêndice

Apêndice

Particionamento Exaustivo

Tabela 7: Etapa: Inserção.

Dimensão	64	128	255
Nós particionados	250	500	999
Particionamentos/inserção	0,25	0,50	1,00
Nós visitados/ inserção	6,25	9,50	15,00
Tempo de UCP (s)	0,767	1,850	4,183
Tempo de UCP/inserção (ms)	0,77	1,85	4,18
Tempo de UCP/particion.(ms)	3,07	3,70	4,19

Tabela 8: Configuração da árvore após a etapa Inserção.

Dimensão	64	128	255
Níveis na árvore	6	9	14
Nós na árvore	2.499	5.005	10.005
Folhas	2.000	3.334	5.000
Nós internos	499	1.671	5.005
Nós/ponto	0,25	0,50	1,00
Tamanho total do índice (KB)	9.996	20.020	40.020
Kbytes/ponto	1,00	2,00	4,00
Total de entradas	12.498	15.004	20.004
Espaço total para entradas	17.493	15.015	20.010
Taxa de ocupação	71%	99%	99%

Tabela 9: Etapa: Consulta.

Dimensão	64	128	255
Pontos Encontrados	1.081	1.081	1.081
Páginas acessadas/consulta	43,36	43,60	49,27
Páginas acessadas/acerto	42,60	42,83	48,40
Tempo de UCP (s)	0,400	0,600	1,167
Tempo de UCP/consulta (ms)	0,40	0,60	1,17
Tempo de UCP/acerto (ms)	0,39	0,59	1,15

Tabela 10: Etapa: Remoção.

Dimensão	64	128	255
Nós eliminados	109	22	198
Nós eliminados/remoção	0,05	0,01	0,10
Nós reinseridos	218	0	0
Reinserções/nó eliminado	2	0	0
Reinserções/remoção	0	0	0
Páginas acessadas/remoção	24,94	27,30	30,23
Tempo de UCP (s)	1,683	2,517	4,917
Tempo de UCP/remoção (ms)	0,84	1,26	2,46

Particionamento Quadrático

Tabela 11: Etapa: Inserção I.

Dimensão	2	4	8	16	32	64
Nós particionados	10	16	29	43	103	496
Particionamentos/ inserção	0,01	0,02	0,03	0,04	0,10	0,50
Nós visitados/ inserção	2,01	3,02	3,03	3,04	4,10	8,50
Tempo de UCP (s)	0,650	0,617	0,650	0,633	0,800	2,017
Tempo de UCP/inserção (ms)	0,65	0,62	0,65	0,63	0,80	2,02
Tempo de UCP/particion.(ms)	65,00	38,54	22,41	14,73	7,77	4,07

Tabela 12: Configuração da árvore após a etapa Inserção I.

Dimensão	2	4	8	16	32	64
Níveis na árvore	2	3	3	3	4	8
Nós na árvore	71	139	238	467	1.025	4.131
Folhas	70	136	231	445	928	2.791
Nós internos	1	3	7	22	97	1.340
Nós/ponto	0,01	0,01	0,02	0,05	0,10	0,41
Tamanho total do índice (KB)	284	556	952	1.868	4.100	16.524
Kbytes/ponto	0,03	0,06	0,10	0,19	0,41	1,65
Total de entradas	10.070	10.138	10.237	10.466	11.024	14.130
Espaço total para entradas	14.484	15.707	14.280	14.010	15.375	28.917
Taxa de ocupação	69%	64%	71%	74%	71%	48%

Tabela 13: Etapa: Consulta.

Dimensão	2	4	8	16	32	64
Pontos Encontrados	1.025	1.018	1.018	1.018	990	1.013
Páginas acessadas/consulta	3,58	129,12	187,10	225,33	159,50	84,57
Páginas acessadas/acerto	3,49	126,83	183,79	221,35	161,11	83,48
Tempo de UCP (s)	0,100	5,467	7,450	4,733	2,050	0,750
Tempo de UCP/consulta (ms)	0,10	5,47	7,45	4,73	2,05	0,75
Tempo de UCP/acerto (ms)	0,10	5,37	7,32	4,65	2,07	0,74

Tabela 14: Etapa: Remoção.

Dimensão	2	4	8	16	32	64
Nós eliminados	31	90	49	94	190	6.849
Nós eliminados/remoção	0,02	0,05	0,03	0,05	0,10	3,45
Nós reinseridos	3.131	4.950	1.421	1.316	1.137	13.692
Reinserções/nó eliminado	101	55	29	14	6	2
Reinserções/remoção	2	2	1	1	1	7
Páginas acessadas/remoção	5,87	70,18	91,69	106,87	83,75	99,77
Tempo de UCP (s)	1,900	5,317	3,267	2,617	2,783	22,300
Tempo de UCP/remoção (ms)	0,95	2,66	1,63	1,31	1,44	11,22

Particionamento Linear

Tabela 15: Etapa: Inserção I.

Dimensão	2	4	8	16	32	64	128	255
Nós particionados	12	15	25	58	90	335	1.000	1.413
Particionamentos/inserção	0,01	0,01	0,03	0,06	0,09	0,34	1	1,41
Nós visitados/inserção	2,01	3,02	3,02	3,06	4,09	7,33	14	18,41
Tempo de UCP (s)	0,083	0,217	0,233	0,317	0,500	1,117	2,783	5,583
Tempo de UCP/inserção (ms)	0,08	0,22	0,23	0,32	0,50	1,12	2,78	5,58
Tempo de UCP/particion.(ms)	6,94	14,44	9,33	5,46	5,56	3,33	2,78	3,95

Tabela 16: Configuração da árvore após a etapa Inserção I.

Dimensão	2	4	8	16	32	64	128	255
Níveis na árvore	2	3	3	3	4	7	13	17
Nós na árvore	77	140	239	470	1.010	3.320	9.995	14.328
Folhas	76	137	232	445	912	2.493	5.000	6.042
Nós internos	1	3	7	25	98	827	4.995	8.286
Nós/ponto	0,01	0,01	0,02	0,05	0,10	0,33	1,00	1,43
Tamanho total do índice (KB)	308	560	956	1.880	4.040	13.280	39.980	57.312
Kbytes/ponto	0,03	0,06	0,10	0,19	0,40	1,33	4,00	5,73
Total de entradas	10.076	10.139	10.238	10.469	11.009	13.319	19.994	24.327
Espaço total para entradas	15.708	15.820	14.340	14.100	15.150	23.240	29.985	28.656
Taxa de ocupação	64%	64%	71%	74%	72%	57%	66%	84%

Tabela 17: Etapa: Consulta.

Dimensão	2	4	8	16	32	64	128	255
Pontos Encontrados	1.025	1.018	1.018	1.018	1.018	1.018	1.018	1.018
Páginas acessadas/consulta	8,22	130,16	188,03	224,97	161,48	74,26	98,77	91,78
Páginas acessadas/acerto	8,02	127,86	184,71	220,99	158,62	72,95	97,02	90,15
Tempo de UCP (s)	0,383	5,533	7,533	4,750	2,150	0,783	1,367	2,333
Tempo de UCP/consulta (ms)	0,38	5,53	7,53	4,75	2,15	0,78	1,37	2,33
Tempo de UCP/acerto (ms)	0,37	5,44	7,40	4,67	2,11	0,77	1,34	2,29

Tabela 18: Etapa: Remoção.

Dimensão	2	4	8	16	32	64	128	255
Nós eliminados	31	76	59	96	166	565	199	701
Nós eliminados/remoção	0,02	0,04	0,03	0,05	0,08	0,28	0,10	0,35
Nós reinseridos	3.131	4.125	1.711	1.344	996	1.130	0	0
Reinserções/nó eliminado	101	54	29	14	6	2	0	0
Reinserções/remoção	2	2	1	1	0	1	0	0
Páginas acessadas/remoção	9,05	67,57	91,62	107,06	80,23	43,95	54,71	47,14
Tempo de UCP (s)	0,783	2,917	3,133	2,383	2,433	2,917	3,083	6,083
Tempo de UCP/remoção (ms)	0,39	1,46	1,57	1,19	1,22	1,46	1,54	3,04