Universidade Estadual de Londrina

PAULO HENRIQUE DE OLIVEIRA

MELHORANDO O DESEMPENHO DE MÉTODOS DE ACESSO MÉTRICOS DINÂMICOS COM PIVÔS ADICIONAIS LOCAIS E ANTECIPAÇÃO DEINFORMAÇÕES

Londrina
2015

PAULO HENRIQUE DE OLIVEIRA

MELHORANDO O DESEMPENHO DE MÉTODOS DE ACESSO MÉTRICOS DINÂMICOS COM PIVÔS ADICIONAIS LOCAIS E ANTECIPAÇÃO DEINFORMAÇÕES

Dissertação apresen tada ao Program a de Mestrado em Ciência da Co mputação da Univers idade Estadual de Londrina para obten ção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Daniel dos Santos Kaster.

Londrina
2015

Catalogação elaborada pela Divisão de Processos Técnicos da Biblioteca Central da Universidade Estadual de Londrina

Dados Internacionais de Catalogação-na-Publicação (CIP)

O48m Oliveira, Paulo Henrique de.
Melhorando o desempenho de métodos de acesso métricos dinâmicos com pivôs adicionais locais e antecipação de informações / Paulo Henrique de Oliveira. – Londrina, 2015.
75 f. : il.

Orientador: Daniel dos Santos Kaster.
Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2015.
Inclui bibliografia.

1. Banco de dados – Gerência – Teses. 2. Estrutura de dados (Computação) – Teses. 3. Sistemas multimídia – Teses. 4. Computadores – Controle de acesso – Teses. I. Kaster, Daniel dos Santos. II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU 519.68.023

PAULO HENRIQUE DE OLIVEIRA

MELHORANDO O DESEMPENHO DE MÉTODOS DE ACESSO MÉTRICOS DINÂMICOS COM PIVÔS ADICIONAIS LOCAIS E ANTECIPAÇÃO DEINFORMAÇÕES

Dissertação apresen tada ao Program a de Mestrado em Ciência da Co mputação da Univers idade Estadual de Londrina para obten ção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Daniel dos Santos Kaster
Universidade Estadual de Londrina – UEL

Prof. Dr. Humberto Luiz Razente
Universidade Federal de Uberlândia – UFU

Prof. Dr. Adilson Luiz Bonifácio
Universidade Estadual de Londrina – UEL

Prof. Dr. Jacques Duílio Brancher
Universidade Estadual de Londrina – UEL

Londrina, 28 de julho de 2015

A Deus, por guiar meus passos e permitir a realização de mais uma etapa.
Aos meus pais, Paulo e Neiva, e à minha irmã, Aline, pelo apoio incondicional.
Ter vocês ao meu lado fez toda a diferença. Muito obrigado!

AGRADECIMENTOS

A Deus, razão da minha existência, por nutrir em mim mais um sonho, por guiar meus passos e por me abençoar através de pessoas incríveis, que tanto me fizeram crescer nesse período.

Ao meu orientador, por seu exemplo como pessoa, pelos ensinamentos, pela ajuda nos momentos difíceis. Por ter falado sobre o mestrado naquela primeira conversa! Afinal de contas, tudo começou ali. Kaster, obrigado por tudo. Muitos caminhos se abriram para mim através de você e serei eternamente grato por isso.

Aos meus pais e à minha irmã, por estarem sempre ao meu lado, por me apoiarem incondicionalmente, por serem quem vocês são na minha vida. Eu não teria chegado até aqui sem o apoio de vocês.

Ao Prof. Dr. Caetano Traina Junior, por ter me recebido durante o tempo em que estive em São Carlos, por sua supervisão. Ao pessoal do GBdI, pelas novas amizades, por todos os conselhos, por toda a ajuda.

Aos amigos e familiares, por torcerem por mim e por me apoiarem. Muito obrigado!

"Consagre ao Senhor tudo o que você faz,
e os seus planos serão bem-sucedidos."
(Bíblia – NVI, Pv 16:3)

OLIVEIRA, P. H.. **Melhorando o desempenho de métodos de acesso métricos dinâmicos com pivôs adicionais locais e antecipação de informações**. 75 p. Dissertação de Mestrado (Mestrado em Ciência da Computação) — Universidade Estadual de Londrina, Londrina–PR, 2015.

RESUMO

Nos últimos anos, o crescimento do volume de dados complexos tem sido acelerado por constantes avanços tecnológicos em dispositivos eletrônicos. Neste trabalho, são considerados dados complexos quaisquer dados não representáveis por tipos tradicionais, como números, caracteres, datas e textos curtos. Dados multimídia, dados georreferenciados e séries temporais são exemplos dessa categoria de dados. A *relação de ordem* é uma propriedade que permite identificar qual elemento precede o outro, segundo algum critério, em cada par de elementos do domínio. Visto que estruturas de indexação tradicionais são baseadas nessa propriedade, elas não são adequadas para dados complexos. Entretanto, existem estruturas apropriadas para domínios complexos, como os Métodos de Acesso Métricos (MAMs). Há diversos MAMs relatados na literatura, categorizados de diferentes formas dependendo dos fatores que são levados em conta para estruturar os dados. Os fatores *tipo de pivô* e *dinamicidade da estrutura* estão diretamente relacionados um ao outro. Neste trabalho, pivôs são elementos que agem como representantes de certas regiões do espaço de busca e são usados para podar elementos irrelevantes durante a execução de consultas. Diz-se que um pivô é global quando todos os elementos do conjunto de dados são referenciados a ele, enquanto um pivô é local quando somente uma porção do conjunto de dados é referenciada a ele. Porque pivôs globais são referenciados por todo o conjunto de dados, eles têm um alto impacto no processo de poda de elementos irrelevantes, uma vez que um único pivô global pode ser usado para descartar uma grande quantidade de elementos irrelevantes. No entanto, MAMs baseados em pivôs globais podem ter sua dinamicidade comprometida pelo fato de eventuais atualizações relacionadas a pivôs precisarem ser propagadas por toda a estrutura. Pivôs locais, por outro lado, permitem que a manutenção ocorra localmente ao preço de um menor poder de poda. Nesse contexto, esta dissertação teve como alvo melhorar o desempenho de MAMs dinâmicos sem comprometer sua dinamicidade, uma vez que várias aplicações manipulam dados complexos *online* e, consequentemente, demandam índices dinâmicos e eficientes para serem bem-sucedidas. Esta dissertação apresenta duas técnicas para aumentar o poder de poda de MAMs dinâmicos: (i) usar pivôs adicionais locais para reduzir cálculos de distância e (ii) antecipar informações de nós filhos para reduzir acessos a disco desnecessários. Ambas as técnicas foram aplicadas a um MAM dinâmico e avaliadas sobre conjuntos de dados reais, reduzindo o tempo de execução em até mais de 50% para consultas por similaridade sobre conjuntos de dados com dimensionalidades e cardinalidades moderadas e altas.

Palavras-chave: Consultas por similaridade. Métodos de Acesso Métricos. Pivôs adicionais locais cortantes. Antecipação de informações.

OLIVEIRA, P. H.. **Improving the performance of dynamic metric access methods with local additional pivots and anticipation of information**. 75 p. Master's Thesis (Master in Science in Computer Science) — State University of Londrina, Londrina–PR, 2015.

ABSTRACT

In recent years, the growth of complex data has been accelerated by constant technological advances in electronic devices. In this work, complex data are considered as any data not representable by traditional types, such as numbers, characters, dates and short texts. Multimedia data, georeferenced data and time series are examples of this category of data. The *order relation* is a property that allows identifying which element precedes the other, according to some criterion, in each pair of elements of the domain. Since traditional index structures are based on this property, they are not suitable for complex data. Nevertheless, there are structures well-suited for complex domains, such as the Metric Access Methods (MAMs). There are several MAMs related in the literature, categorized in different ways depending on which factors are taken into account to structure the data. The factors *pivot type* and *structure dynamicity* are directly related to each other. In this work, pivots are elements that act as representatives of certain regions of the search space and are employed to prune irrelevant elements during the query execution. It is said that a pivot is global when all elements of the dataset are referenced to it, whereas a pivot is local when only a portion of the dataset is referenced to it. Because global pivots are referenced by the whole dataset, they have a high impact in the pruning process of irrelevant elements, once that a single global pivot can be used to discard a large amount of irrelevant elements. However, MAMs based on global pivots may have their dynamicity compromised by the fact that eventual pivot-related updates need to be propagated through the entire structure. Local pivots, on the other hand, allow the maintenance to occur locally at the price of a lower pruning ability. In this context, this dissertation aimed at improving the performance of dynamic MAMs without harming their dynamicity, once that several applications handle online complex data and, consequently, demand efficient dynamic indexes to be successful. This dissertation presents two techniques for improving the pruning ability of dynamic MAMs: (i) using local additional pivots to reduce distance calculations and (ii) anticipating information from child nodes to reduce unnecessary disk accesses. Both techniques have been applied to a dynamic MAM and evaluated over real datasets, reducing execution time in up to more than 50% for similarity queries posed on datasets ranging from moderate to high dimensionality and cardinality.

Keywords: Similarity queries. Metric Access Methods. Cutting local additional pivots. Anticipation of information.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACIR	*Anticipation of Child Information regarding Representatives*
ACIP	*Anticipation of Child Information regarding Pivots*
ALOI	*Amsterdam Library of Object Images*
CLAP	*Cutting Local Additional Pivots*
GBDI	Grupo de Bases de Dados e Imagens
ICMC	Instituto de Ciências Matemáticas e de Computação
MAE	Método de Acesso Espacial
MAM	Método de Acesso Métrico
MST	*Minimal Spanning Tree*
OID	*Object IDentifier*
SGBD	Sistema Gerenciador de Bancos de Dados
USP	Universidade de São Paulo

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, o avanço tecnológico de dispositivos eletrônicos, considerando, principalmente, a popularização de dispositivos móveis, como *smartphones* e *laptops*, tem acelerado o crescimento do volume de dados complexos. Neste trabalho, são considerados dados complexos quaisquer dados não representáveis por tipos tradicionais, como números, caracteres, datas e textos curtos. Dados multimídia, dados georreferenciados e séries temporais são alguns exemplos dessa categoria de dados. Algumas razões para esse crescimento são: diminuição do preço de câmeras digitais e de outros dispositivos de captura de vídeo, câmeras de alta definição embutidas em telefones móveis, ferramentas mais amigáveis para processamento e edição de imagens e vídeos, aquisição de dados provenientes de equipamentos para exames médicos, captura de dados por meio de redes de sensores e aumento de largura de banda das conexões de rede. O sucesso de serviços de compartilhamento de conteúdo multimídia como YouTube e Flickr e das redes sociais é mais uma evidência desse crescimento [1].

Em domínios de dados tradicionais, os elementos possuem *relação de ordem*, permitindo identificar qual elemento precede o outro, segundo algum critério, em cada par de elementos do domínio. Devido a essa propriedade, grande parte das estruturas de indexação implementadas por Sistemas Gerenciadores de Bancos de Dados (SGBDs) é capaz de executar consultas eficientemente. Entretanto, a relação de ordem não se aplica à grande maioria dos domínios complexos [2]. Uma vez que as estruturas de indexação tradicionais baseiam-se nessa propriedade, elas não são adequadas para essa categoria de dados. Entretanto, existem estruturas apropriadas para domínios complexos, como os Métodos de Acesso Métricos (MAMs).

Há inúmeros MAMs relatados na literatura, categorizados de diferentes maneiras dependendo de quais fatores são levados em conta para estruturar os dados. Em geral, considera-se o tipo de resposta, a dinamicidade da estrutura, a forma de particionamento do espaço de busca e o tipo de pivô. Quanto ao tipo de resposta, os MAMs podem ser exatos ou aproximados. No caso dos MAMs exatos, a resposta é sempre exata. Já no caso dos MAMs aproximados, a resposta tem sua precisão relaxada para que as consultas sejam mais eficientes. Quanto à dinamicidade da estrutura, os MAMs podem ser dinâmicos ou estáticos. MAMs dinâmicos permitem adicionar e remover elementos a qualquer momento sem degradar sua estrutura. MAMs estáticos requerem a existência a priori do conjunto de dados a ser indexado e não permitem atualizações posteriores. Considerando as formas de particionamento do espaço de busca, os tipos básicos incluem: particionamento por bola [3], particionamento por hiperplano generalizado [3] e particionamento por meio excluído [4]. Por fim, considerando o tipo de pivô, os MAMs podem ser baseados em pivôs

globais ou em pivôs locais.

No contexto deste trabalho, pivôs são elementos que atuam como representantes de determinadas regiões do espaço de busca e são utilizados no processo de poda de elementos irrelevantes durante a execução de consultas. Diz-se que um pivô é global quando todos os elementos do conjunto de dados são referenciados a ele. Em contrapartida, um pivô é local quando apenas uma parcela dos elementos do conjunto é referenciada a ele.

Propostas recentes de métodos baseados em pivôs globais apresentam bom desempenho para consultas. Por serem referenciados por todo o conjunto de dados, pivôs globais têm um forte impacto no processo de poda de elementos irrelevantes, pois um único pivô global pode ser utilizado para descartar uma grande quantidade de elementos. No entanto, métodos baseados em pivôs globais podem ter sua dinamicidade comprometida pelo fato de certas atualizações relacionadas a pivôs precisarem ser propagadas por toda a estrutura.

Pivôs locais, por outro lado, permitem que a manutenção ocorra de forma localizada. Como cada pivô local é referenciado por uma pequena parcela do conjunto de dados, apenas essa parcela precisa ser atualizada em casos de mudanças de pivô. Isso contribui para a dinamicidade de métodos baseados nesse tipo de pivô, porém ao preço de um menor poder de poda. Nesse contexto, o desafio desta dissertação é melhorar o desempenho de MAMs dinâmicos sem comprometer sua dinamicidade, uma vez que diversas aplicações manipulam dados complexos *online* e, consequentemente, demandam índices dinâmicos e eficientes para serem bem-sucedidas.

Esta dissertação apresenta duas técnicas para aumentar o poder de poda de MAMs dinâmicos: (i) usar pivôs adicionais locais para reduzir cálculos de distância e (ii) antecipar informações de nós filhos para reduzir acessos a disco desnecessários. A importância da redução do número de cálculos de distância está no fato de que, em diversos domínios complexos, são utilizadas funções de distância com custos computacionais mais elevados. Nessas situações, os cálculos de distância apresentam um impacto maior no desempenho das consultas. Reduzir o número de acessos a disco, por sua vez, é essencial em qualquer situação, visto que o tempo utilizado para um acesso a disco é de seis ordens de magnitude maior do que o tempo utilizado para um acesso a memória. Dessa forma, busca-se reduzir o tempo total de execução de consultas sobre dados complexos. Ambas as técnicas foram desenvolvidas e aplicadas a um MAM dinâmico, avaliadas sobre conjuntos de dados reais com dimensionalidades e cardinalidades moderadas e altas. Os resultados de tais avaliações mostram ganhos de até mais de 50% no tempo de execução de consultas sobre os conjuntos de dados complexos utilizados.

1.1 Organização do trabalho

Esta dissertação está organizada conforme apresentado a seguir.

- O Capítulo 2 apresenta a fundamentação teórica, abordando consultas sobre dados complexos e trabalhos correlatos envolvendo propostas de MAMs baseados em pivôs;
- O Capítulo 3 apresenta o desenvolvimento deste trabalho;
- O Capítulo 4 descreve os experimentos realizados e apresenta os resultados obtidos;
- O Capítulo 5 apresenta as conclusões do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta conceitos fundamentais para o desenvolvimento desta dissertação, organizados conforme segue: a Seção 2.1 introduz conceitos referentes a consultas por similaridade; a Seção 2.2 descreve os tipos básicos de consulta por similaridade; a Seção 2.3 descreve algumas das principais funções de distância, utilizadas na construção de métodos de acesso para dados complexos e na execução de consultas; a Seção 2.4 apresenta o conceito de espaços métricos, fundamental para a representação dos mais diversos tipos de dados complexos; a Seção 2.5 descreve os tipos básicos de particionamento do espaço métrico; a Seção 2.6 descreve alguns dos principais métodos de acesso para consultas por similaridade; a Seção 2.7 apresenta trabalhos correlatos.

2.1 Consultas sobre dados complexos

Dados complexos, em sua maioria, não possuem *relação de ordem*. Por essa razão, não é possível empregar operadores relacionais de comparação diretamente sobre eles a fim de tentar identificar alguma precedência. Isso vale para os operadores $<$, $\leq$, $>$ e $\geq$. Também não é comum empregar os operadores $=$ e $\neq$ sobre dados complexos, pois é bastante improvável que dois elementos complexos sejam exatamente iguais. Considere duas imagens como exemplo. O fato de um único *pixel* ser diferente entre elas já seria o suficiente para afirmar que elas não são iguais. Por isso, em domínios complexos, usualmente, faz mais sentido realizar consultas que considerem a similaridade entre os elementos.

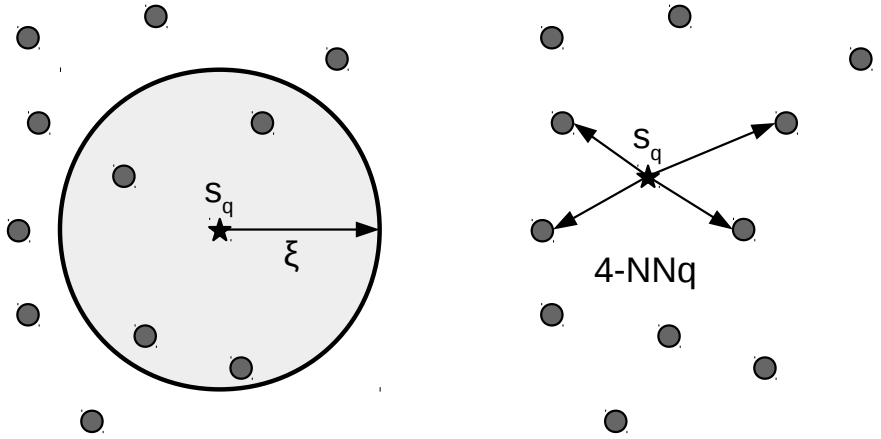
De forma a possibilitar a realização de consultas sobre dados complexos, é necessário trabalhar com características extraídas a partir do conteúdo dos elementos do conjunto de dados. A recuperação de dados complexos por meio dessas características é conhecida como *recuperação baseada em conteúdo*. O conjunto de características extraído, chamado de *vetor de características* ou *assinatura*, é usado no lugar dos dados originais como base para as consultas. Usualmente, dados complexos são comparados por meio de relações de dissimilaridade entre pares de vetores de características. Isso acontece empregando-se uma *função de distância*, cujo valor de retorno representa o quão dissimilares os dois vetores de características são um do outro. Por isso, consultas realizadas sobre dados complexos são chamadas de *consultas por similaridade*, uma vez que recuperam do conjunto de dados os elementos mais similares a um ou mais elementos de consulta fornecidos. Os critérios de seleção dos elementos mais similares dependem do tipo de consulta por similaridade [5].

2.2 Tipos básicos de consulta por similaridade

Existem vários tipos de consulta por similaridade, que vão desde seleções e junções por similaridade até consultas por similaridade agregada. Os dois tipos mais comuns são a *Range query* e a *k-Nearest Neighbors query* (Figura 1) [6].

Range query — Rq: Dado um limiar máximo ξ, a consulta recupera toda tupla t_i da relação R cujo valor s_i do atributo S_j, que diz respeito ao vetor de características do elemento, satisfaz a condição $\delta(s_i, s_q) \leq \xi$, onde s_q é o valor que o elemento de consulta possui para o atributo S_j e δ é a função de distância que retorna a dissimilaridade entre s_i e s_q. Considerando R como uma relação de imagens, um exemplo de Range query seria: "Selecione as imagens que são similares à imagem Q em até 5 unidades".

k-Nearest Neighbors query — k-NNq: Dado um valor inteiro $k \geq 1$, a consulta recupera k tuplas t_i da relação R cujos valores s_i do atributo S_j têm a menor distância do elemento de consulta s_q de acordo com a função de distância δ. Um exemplo de k-Nearest Neighbors query seria: "Selecione as 4 imagens mais similares à imagem Q".



(a) Range query. (b) k-Nearest Neighbors query.

Figura 1 – Consultas por similaridade mais comuns.

Na Figura 1, o exemplo de Rq apresenta uma consulta que retorna os 4 elementos cujas distâncias de s_q são menores ou iguais a ξ. O exemplo de k-NNq mostra uma consulta que retorna os 4 elementos mais próximos a s_q.

2.3 Funções de distância

Existem várias funções de distância propostas na literatura. Essa variedade se dá pelo fato de que diversas áreas do conhecimento fazem uso de funções de distância. Além disso, existem funções de distância mais apropriadas para determinados contextos do que outras. Esta seção apresenta algumas das principais, entre as quais encontram-se as que são utilizadas neste trabalho.

2.3.1 Família de distâncias Minkowski

A família de distâncias Minkowski [7], denominada L_p, é comumente empregada em espaços vetoriais, isto é, espaços em que os elementos dizem respeito a valores numéricos estruturados em vetores, o que implica um número fixo de dimensões. Um elemento de um espaço vetorial n-dimensional é representado por n coordenadas de valores reais $(x_1, \ldots, x_n)$. Assim sendo, a família de distâncias L_p é definida como:

$$L_p((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{2.1}$$

A função de distância L_1 ($p = 1$), também chamada de *City Block* ou *Manhattan*, consiste no somatório dos módulos das diferenças entre as coordenadas. Nesse caso, considerando um plano (bidimensional), o conjunto de pontos equidistantes a um determinado ponto c por um raio de abrangência r forma um losango cujos diâmetros são paralelos aos eixos das coordenadas (Figura 2a). A distância L_1 entre dois elementos x e y é definida como:

$$L_1(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{2.2}$$

A função de distância L_2, denominada Euclidiana, diz respeito à função comumente utilizada para calcular a distância entre dois pontos no espaço Euclidiano. O conjunto de pontos equidistantes a c por um raio r forma uma circunferência no plano (Figura 2b). A distância L_2 entre dois elementos x e y é definida como:

$$L_2(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.3}$$

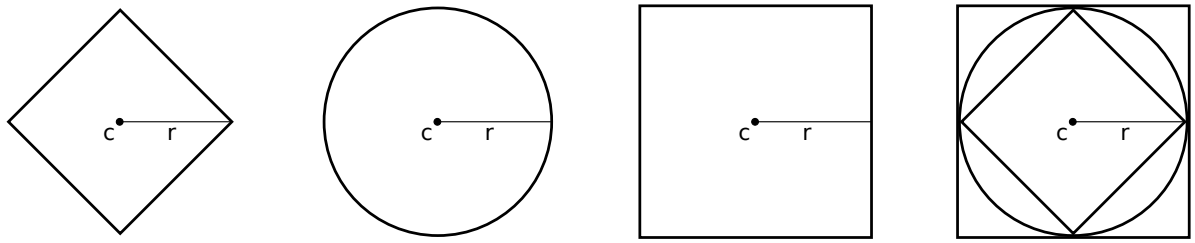
Calculando-se o limite da equação 2.1 com $p \to \infty$, obtém-se a função de distância L_∞, também chamada de *Infinity* ou *Chebyshev*. Nesse caso, o conjunto de pontos equidistantes a c por um raio r forma um quadrado com seus lados paralelos aos eixos das coordenadas (Figura 2c). A distância L_∞ entre dois elementos x e y é definida como:

$$L_\infty(x, y) = \max_{i=1}^{n} |x_i - y_i| \tag{2.4}$$

A Figura 2d ilustra, juntamente, as formas geométricas definidas por pontos equidistantes através das funções de distância L_1, L_2 e L_∞.

2.3.2 Função de distância Mahalanobis

Considerando histogramas de cores de imagens, a família de distâncias Minkowski trata todos os seus *bins* de forma independente, sem levar em conta o fato de que certos



(a) L_1. (b) L_2. (c) L_∞. (d) L_1, L_2 e L_∞.

Figura 2 – Família de distâncias Minkowski.

pares de *bins* correspondem a características perceptualmente mais similares do que outros pares. Para abordar essa questão, foi proposta a função de distância Mahalanobis [8, 9], conhecida também como *distância quadrática de histogramas*. Dados dois histogramas de cores $x = (x_1, \ldots, x_n)$ e $y = (y_1, \ldots, y_n)$, a distância Mahalanobis entre eles é:

$$\sqrt{(x-y)^T A (x-y)}, \tag{2.5}$$

onde $A = [a_{ij}]$ é uma matriz definida positiva simétrica $n \times n$ de elementos a_{ij} que denotam a similaridade entre os *bins* i e j. O valor de a_{ij} é dado por:

$$a_{ij} = 1 - \frac{d_{ij}}{\max[d_{ij}]}, \qquad \text{onde } d_{ij} = |x_i - y_j|$$

A função de distância Mahalanobis tem sido usada em vários sistemas de recuperação baseada em conteúdo para histogramas de cores. Foi mostrado que o uso dessa função de distância conduz a resultados perceptualmente mais desejáveis quando comparada à família de distâncias Minkowski, pelo fato de ser considerada a similaridade entre os *bins* do histograma de cores [10].

2.4 Espaço métrico

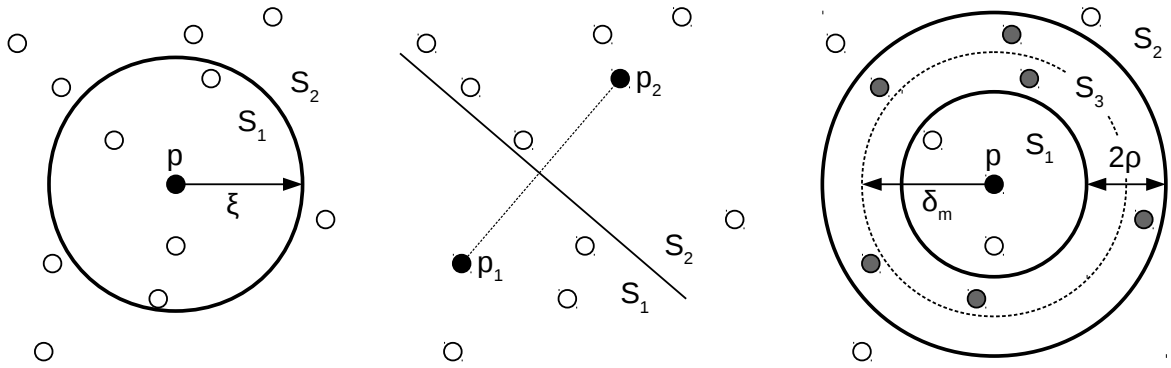
Um espaço métrico é definido por um par $\langle \mathbb{S}, \delta \rangle$, onde $\mathbb{S}$ é o domínio dos vetores de características indexados e δ é uma função de distância $\delta : \mathbb{S} \times \mathbb{S} \mapsto \mathbb{R}$, também chamada de métrica, que possui propriedades conhecidas como *postulados do espaço métrico* [11]:

- $\forall x, y \in \mathbb{S}, \delta(x, y) \geq 0$ — não-negatividade
- $\forall x, y \in \mathbb{S}, \delta(x, y) = \delta(y, x)$ — simetria
- $\forall x, y \in \mathbb{S}, x = y \iff \delta(x, y) = 0$ — identidade
- $\forall x, y, z \in \mathbb{S}, \delta(x, z) \leq \delta(x, y) + \delta(y, z)$ — desigualdade triangular

Essas propriedades, principalmente a desigualdade triangular, permitem que Métodos de Acesso Métricos — assim chamados justamente por serem baseados em espaços métricos — utilizem técnicas eficientes para responder a consultas por similaridade. Uma característica importante dos espaços métricos é que, além de englobarem espaços vetoriais, eles são capazes de englobar espaços adimensionais, que são domínios cujos elementos não podem ser identificados através de coordenadas em eixos ortogonais. Quaisquer tipos de dados podem ser imersos em um espaço métrico, como coordenadas geográficas, imagens, sons, palavras e sequências de DNA. Contudo, é importante que os dados a serem indexados sejam devidamente representados ao terem suas características extraídas, para que seja possível o uso de uma métrica adequada para o domínio em questão.

2.5 Tipos básicos de particionamento do espaço métrico

Um princípio fundamental de qualquer estrutura de indexação é o particionamento, que tem como objetivo dividir o espaço de busca em subgrupos de forma que, ao realizar-se uma consulta, apenas alguns desses subgrupos sejam acessados [11]. Dado um conjunto de elementos $S \subseteq \mathbb{S}$ em um espaço métrico $\langle \mathbb{S}, \delta \rangle$, [3] apresenta o *particionamento por bola* e o *particionamento por hiperplano generalizado*, enquanto [4] apresenta o *particionamento por meio excluído*.



(a) Por bola. (b) Por hiperplano generalizado. (c) Por meio excluído.

Figura 3 – Tipos básicos de particionamento do espaço métrico.

Particionamento por bola: Divide o conjunto S em subconjuntos S_1 e S_2 através de um corte esférico em relação a $p \in S$, onde p é um pivô escolhido por meio de algum critério, como ilustra a Figura 3a. Seja δ_m a mediana de $\{\delta(o_i, p), \forall o_i \in S\}$. Então, todo $o_j \in S$ é distribuído entre S_1 ou S_2 de acordo com as seguintes regras:

- $S_1 \leftarrow \{o_j | \delta(o_j, p) \leq \delta_m\}$
- $S_2 \leftarrow \{o_j | \delta(o_j, p) \geq \delta_m\}$

As condições redundantes $\leq$ e $\geq$ garantem o balanceamento quando o valor mediano δ_m não for único. Isso é possível atribuindo-se cada elemento cuja distância é o valor mediano a um dos subconjuntos de modo arbitrário, mas balanceado.

Particionamento por hiperplano generalizado: Divide S em subconjuntos S_1 e S_2 fazendo-se uso de dois pivôs $p_1, p_2 \in \mathbb{S}$ (Figura 3b). Todos os outros objetos $o_j \in S$ são atribuídos a S_1 ou S_2 de acordo com as seguintes regras:

- $S_1 \leftarrow \{o_j | \delta(o_j, p_1) \leq \delta(o_j, p_2)\}$
- $S_2 \leftarrow \{o_j | \delta(o_j, p_1) \geq \delta(o_j, p_2)\}$

Esse tipo de particionamento não garante uma divisão balanceada dos elementos. Além disso, escolher os dois pivôs adequadamente é desafiador.

Particionamento por meio excluído: Divide S em subconjuntos S_1, S_2 e S_3, estendendo o particionamento por bola, como mostra a Figura 3c. Embora consultas por similaridade busquem elementos dentro de uma pequena vizinhança do elemento de consulta, quando este aparece próximo à fronteira do particionamento, tipicamente é preciso acessar os subconjuntos de ambos os lados. Diante disso, foi proposto o particionamento por meio excluído, cuja ideia principal é desconsiderar elementos próximos à fronteira do particionamento ao definir os subconjuntos S_1 e S_2. Os elementos excluídos dão origem ao subconjunto S_3. Assim, a busca de elementos similares ignora pelo menos um dos subconjuntos S_1 ou S_2, uma vez que o alcance da busca é menor do que a largura da zona de exclusão. Dada a largura da zona de exclusão 2ρ, o particionamento é definido pelas seguintes regras:

- $S_1 \leftarrow \{o_j | \delta(o_j, p) \leq \delta_m - \rho\}$
- $S_2 \leftarrow \{o_j | \delta(o_j, p) > \delta_m + \rho\}$
- $S_3 \leftarrow$ caso contrário

Esse tipo de particionamento não garante uma divisão balanceada dos elementos entre os subconjuntos S_1 e S_2.

2.5.1 Extensões dos tipos básicos de particionamento

Os tipos básicos de particionamento podem ser estendidos. Por exemplo:

- Pode-se estender o particionamento binário para um particionamento múltiplo, isto é, o conjunto S pode ser dividido em mais de 2 subconjuntos.

- O processo de particionamento pode continuar recursivamente até que uma estrutura hierárquica seja construída.

Combinações específicas dessas estratégias deram origem a diversas estruturas de indexação, disponíveis no *survey* realizado por [11].

2.6 Métodos de acesso para consultas por similaridade

Para agilizar a execução de consultas por similaridade sobre dados complexos, são utilizados mecanismos para indexar o conjunto de características extraídas dos elementos. Estruturas de dados que permitem indexar e recuperar rapidamente uma determinada informação armazenada são também conhecidas como *métodos de acesso*. Métodos de acesso convencionais, implementados por SGBDs, são adequados apenas para dados tradicionais, visto que seu funcionamento é baseado na relação de ordem total. Os primeiros métodos de acesso foram propostos para operar sobre dados que possuem relação de ordem total, como a família B-tree [12, 13].

Em domínios de dados complexos, devem ser utilizados outros tipos de métodos de acesso, capazes de considerar a similaridade entre elementos. Nessa categoria, encontram-se métodos de acesso que suportam domínios de dados representados em espaços dimensionais, principalmente dados espaciais. Esses métodos são conhecidos como Métodos de Acesso Espaciais (MAEs). A K-D-B-tree [14] e a R-tree [15] foram os trabalhos pioneiros nessa linha de pesquisa. Em seguida, a R^+-tree [16] e a R^*-tree [17] foram propostas como extensões da R-tree, bem como algumas adaptações [18, 19] (para uma comparação geral entre MAEs, veja [20]). Entretanto, para dados de alta dimensionalidade e adimensionais, todos esses métodos de acesso mostraram-se inadequados.

Foram propostos métodos de acesso voltados para dados imersos em espaços métricos, conhecidos como Métodos de Acesso Métricos (MAMs), já mencionados anteriormente. Tais métodos de acesso suportam naturalmente consultas por similaridade sobre dados complexos das mais variadas naturezas e dimensionalidades. Dados indexados por MAMs compreendem vetores de características extraídas a partir dos elementos do conjunto de dados. Tais características podem ser, por exemplo, atributos de forma, textura, histogramas de cores [21, 22] e resultados de transformações aplicadas sobre os dados [2]. Esse conjunto de características extraídas é utilizado na construção de MAMs.

A ideia geral de grande parte dos MAMs consiste em selecionar alguns elementos e colocá-los como representantes de determinados subconjuntos de dados. Tais elementos são também chamados de pivôs. Sempre que um elemento $s_i \in \mathbb{S}$ é inserido, é calculada e armazenada na estrutura a distância entre ele e seu representante. Esses valores de distância são posteriormente utilizados em consultas por similaridade para podar elementos irrelevantes por meio do uso da desigualdade triangular, definida na Seção 2.4. Conforme

a Figura 4, são descartados — i.e. desconsiderados sem a necessidade de calcular sua distância a s_q, pois sabe-se que não farão parte da resposta — os elementos s_i que satisfazem a uma das condições a seguir [23], onde r_i é o raio de cobertura do nó s_i, definidas através de manipulações da inequação original referente à propriedade de desigualdade triangular:

$$\delta(s_{rep}, s_i) + r_i < \delta(s_{rep}, s_q) - \xi \tag{2.6}$$

$$\delta(s_{rep}, s_i) - r_i > \delta(s_{rep}, s_q) + \xi \tag{2.7}$$

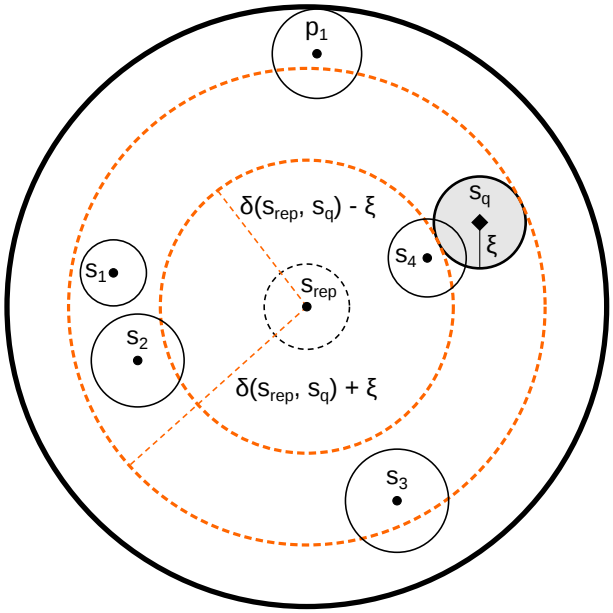


Figura 4 – Poda por desigualdade triangular. Circunferências pontilhadas dizem respeito a elementos descartados.

Na Figura 4, assume-se um espaço bidimensional para facilitar a visualização do mecanismo de poda. As condições apresentadas anteriormente representam, visualmente, o anel alaranjado centrado em s_{rep}. Todo elemento que intercepta esse anel — na figura, todos exceto s_{rep} — não pode ser podado, devendo ter sua distância até s_q calculada para avaliar se o respectivo nó filho deve ser visitado ou não.

Existem inúmeros MAMs relatados na literatura. Eles podem ser categorizados de diferentes maneiras, dependendo de quais fatores são levados em conta. Em geral, eles são categorizados considerando-se: o tipo de resposta, a dinamicidade da estrutura, a forma de particionamento do espaço de busca e o tipo de pivô, conforme descrito no Capítulo 1. O primeiro trabalho encontrado na literatura apresenta a BK-tree [23]. Foram propostas três técnicas para particionar espaços métricos de forma recursiva, cujo processo recursivo é materializado através das BK-trees. A partir desse trabalho, começaram a surgir outros MAMs, como os MAMs hierárquicos VP-tree [24], MVP-tree [25] e FQ-tree [26], que são estáticos. O primeiro MAM dinâmico desenvolvido foi a M-tree [27], que é uma estrutura de indexação hierárquica, balanceada, cujo particionamento é por bola, com crescimento

bottom-up e dois tipos de nós: índice e folha. A Slim-tree [28, 29] é uma evolução da M-tree e traz como melhorias a avaliação e minimização do grau de sobreposição entre seus nós e algoritmos de *split* mais eficientes: o algoritmo MST (*Minimal Spanning Tree*), baseado na árvore de cobertura minimal, e os algoritmos *Maximum Dissimilarity*, *Path Distance Sum* e *Reference Element*, propostos em [30]. Outros exemplos de MAMs dinâmicos incluem a DBM-tree [31, 32] e a M*-tree [33].

2.7 Trabalhos correlatos

A abordagem desta dissertação explora o uso de múltiplos pivôs de maneira a melhorar o desempenho de MAMs dinâmicos. Mais especificamente, este trabalho apresenta técnicas baseadas em múltiplos pivôs visando reduzir tanto o número de cálculos de distância quanto o número de acessos a disco, com o objetivo final de reduzir o tempo total de execução de consultas por similaridade. Nesse contexto, esta seção apresenta trabalhos correlatos cujas propostas tocam o objetivo de melhorar o desempenho de MAMs através de pivôs, porém destacando suas diferenças em relação à abordagem desta dissertação.

A Slim-tree, apresentada brevemente na Seção 2.6, é um dos MAMs diretamente relacionados a este trabalho de mestrado, tanto por ser baseada em pivôs locais quanto por ter sido utilizada como base para o desenvolvimento das técnicas desta dissertação. Uma vez que cada elemento inserido na Slim-tree tem sua distância ao elemento representante calculada e armazenada, o próprio elemento representante age como pivô local em seu nó. Outro exemplo é a M*-tree, proposta em [33], que possui múltiplos pivôs locais em cada nó, além do elemento representante. A ideia geral da M*-tree é que, para cada elemento de seus nós, armazena-se qual é seu vizinho mais próximo e sua distância até ele, formando um *grafo de vizinhos mais próximos*, também chamado de *NN-graph* (Figura 5). Com isso, em cada nó acessado no momento da consulta, são aplicadas heurísticas para selecionar os elementos que serão utilizados como pivôs locais no processo de poda.

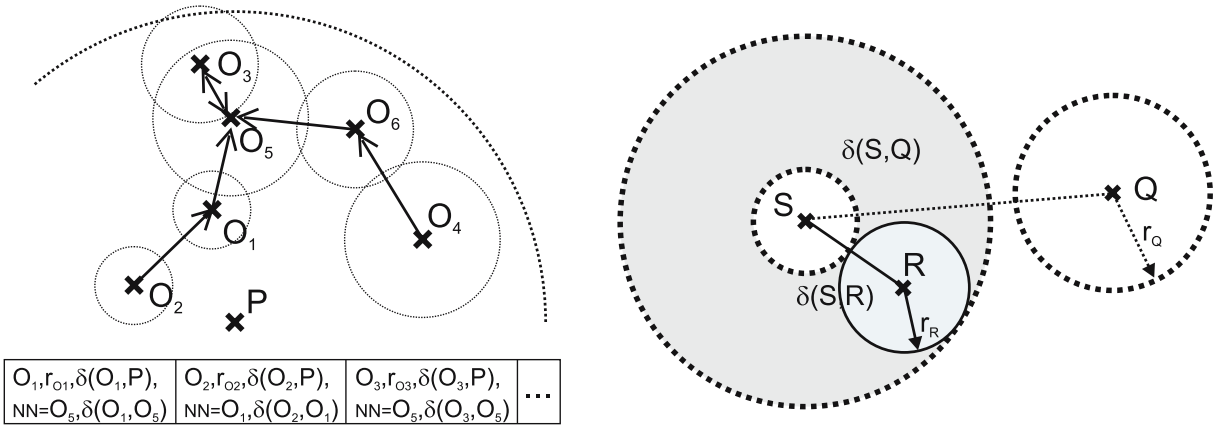


Figura 5 – Representação da M*-tree [33].

Na parte esquerda da Figura 5, é apresentado o NN-graph de um nó índice, onde P

é o representante, r_{O_i} é o raio de O_i e NN é o vizinho mais próximo de cada elemento O_i. Na parte direita da Figura 5, é apresentado um exemplo de Range query, cujo elemento de consulta é Q, em que o elemento S é utilizado como pivô local para tentar podar todo elemento que o tem como vizinho mais próximo, que, nesse caso, é o elemento R.

Tanto a Slim-tree quanto a M*-tree têm a vantagem de serem sempre dinâmicas, inclusive em casos de mudanças de pivô. Porém, os benefícios obtidos pelo uso de pivôs locais nesses MAMs são apenas em termos de cálculos de distância.

Outros exemplos de MAMs que utilizam múltiplos pivôs por nó são a MM-tree [34] e a Onion-tree [35]. Nessas estruturas, os pivôs adicionais são empregados, inclusive, no particionamento do espaço métrico, diferentemente da M*-tree, em que os pivôs adicionais não afetam o particionamento. Além disso, o principal objetivo da MM-tree e da Onion-tree é indexar dados na memória principal de forma a evitar sobreposições entre nós, o que pode torná-las altamente desbalanceadas conforme atualizações ocorrem.

Existem diversas propostas de MAMs baseados em pivôs globais. Nessas propostas, em geral, os pivôs selecionados são elementos do conjunto de dados que estão distantes. De fato, estudos desenvolvidos em [36] apontam que bons pivôs globais estão distantes uns dos outros. Essa afirmação faz sentido naturalmente, uma vez que pivôs muito próximos tendem a dar informações semelhantes no processo de poda.

Em [37, 38], é proposta a OMNI-family, que consiste em uma família de MAMs dinâmicos que podem ser implementados a partir de métodos de acesso existentes, como B-trees, R-trees e Slim-trees. O desenvolvimento dessa família de MAMs dinâmicos é feito através de técnicas baseadas em pivôs globais. A ideia é definir um conjunto de pivôs globais, também chamados de focos, e armazenar as distâncias de cada elemento da base de dados a esses focos. Tais distâncias podem ser armazenadas em um arquivo sequencial ou podem ser indexadas através de métodos de acesso. Durante as consultas, essas distâncias são acessadas e utilizadas para restringir a região de busca através da desigualdade triangular, reduzindo tanto o número de cálculos de distância quanto o número de acessos a disco. Nessa proposta, o espaço de busca não é particionado, isto é, os elementos não são subdivididos em regiões e não existe a noção de elementos representantes. Visualmente, é como se os elementos do conjunto de dados estivessem todos soltos no espaço. Assim, ao delimitar a região de busca através dos focos, utiliza-se o OID (*Object IDentifier*) de cada elemento que se encontra dentro dessa região de interesse para acessar suas distâncias aos focos, que estão armazenadas em um arquivo sequencial ou indexadas através de métodos de acesso, como B-trees, R-trees ou Slim-trees. Uma vez acessadas, as distâncias são utilizadas por meio da desigualdade triangular no processo de poda de elementos irrelevantes. O ponto negativo dessa proposta é que, após a definição dos focos, que ocorre no início da fase de construção, eles não podem mais ser mudados, pois isso exigiria a reconstrução de toda a estrutura. Assim, apesar de a OMNI-family ser considerada dinâmica em termos de

inserções e remoções de elementos, sua dinamicidade é comprometida no que diz respeito a atualizações relacionadas a pivôs.

Outro exemplo de MAM baseado em pivôs globais é a DF-tree [39]. Diferentemente da OMNI-family, que não particiona o espaço de busca, a DF-tree é uma estrutura hierárquica que utiliza o particionamento por bola para subdividir os elementos. Sua estrutura é bastante similar à da Slim-tree. Porém, além do representante de cada nó, que age como um pivô local, são utilizados pivôs globais no processo de poda de elementos irrelevantes. Sua principal vantagem é que o número de cálculos de distância reduz significativamente graças ao aumento no poder de poda garantido pelo pivô local de cada nó e pelos pivôs globais. Entretanto, a DF-tree também tem sua dinamicidade comprometida em casos de mudanças de pivôs globais.

Em [40, 41, 42, 43], é proposto o PP-Index, um índice baseado em permutação cujas buscas são aproximadas. Essa estrutura também é baseada em pivôs globais, a partir dos quais cada elemento do conjunto de dados é representado. Mais especificamente, cada elemento é representado por uma sequência ordenada de OIDs de pivôs globais, chamada de permutação, ordenada do pivô mais próximo ao pivô mais distante. A fim de poupar espaço em disco, são armazenados prefixos das permutações que representam os elementos (daí o nome *Permutation Prefix Index*). Tanto o uso de pivôs globais quanto o fato de as consultas serem aproximadas tornam esse método eficiente. Contudo, ele possui o mesmo ponto fraco que a OMNI-family e a DF-tree possuem em termos de dinamicidade.

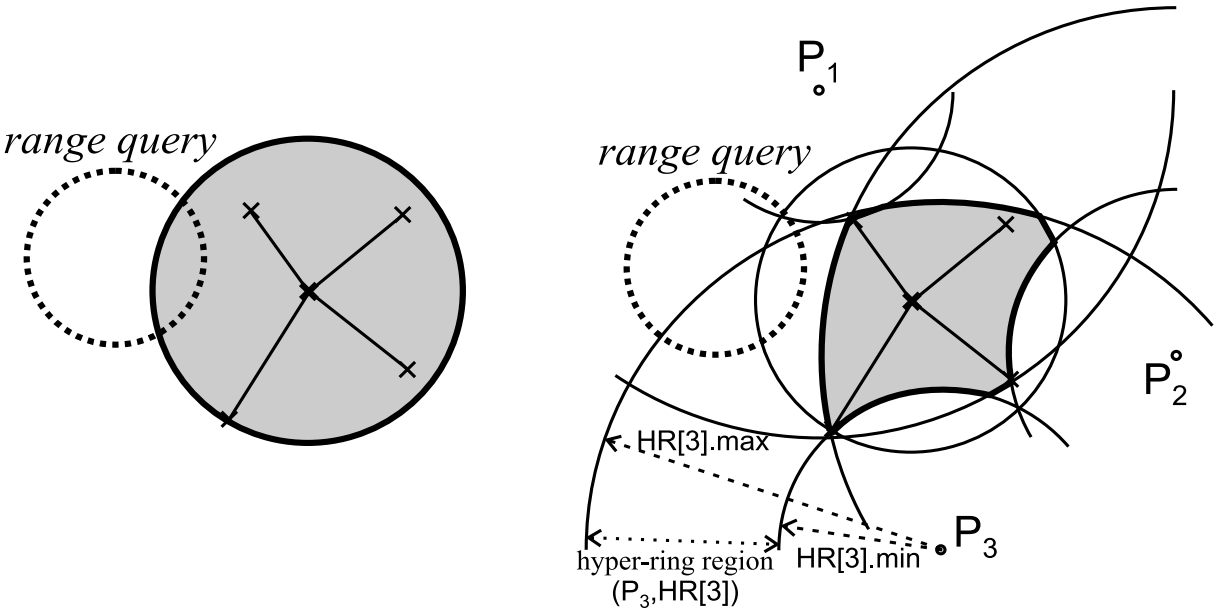


Figura 6 – Representação da PM-tree [44].

Em [45, 46, 44], é apresentada a PM-tree (*Pivoting M-tree*), uma estrutura particionada por bola que reduz a região de busca dos nós através da intersecção de hiperanéis definidos por pivôs globais (Figura 6). Se uma PM-tree tem n pivôs globais, cada nó ar-

mazena o início e o fim da região de n hiperanéis. O início da região do i-ésimo hiperanel é definido pela *menor* distância do i-ésimo pivô global a um elemento do nó ($HR[3].min$ na Figura 6), enquanto o fim da região do i-ésimo hiperanel é definido pela *maior* distância do i-ésimo pivô global a um elemento do nó ($HR[3].max$ na Figura 6). Uma evolução da PM-tree, a PM*-tree [33], acrescenta múltiplos pivôs locais da mesma forma como ocorre na M*-tree. Assim como nos MAMs anteriores baseados em pivôs globais, atualizações relacionadas a pivôs comprometem a dinamicidade da PM-tree e da PM*-tree. A PM*-tree, porém, é capaz de atualizar os pivôs locais herdados da M*-tree. Os conceitos propostos pela PM-tree, referentes a hiperanéis, foram formalizados em [47, 48] como *cut-regions* e podem ser aplicados a outros métodos de acesso.

2.8 Considerações finais

Este capítulo apresentou os principais conceitos referentes a consultas por similaridade sobre dados complexos. Existem diversas estruturas de indexação na literatura que foram criadas para acelerar tais consultas, com destaque para os Métodos de Acesso Métricos. Muitas dessas propostas baseiam-se no uso de pivôs globais, que, embora permitam melhorar o desempenho de busca, comprometem sua dinamicidade.

3 DESENVOLVIMENTO DO TRABALHO

Este trabalho de mestrado apresenta novas técnicas que possibilitam melhorar o desempenho de consultas por similaridade em MAMs dinâmicos hierárquicos baseados em pivôs locais. O uso de pivôs adicionais é um dos mecanismos utilizados por essas técnicas. No que diz respeito a pivôs de um modo geral, a escolha de pivôs em MAMs pode resultar em *pivôs globais* ou em *pivôs locais*. Pivôs globais são referenciados por todos os elementos do conjunto de dados indexado, enquanto pivôs locais são referenciados por apenas parte dos elementos. Os critérios de escolha de pivôs no particionamento do espaço métrico têm influência no desempenho de consultas por similaridade, visto que eles restringem a região de busca durante a poda de elementos irrelevantes.

Diante de vários exemplos de MAMs apresentados na Seção 2.7, é possível perceber vantagens e desvantagens decorrentes do uso de pivôs globais e pivôs locais. Pivôs globais têm impacto sobre grande parte do conjunto de dados indexado e são capazes de reduzir consideravelmente o número de cálculos de distância e de acessos a disco, o que aumenta a eficiência de consultas por similaridade. Porém, eles podem comprometer a dinamicidade da estrutura. Por outro lado, pivôs locais permitem que a manutenção ocorra localmente, sem comprometer a dinamicidade. Quanto às vantagens decorrentes do uso de pivôs locais em consultas por similaridade, pode-se dizer que o pivô principal, isto é, o representante de cada nó contribui para a redução do número de acessos a disco, uma vez que eles determinam as subregiões criadas com o particionamento do espaço de busca. Esse mecanismo pode ser visualizado em estruturas como a M-tree e a Slim-tree. Contudo, os benefícios obtidos diretamente pela utilização de pivôs locais *adicionais*, como acontece na M*-tree e na PM*-tree [33], referem-se apenas a cálculos de distância devido ao modo como esses MAMs foram concebidos.

Nesse contexto, esta dissertação apresenta e avalia novas técnicas envolvendo múltiplos pivôs locais, a fim de melhorar o desempenho de consultas por similaridade realizadas por MAMs ao mesmo tempo em que se mantém sua dinamicidade. As contribuições desta dissertação dividem-se em duas categorias: (i) utilização de pivôs adicionais locais, direcionada à redução do número de cálculos de distância, e (ii) antecipação de informações de nós filhos, direcionada à redução do número de acessos a disco.

Este capítulo está organizado da seguinte maneira: a Seção 3.1 apresenta a técnica CLAP de utilização de pivôs adicionais locais; a Seção 3.2 apresenta as técnicas ACIR e ACIR+ACIP de antecipação de informações de nós filhos; a Seção 3.3 descreve a aplicação das contribuições à Slim-tree; a Seção 3.4, por fim, descreve os algoritmos de construção e das consultas por similaridade realizadas pelas técnicas apresentadas.

3.1 Pivôs adicionais locais com a técnica CLAP

Esta seção descreve a técnica CLAP (*Cutting Local Additional Pivots*), desenvolvida a fim de aprimorar a capacidade de poda em MAMs por meio do uso da desigualdade triangular em consultas por similaridade, reduzindo-se o número de cálculos de distância. Diferentemente de outras estruturas que utilizam múltiplos pivôs locais, em que os pivôs definem a região de cobertura de cada nó, como na MM-tree [34] e na Onion-tree [35], os pivôs da técnica CLAP são utilizados no processo de poda para "cortar" a região de incerteza originalmente definida a partir do representante do nó, empregando-se a desigualdade triangular para cada pivô adicional local. Além disso, diferentemente da M*-tree, que também utiliza pivôs adicionais locais, cada pivô adicional dessa contribuição é referenciado por todos os elementos do nó.

Apesar de alterar a estrutura do nó, essa técnica não altera seu particionamento, isto é, o raio de cobertura de cada nó continua definido por seu representante s_{rep} — também chamado de *pivô principal* —, não comprometendo a dinamicidade da estrutura. Essa contribuição permite definir, no momento de criação de uma instância do MAM, o número de pivôs adicionais, que deve ser fixo em todos os nós. A técnica CLAP pode ser aplicada a quaisquer métodos cujo particionamento seja definido por elementos representantes, de modo que faça sentido a noção de pivôs adicionais locais.

Conforme descrito na Seção 2.6, considerando o pivô principal s_{rep}, o uso da desigualdade triangular no processo de poda consiste em descartar — isto é, desconsiderar sem a necessidade de calcular sua distância ao elemento de consulta s_q — todo elemento s_i que satisfaz a uma das Inequações 3.1 e 3.2 [23]:

$$\delta(s_{rep}, s_i) + r_i < \delta(s_{rep}, s_q) - \xi \tag{3.1}$$

$$\delta(s_{rep}, s_i) - r_i > \delta(s_{rep}, s_q) + \xi \tag{3.2}$$

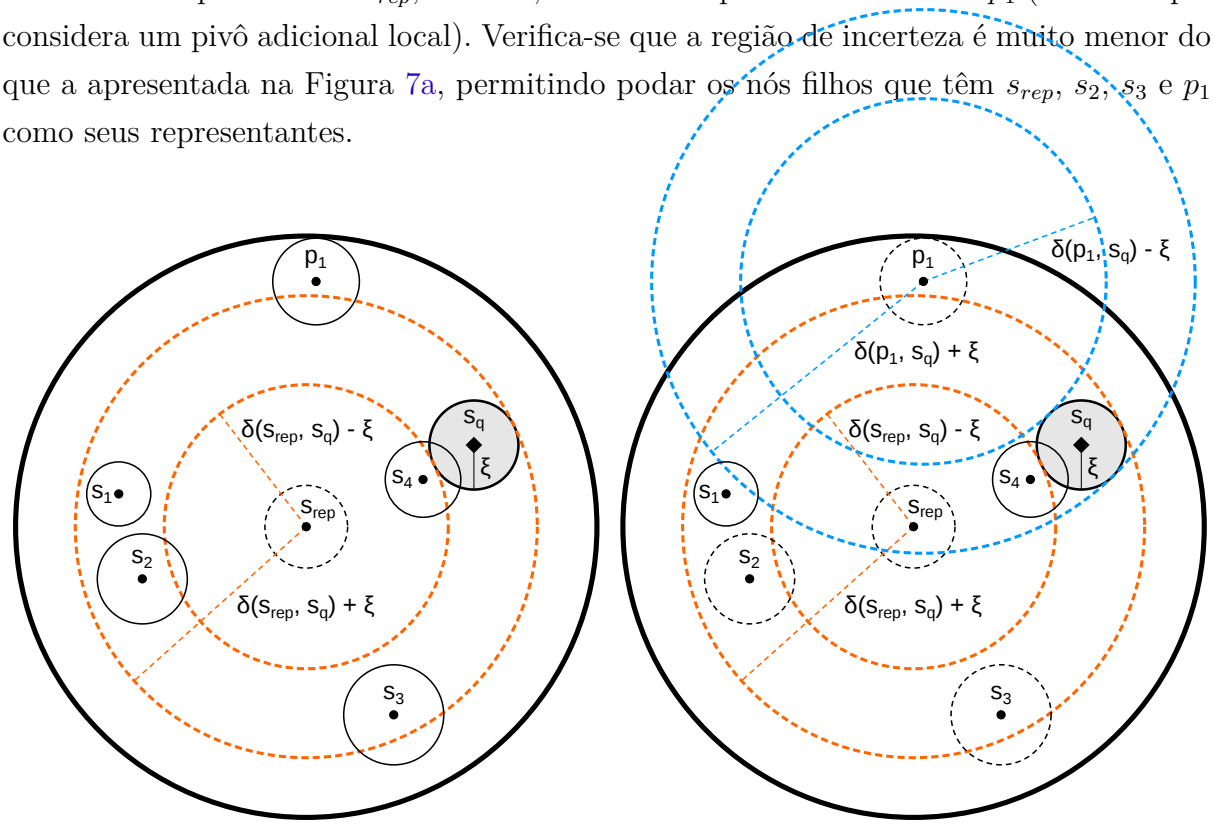
A Figura 7a ilustra o uso dessa propriedade, em que o nó filho cujo representante também é s_{rep} (a circunferência pontilhada preta, centrada em s_{rep}) é podado, uma vez que ele não toca a região de incerteza (o anel pontilhado alaranjado). O que a técnica CLAP permite é estender o uso dessa propriedade para todos os pivôs adicionais locais. Portanto, para todo elemento s_i que não foi podado por desigualdade triangular envolvendo o pivô principal, cada pivô adicional p_j, $1 \leq j \leq n$, onde n é o número de pivôs adicionais, é usado para descartar os elementos que satisfazem a uma das condições adicionais representadas pelas Inequações 3.3 e 3.4:

$$\delta(p_j, s_i) + r_i < \delta(p_j, s_q) - \xi \tag{3.3}$$

$$\delta(p_j, s_i) - r_i > \delta(p_j, s_q) + \xi \tag{3.4}$$

Similarmente às Inequações 3.1 e 3.2, essas novas condições envolvem dois cálculos de distância. Um deles, o valor $\delta(p_j, s_i)$, é calculado e armazenado no nó quando o MAM

é construído. O valor $\delta(p_j, s_q)$, embora deva ser calculado para cada pivô p_j quando o nó é visitado em uma consulta, permite reduzir ainda mais o número de cálculos de distância. A Figura 7b ilustra a nova região de incerteza como a intersecção de dois anéis: o alaranjado, centrado no representante s_{rep}, e o azul, centrado no pivô adicional local p_1 (este exemplo considera um pivô adicional local). Verifica-se que a região de incerteza é muito menor do que a apresentada na Figura 7a, permitindo podar os nós filhos que têm s_{rep}, s_2, s_3 e p_1 como seus representantes.



(a) Pelo representante.

(b) Pelo representante mais um pivô adicional.

Figura 7 – Poda por desigualdade triangular com um pivô adicional. Circunferências pontilhadas centradas em s_{rep}, s_2, s_3 e p_1 representam elementos descartados.

3.2 Antecipação de informações

Esta seção apresenta uma nova abordagem que visa evitar acessos a disco desnecessários modificando-se a estrutura dos nós em MAMs a fim de obter, antecipadamente, informações de nós filhos que somente seriam acessadas quando tais nós fossem lidos do disco. Considerando o MAM Slim-tree como exemplo, quando um nó é visitado durante uma consulta, o primeiro passo é utilizar a desigualdade triangular como mecanismo de poda com base no elemento representante. Para todo elemento s_i não descartado através desse passo em um nó índice, o nó correspondente a s_i deve ser acessado em disco mesmo se nenhum de seus elementos tocar a região de busca. No entanto, se determinadas informações dos nós filhos estivessem disponíveis no nó pai, como valores de distância e raios de cobertura, seria possível antecipar o mecanismo de poda por desigualdade triangular. Isso permitiria descartar elementos de nós filhos antes mesmo de acessá-los em disco. Em situações onde todos os elementos de um nó filho fossem descartados, por meio do uso

antecipado do mecanismo de poda por desigualdade triangular, esse nó filho poderia ser podado com segurança, mesmo se ele tocasse a região de busca com seu raio de cobertura.

Nesse contexto, são propostas duas técnicas: (i) ACIR (*Anticipation of Child Information regarding Representatives*), que visa antecipar informações referentes ao elemento representante de cada nó filho, e (ii) ACIR+ACIP (ACIR + *Anticipation of Child Information regarding Pivots*), que visa antecipar informações referentes aos pivôs adicionais de cada nó filho. Ambas as técnicas são apresentadas nas Seções 3.2.1 e 3.2.2.

3.2.1 A técnica ACIR

A técnica ACIR consiste em antecipar, para cada nó filho s_i do nó atual centrado em s_{rep}, o vetor de distâncias de seu representante s_i mais o vetor de raios de cobertura (somente as distâncias quando os nós filhos forem do tipo folha, uma vez que suas entradas não possuem raio). Consequentemente, o mecanismo de poda por desigualdade triangular considerando o representante é antecipado para cada nó filho que intercepta a região de busca em uma consulta. Com a técnica ACIR, a sequência de passos para cada nó acessado durante uma consulta por similaridade é:

1. Utilizar o mecanismo de poda por desigualdade triangular pelos pivôs adicionais, definidos com a técnica CLAP, nas entradas do nó atual não podadas por antecipação em seu nó pai (este passo não é realizado no nó raiz);
2. Calcular as distâncias entre o elemento de consulta s_q e cada elemento não podado no passo anterior;
3. Utilizar o mecanismo de poda por desigualdade triangular por representante, antecipadamente, nas entradas de cada nó filho interceptando a região de busca;
4. Ler do disco cada nó filho que tiver entradas não podadas no passo anterior.

Com o uso antecipado da desigualdade triangular pelo representante nas entradas dos nós filhos, é possível evitar acessos a disco desnecessários ao identificar nós que interceptam a região de busca e, apesar disso, são irrelevantes para o resultado da consulta, uma vez que nenhum de seus elementos intercepta a região de incerteza.

A Figura 8 ilustra um exemplo de situação em que um acesso a disco desnecessário é evitado através da técnica ACIR. Na figura, o nó filho centrado em s_1 intercepta a região de busca definida a partir de s_q, mas nenhum de seus nós filhos — s_{11}, s_{12}, s_{13} e s_{14}, que são netos do nó atual (o maior, centrado em s_{rep}) — intercepta a região de incerteza representada pelo anel pontilhado alaranjado dentro do nó filho centrado em s_1. Essa avaliação acerca dos nós netos, que determina se eles interceptam ou não a região de incerteza, pode acontecer no nível atual somente porque suas distâncias e raios foram antecipados. Se não

tivessem sido, o nó filho centrado em s_1 precisaria ser lido do disco (desnecessariamente) para que essa avaliação pudesse acontecer. Ainda na Figura 8, as pequenas circunferências centradas em s_{11}, s_{12}, s_{13} e s_{14} representam os raios de cobertura dos respectivos nós netos. Essas regiões aparecem na figura apenas para ilustrar o uso antecipado da desigualdade triangular que a técnica permite. Elas não são conhecidas quando o nó atual (centrado em s_{rep}) é processado, uma vez que os vetores de características dos representantes dos nós netos não são armazenados no nó atual.

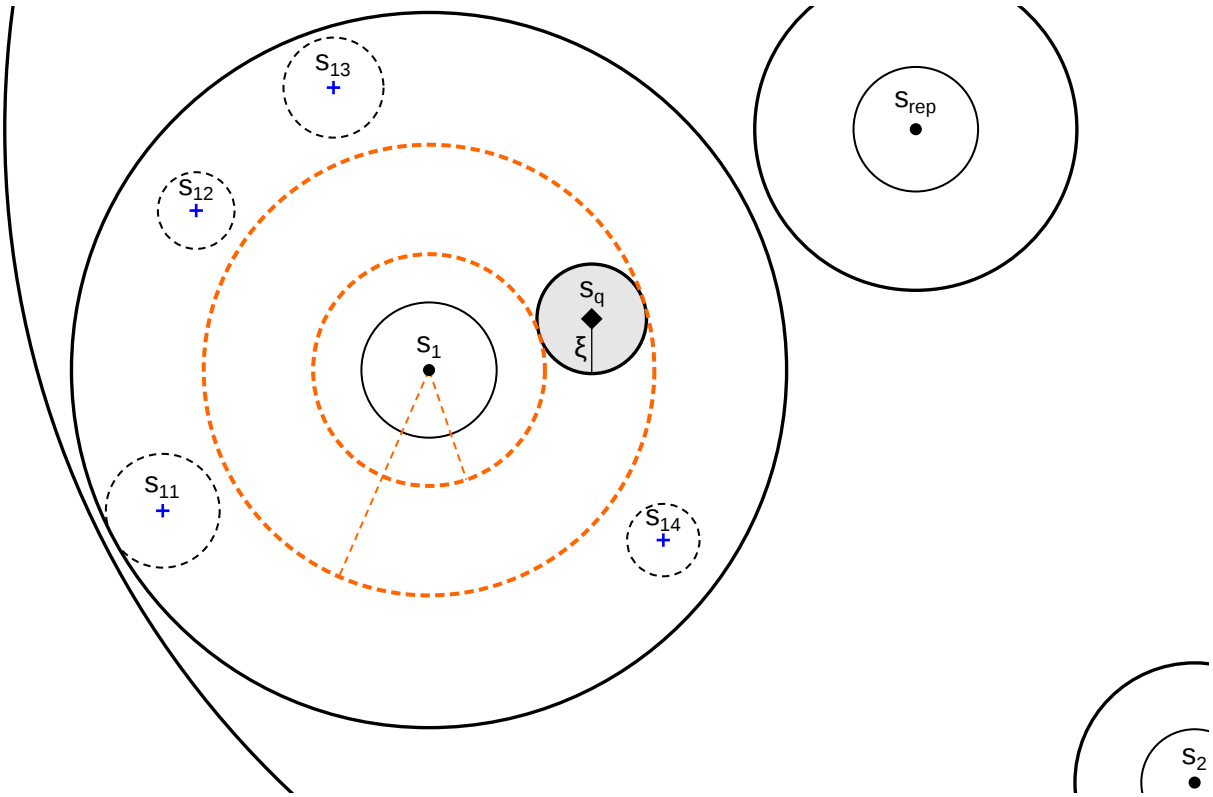


Figura 8 – Evitando um acesso a disco através da técnica ACIR. Nenhum dos nós netos (s_{11}, s_{12}, s_{13}, s_{14} e o próprio representante s_1, que são filhos do nó filho centrado em s_1) toca a região de incerteza, representada pelo anel alaranjado.

3.2.2 A técnica ACIR+ACIP

A técnica ACIR+ACIP consiste em antecipar, para cada nó filho s_i do nó atual centrado em s_{rep}, os vetores de distâncias referentes ao representante s_i e ao pivô adicional local p_i, além do vetor de raios de cobertura no caso de nós do tipo índice. Além disso, para que os valores de distância referentes aos pivôs adicionais possam ser utilizados, antecipa-se uma cópia do vetor de características de p_i, assim como já acontece originalmente com o vetor de características de cada s_i. Diante disso, a técnica ACIR+ACIP define o critério de incluir somente um pivô adicional por meio da técnica CLAP, para que sua antecipação resulte em um *overhead* controlado de informações em nós do tipo índice.

Consequentemente, para cada nó filho que intercepta a região de busca em uma consulta, antecipa-se o mecanismo de poda por desigualdade triangular tanto por seu representante quanto por seu pivô adicional. Com a técnica ACIR+ACIP, a sequência de passos para cada nó acessado durante uma consulta por similaridade é:

1. Calcular as distâncias entre o elemento de consulta s_q e as entradas do nó atual não podadas por antecipação em seu nó pai (este passo não é realizado no nó raiz);
2. Utilizar, antecipadamente, o mecanismo de poda por desigualdade triangular pelo representante nas entradas de cada nó filho não podado no passo anterior;
3. Utilizar, antecipadamente, o mecanismo de poda por desigualdade triangular pelo pivô adicional nas entradas de cada nó filho não podadas no passo anterior;
4. Ler do disco cada nó filho que tiver entradas não podadas no passo anterior.

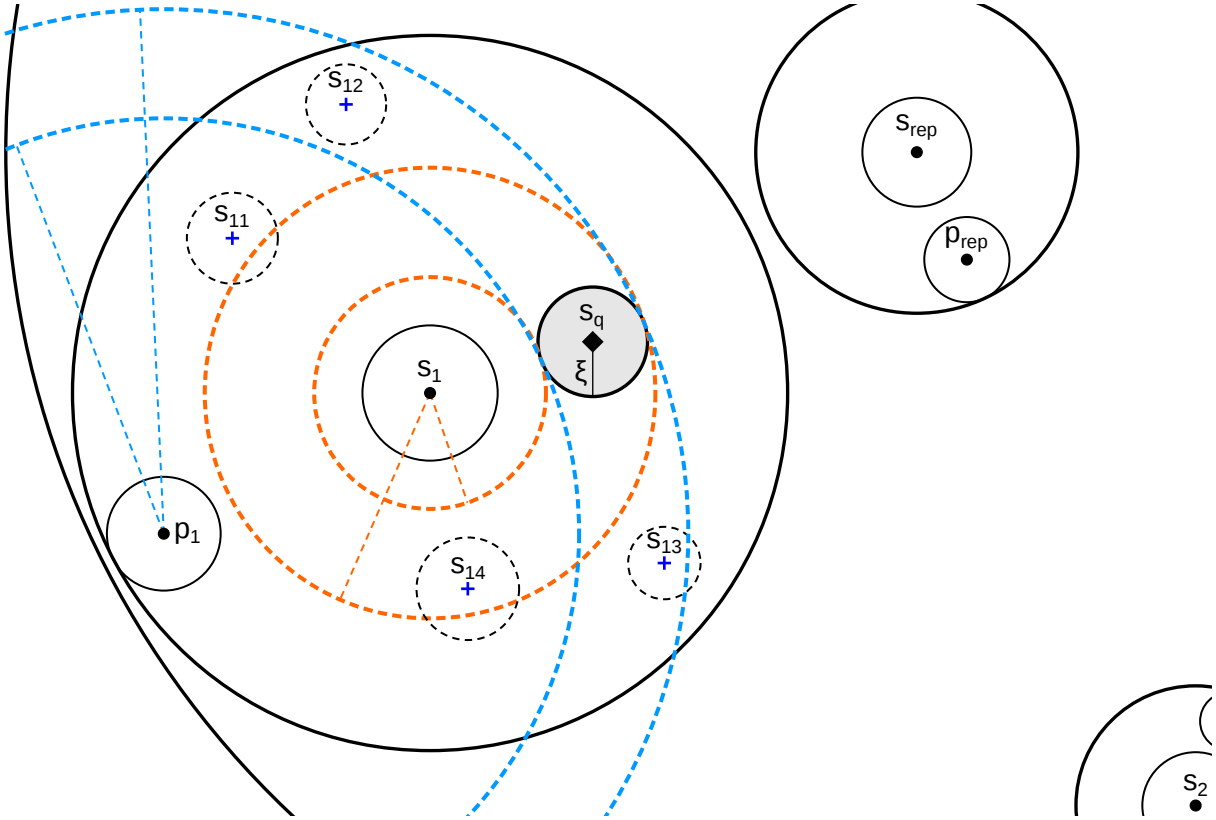


Figura 9 – Evitando um acesso a disco através da técnica ACIR+ACIP. Nenhum dos nós netos (s_{11}, s_{12}, s_{13}, s_{14}, o representante s_1 e o pivô p_1, que são filhos do nó filho centrado em s_1) toca a região de incerteza, representada pela intersecção do anel alaranjado com o azul.

A Figura 9 ilustra um exemplo de situação em que um acesso a disco desnecessário é evitado através da técnica ACIR+ACIP. Na figura, o nó filho centrado em s_1 intercepta a região de busca definida a partir de s_q, mas nenhum de seus nós filhos — s_{11}, s_{12}, s_{13},

s_{14} e p_1, que são netos do nó atual (o maior, centrado em s_{rep}) — intercepta a região de incerteza, representada pela intersecção do anel alaranjado com o azul, dentro do nó filho centrado em s_1. Essa avaliação acerca dos nós netos, que determina se eles interceptam ou não a região de incerteza, pode acontecer no nível atual somente porque suas distâncias e raios foram antecipados. Se não tivessem sido, o nó filho centrado em s_1 seria lido do disco (desnecessariamente) para que essa avaliação pudesse acontecer.

3.3 Aplicação das contribuições à Slim-tree

A Slim-tree possui dois tipos de nó: nós índice e nós folha. Um nó folha apresenta a seguinte estrutura (os caracteres $\langle$ e $\rangle$ delimitam um vetor):

$$leaf\ node\ [\langle OID_i, s_i, \delta(s_i, s_{rep})\rangle]$$

onde OID_i é o identificador do elemento; s_i é o próprio elemento, armazenado como um vetor de características, e $\delta(s_i, s_{rep})$ é a distância entre s_i e o representante. Um nó índice apresenta a seguinte estrutura:

$$index\ node\ [\langle s_i, r_i, \delta(s_i, s_{rep}), Ptr(T_{s_i}), \#Ent(T_{s_i})\rangle]$$

onde s_i é o vetor de características do elemento representante da subárvore T_{s_i}, apontada por $Ptr(T_{s_i})$; r_i é o raio de cobertura dessa subárvore, determinado pela distância de s_i ao elemento mais distante no nó da subárvore; $\delta(s_i, s_{rep})$ é a distância de s_i ao representante do nó atual e, por fim, $\#Ent(T_{s_i})$ é o número de entradas em T_{s_i}.

Ao aplicar as técnicas CLAP, ACIR e ACIR+ACIP à Slim-tree, os nós índice e os nós folha passam a ter novas estruturas, conforme apresentadas nas Seções 3.3.1 e 3.3.2.

3.3.1 Aplicação das técnicas CLAP e ACIR

Nas definições a seguir, símbolos em negrito representam as informações adicionadas pela técnica CLAP, enquanto os símbolos em azul representam as mudanças promovidas pela técnica ACIR. Um nó folha apresenta a seguinte estrutura:

$$leaf\ node\ [\langle \boldsymbol{Pos_j}\rangle, \langle OID_i, s_i, \langle \boldsymbol{\delta(s_i, p_j)}\rangle\rangle]$$

onde Pos_j é a posição do pivô adicional p_j (e.g. Pos_j vale 2 se p_j for o segundo elemento armazenado no nó), $\delta(s_i, p_j)$ é a distância de s_i ao pivô p_j e o restante das informações é como na estrutura original. Em relação à técnica ACIR, como os valores de distância ao representante são antecipados um nível acima no MAM, estes são removidos do nó folha.

O critério de escolha dos pivôs adicionais locais utilizado é a maior soma de distâncias até os pivôs anteriores. No caso do primeiro pivô adicional, este é o elemento que tem a maior distância até o pivô principal s_{rep}; o segundo pivô adicional é o elemento que tem a maior soma da distância até s_{rep} mais a distância até p_1 e assim por diante.

Os nós índice, por sua vez, são divididos em dois novos tipos: nós índice que são pais de nós folha, do tipo *l-index node*, e nós índice que são pais de nós índice, do tipo *i-index node*. Suas estruturas são as seguintes:

$$\textit{l-index node}\ [\langle \boldsymbol{Pos_j} \rangle, \langle s_i, \langle \boldsymbol{\delta(s_{il}, s_i)} \rangle, Ptr(T_{s_i}), \#Ent(T_{s_i}), \langle \boldsymbol{\delta(s_i, p_j)} \rangle \rangle]$$

$$\textit{i-index node}\ [\langle \boldsymbol{Pos_j} \rangle, \langle s_i, \langle \boldsymbol{\delta(s_{il}, s_i)} \rangle, \langle \boldsymbol{r_{il}} \rangle, Ptr(T_{s_i}), \#Ent(T_{s_i}), \langle \boldsymbol{\delta(s_i, p_j)} \rangle \rangle]$$

A diferença entre esses tipos de nó é que, em um *l-index node*, somente o vetor de distâncias $\langle \delta(s_{il}, s_i) \rangle$ são adicionados, onde s_{il} é a l-ésima entrada do i-ésimo nó filho e s_i é seu representante. Já em um *i-index node*, o raio de cobertura r_{il} também é adicionado. Em ambos os tipos de nó índice, a distância até s_{rep} e o raio de cada entrada, presentes na Slim-tree original, são removidos ao aplicar a técnica ACIR devido à sua antecipação um nível acima na estrutura. Como cada nó índice já armazena distâncias e raios referentes aos nós filhos, esses são os valores utilizados na consulta quando o nó é acessado.

3.3.2 Aplicação das técnicas CLAP e ACIR+ACIP

Idem à seção anterior, símbolos em negrito representam as informações adicionadas pela técnica CLAP (porém antecipadas pela técnica ACIR+ACIP), enquanto os símbolos em azul representam as informações adicionadas pela técnica ACIR+ACIP. Um nó folha apresenta a seguinte estrutura:

$$\textit{leaf node}\ [\langle OID_i, s_i \rangle]$$

onde OID_i é o identificador do elemento e s_i é seu vetor de características. O restante das informações, presentes na Slim-tree original e/ou em sua extensão com as técnicas CLAP e ACIR, deixam de estar presentes no nó folha quando a técnica ACIR+ACIP é aplicada, uma vez que elas são antecipadas um nível acima na estrutura.

Os nós índice são divididos em dois tipos, como na técnica ACIR. Suas estruturas são as seguintes:

$$\textit{l-index node}\ [\langle s_i, \boldsymbol{p_i}, \langle \boldsymbol{\delta(s_{il}, s_i)} \rangle, \langle \boldsymbol{\delta(s_{il}, p_i)} \rangle, Ptr(T_{s_i}), \#Ent(T_{s_i}) \rangle]$$

$$\textit{i-index node}\ [\langle s_i, \boldsymbol{p_i}, \langle \boldsymbol{\delta(s_{il}, s_i)} \rangle, \langle \boldsymbol{r_{il}} \rangle, \langle \boldsymbol{\delta(s_{il}, p_i)} \rangle, Ptr(T_{s_i}), \#Ent(T_{s_i}) \rangle]$$

onde p_i é uma cópia do vetor de características do pivô adicional local do i-ésimo nó filho (definido pela técnica CLAP), $\langle \delta(s_{il}, p_i) \rangle$ é o vetor antecipado de distâncias em relação ao pivô p_i e o restante das informações é como na técnica ACIR.

3.4 Descrição dos algoritmos

Esta seção apresenta os algoritmos de inserção e das consultas referentes às técnicas propostas. Embora o pseudocódigo seja baseado na técnica ACIR+ACIP, são explicitadas as diferenças quanto à técnica ACIR. A Seção 3.4.1 contém os Algoritmos 1 e 2, que descrevem a inserção de um elemento na estrutura. Já a Seção 3.4.2 contém os Algoritmos 3 e 4, que descrevem a Range query, e os Algoritmos 5 e 6, que descrevem a k-NN query.

3.4.1 Algoritmos de construção

Algoritmo 1 Inserção de um elemento

```
1:  function ADD(featureVector)
2:      if GetRootID() == 0 then
3:          rootNode = new LeafNode()
4:          rootNode.AddEntry(featureVector)
5:      else
6:          rootNode = GetRootNode()
7:          childInformation = rootNode.GetChildInformation()
8:          result = InsertRecursively(GetRootID(), featureVector, childInformation)
9:          if result.status == "SPLIT" then
10:             AddNewRoot(result.dataFromSplit)
11:         end if
12:     end if
13: end function
```

No Algoritmo 1, as Linhas 3–4 criam a raiz pela primeira vez e inserem o elemento. Nas Linhas 6–11, o algoritmo traz o nó raiz para a memória principal, obtém suas informações dos nós filhos, que compreendem os valores de distância e os raios de cobertura antecipados, e chama o método que percorre a estrutura recursivamente até o nível folha, a fim de inserir o elemento. Se o retorno do método for "SPLIT", cria-se uma nova raiz.

No Algoritmo 2, a Linha 8 chama o método InsertRecursively até que o nó trazido para a memória principal seja do tipo folha. Quando o nó é do tipo folha, as Linhas 56–63 inserem o elemento caso haja espaço livre no nó. Caso contrário, o nó sofre *split*. Quando o nó é do tipo índice, após o retorno da chamada recursiva do método InsertRecursively, o algoritmo executa diferentes passos dependendo do estado retornado, que pode ser "NO ACTION", "REPRESENTATIVE CHANGE" ou "SPLIT", visando a manutenção do nó índice atual, como a propagação de novas informações dos nós filhos ou a atualização do raio de cobertura.

As Linhas 9–12, referentes ao estado "NO ACTION", atualizam o pivô antecipado, caso este tenha sido alterado no nó filho.

Algoritmo 2 Trajeto da raiz ao nível folha durante inserção

```
1: function InsertRecursively(pageID, featureVector, parentInformation)
2:     node = CreateNode(pageID)
3:     if node.GetNodeType() == "INDEX" then
4:         indexNode = (IndexNode)(node)
5:         subtree = ChooseSubtree(indexNode, featureVector, parentInformation)
6:         subtreePageID = indexNode.GetPageID(subtree)
7:         childInformation = indexNode.GetChildInformation()
8:         result = InsertRecursively(subtreePageID, featureVector, childInformation)
9:         if result.status == "NO ACTION" then
10:            if result.pivot ≠ null then
11:                indexNode.ReplaceEntryPivot(subtree, result.pivot)
12:            end if
13:        else if result.status == "REPRESENTATIVE CHANGE" then
14:            indexNode.RemoveEntry(subtree)
15:            index = indexNode.AddEntry(result.representative)
16:            indexNode.AddEntryPivot(index, result.pivot)
17:            if it is not the root level then
18:                if subtree was the representative then
19:                    UpdateDistances(indexNode, index, parentInformation)
20:                else
21:                    result.status = "NO ACTION"
22:                end if
23:            end if
24:        else if result.status == "SPLIT" then
25:            if result.dataFromSplit.current.representative == null then
26:                if result.dataFromSplit.current.pivot ≠ null then
27:                    indexNode.ReplaceEntryPivot(subtree, result.dataFromSplit.current.pivot)
28:                end if
29:                if indexNode.ObjectFits(result.dataFromSplit.new) then
30:                    index = indexNode.AddEntry(result.dataFromSplit.new.representative)
31:                    indexNode.AddEntryPivot(index, result.dataFromSplit.new.pivot)
32:                else
33:                    result.dataFromSplit = indexNode.Split(result.dataFromSplit.new)
34:                end if
35:            else
36:                indexNode.RemoveEntry(subtree)
37:                index = indexNode.AddEntry(result.dataFromSplit.current.representative)
38:                indexNode.AddEntryPivot(index, result.dataFromSplit.current.pivot)
39:                if it is not the root level then
40:                    if subtree was the representative then
41:                        UpdateDistances(indexNode, index, parentInformation)
42:                        result.status = "REPRESENTATIVE CHANGE"
43:                    else
44:                        result.status = "NO ACTION"
45:                    end if
46:                end if
47:                if indexNode.ObjectFits(result.dataFromSplit.new) then
48:                    index = indexNode.AddEntry(result.dataFromSplit.new.representative)
49:                    indexNode.AddEntryPivot(index, result.dataFromSplit.new.pivot)
50:                else
51:                    result.dataFromSplit = indexNode.Split(result.dataFromSplit.new)
52:                end if
53:            end if
54:        end if
55:    else
56:        leafNode = (LeafNode)(node)
57:        if leafNode.ObjectFits(featureVector) then
58:            index = leafNode.AddEntry(featureVector)
59:            result.status = "NO ACTION"
60:        else
61:            result.dataFromSplit = leafNode.Split(featureVector)
62:            result.status = "SPLIT"
63:        end if
64:    end if
65:    Update distances and radii within parentInformation
66:    Check whether the current node had its additional pivot modified and propagate any changes
67:    return result
68: end function
```

As Linhas 14–23, relacionadas ao estado “REPRESENTATIVE CHANGE”, removem a entrada referente ao representante anterior, adicionam a entrada referente ao novo representante, juntamente ao pivô antecipado dessa entrada, e atualizam as distâncias ao novo representante caso não seja o nível raiz.

As Linhas 25–53, relacionadas ao estado “SPLIT”, reagem ao *split* do nó filho de acordo com duas possibilidades:

1. O nó filho que sofreu *split* não teve seu representante mudado. Nesse caso, atualiza-se o pivô antecipado se este tiver sido alterado no nó filho e tenta-se adicionar uma nova entrada, que representa o novo nó filho que surgiu a partir do processo de *split*. Caso não haja espaço livre para a nova entrada, o nó atual sofre *split*.

2. O nó filho que sofreu *split* teve seu representante mudado. Nesse caso, remove-se a entrada referente ao representante anterior e adiciona-se a entrada referente ao novo representante, da mesma maneira como ocorre nas Linhas 14–23. Em seguida, tenta-se adicionar a entrada que representa o novo nó filho que surgiu com o processo de *split*. Se não houver espaço livre para a nova entrada, o nó atual sofre *split*.

Em ambas as técnicas de antecipação, juntamente à técnica CLAP, existem instruções nos algoritmos de construção relacionadas à manutenção das informações trazidas por essas contribuições, o que implica um custo computacional de construção mais elevado se comparado ao custo do MAM original.

Considerando a técnica CLAP, quando um elemento é inserido em um nó, é preciso checar se esse nó terá seu pivô adicional alterado após a inserção. Isso requer a verificação da distância de cada elemento ao representante do nó, a fim de encontrar o elemento que possui o maior valor de distância. Caso o maior valor de distância pertença ao elemento recém-inserido, este passará a ser o novo pivô adicional do nó. Com isso, deve-se recalcular a distância de cada elemento do nó ao pivô adicional, o que implica n cálculos de distância a mais, onde n é o número de entradas no nó. Esses passos são sumarizados na Linha 66.

Considerando as técnicas de antecipação, nenhum cálculo de distância é realizado a mais. Toda informação antecipada é propagada um nível acima por meio de parâmetros do método InsertRecursively. Ao retornar de uma chamada recursiva no processo de inserção, na técnica ACIR+ACIP, pode ser necessário atualizar o pivô antecipado, que consiste na cópia de um vetor de características em memória.

3.4.2 Algoritmos de consulta

Nos Algoritmos 3 e 4, a primeira instrução diz respeito à instanciação em memória do nó atual, que é o nó raiz no Algoritmo 3 ou o nó cuja página é identificada por pageID no Algoritmo 4. Em seguida, executa-se diversas instruções dependendo do tipo do nó.

Algoritmo 3 Range query — Nível raiz — CLAP e ACIR+ACIP

```
1: function CallRangeQuery(sample, range)
2:     rootNode = GetRootNode()
3:     if rootNode.GetNodeType() == "INDEX" then
4:         indexNode = (IndexNode)(rootNode)
5:         childRadii = indexNode.GetChildRadii()
6:         childDistances = indexNode.GetChildDistances()
7:         childPivotDistances = indexNode.GetChildPivotDistances()
8:         childIndex = 0
9:         for i = 0; i < indexNode.GetNumberOfEntries(); i++ do
10:            object = indexNode.GetObject(i)
11:            distance = GetDistance(object, sample)
12:            if childRadii ≠ null then
13:                for j = 0; j < indexNode.GetNumberOfChildren(i); j++ do
14:                    if |distance - childDistances[i][j]| ≤ range + childRadii[i][j] then
15:                        notPrunedEntries.Add(j)
16:                    end if
17:                end for
18:            else
19:                for j = 0; j < indexNode.GetNumberOfChildren(i); j++ do
20:                    if |distance - childDistances[i][j]| ≤ range then
21:                        notPrunedEntries.Add(j)
22:                    end if
23:                end for
24:            end if
25:            if notPrunedEntries.NotEmpty() then
26:                it = notPrunedEntries.Begin()
27:                pivot = indexNode.GetEntryPivot(i)
28:                distance = GetDistance(pivot, sample)
29:                if childRadii ≠ null then
30:                    while it ≠ notPrunedEntries.End() do
31:                        if |distance - childPivotDistances[i][it]| > range + childRadii[i][it] then
32:                            notPrunedEntries.Erase(it)
33:                        end if
34:                        it++
35:                    end while
36:                else
37:                    while it ≠ notPrunedEntries.End() do
38:                        if |distance - childPivotDistances[i][it]| > range then
39:                            notPrunedEntries.Erase(it)
40:                        end if
41:                        it++
42:                    end while
43:                end if
44:                if notPrunedEntries.NotEmpty() then
45:                    pageID = indexNode.GetPageID(i)
46:                    RangeQuery(sample, range, result, pageID, childRadii, childIndex, notPrunedEntries)
47:                    notPrunedEntries.Clear()
48:                end if
49:            end if
50:            childIndex += indexNode.GetNumberOfChildren(i)
51:        end for
52:    else
53:        leafNode = (LeafNode)(rootNode)
54:        for i = 0; i < leafNode.GetNumberOfEntries(); i++ do
55:            object = leafNode.GetObject(i)
56:            distance = GetDistance(object, sample)
57:            if distance ≤ range then
58:                result.AddPair(object, distance)
59:            end if
60:        end for
61:    end if
62:    return result
63: end function
```

Algoritmo 4 Range query — Demais níveis — CLAP e ACIR+ACIP

```
1: function RangeQuery(sample, range, result, pageID, radii, parentIndex, entriesToCheck)
2:     node = CreateNode(pageID)
3:     if node.GetNodeType() == "INDEX" then
4:         indexNode = (IndexNode)(node)
5:         childRadii = indexNode.GetChildRadii()
6:         childDistances = indexNode.GetChildDistances()
7:         childPivotDistances = indexNode.GetChildPivotDistances()
8:         childIndex = 0
9:         for i = entriesToCheck.Begin(); i ≠ entriesToCheck.End(); i++ do
10:            object = indexNode.GetObject(i)
11:            distance = GetDistance(object, sample)
12:            if distance ≤ range + radii[parentIndex + i] then
13:                if childRadii ≠ null then
14:                    for j = 0; j < indexNode.GetNumberOfChildren(i); j++ do
15:                        if |distance - childDistances[i][j]| ≤ range + childRadii[i][j] then
16:                            notPrunedEntries.Add(j)
17:                        end if
18:                    end for
19:                else
20:                    for j = 0; j < indexNode.GetNumberOfChildren(i); j++ do
21:                        if |distance - childDistances[i][j]| ≤ range then
22:                            notPrunedEntries.Add(j)
23:                        end if
24:                    end for
25:                end if
26:                if notPrunedEntries.NotEmpty() then
27:                    it = notPrunedEntries.Begin()
28:                    pivot = indexNode.GetEntryPivot(i)
29:                    distance = GetDistance(pivot, sample)
30:                    if childRadii ≠ null then
31:                        while it ≠ notPrunedEntries.End() do
32:                            if |distance - childPivotDistances[i][it]| > range + childRadii[i][it] then
33:                                notPrunedEntries.Erase(it)
34:                            end if
35:                            it++
36:                        end while
37:                    else
38:                        while it ≠ notPrunedEntries.End() do
39:                            if |distance - childPivotDistances[i][it]| > range then
40:                                notPrunedEntries.Erase(it)
41:                            end if
42:                            it++
43:                        end while
44:                    end if
45:                    if notPrunedEntries.NotEmpty() then
46:                        pageID = indexNode.GetPageID(i)
47:                        RangeQuery(sample, range, result, pageID, childRadii, childIndex, notPrunedEntries)
48:                        notPrunedEntries.Clear()
49:                    end if
50:                end if
51:            end if
52:            childIndex += indexNode.GetNumberOfChildren(i)
53:        end for
54:    else
55:        leafNode = (LeafNode)(node)
56:        for i = entriesToCheck.Begin(); i ≠ entriesToCheck.End(); i++ do
57:            object = leafNode.GetObject(i)
58:            distance = GetDistance(object, sample)
59:            if distance ≤ range then
60:                result.AddPair(object, distance)
61:            end if
62:        end for
63:    end if
64:    return result
65: end function
```

Caso o nó seja do tipo folha, calcula-se a distância de suas entradas até o elemento de consulta *sample* e verifica-se se a distância é menor do que o raio de abrangência *range*. As entradas cuja distância for menor são adicionadas como resposta à consulta. Se o nó folha for a raiz, todas as suas entradas são checadas. Caso contrário, verifica-se somente as entradas da lista *entriesToCheck*, criada no nó pai e passada como parâmetro para o nó atual, que são as entradas não podadas por antecipação no nó pai.

No caso de o nó ser do tipo índice, esta seção descreve o que cada bloco de linhas faz no Algoritmo 4, que diz respeito aos demais níveis, e mostra as diferenças em relação ao Algoritmo 3, que diz respeito ao nível raiz.

Nas Linhas 4–8, o algoritmo obtém as informações dos nós filhos. Em seguida, para cada entrada da lista entriesToCheck, calcula-se sua distância até o elemento de consulta e verifica-se a existência de intersecção com a região de busca. Note-se que, no Algoritmo 3, todas as entradas do nó são checadas ao invés da lista entriesToCheck. Além disso, após o cálculo de distância, não é verificada a existência de intersecção com a região de busca, visto que o nó raiz não possui os valores de raio de cobertura de suas entradas.

Nas Linhas 13–25, tenta-se podar os nós filhos da i-ésima entrada por desigualdade triangular. Cada nó filho não podado tem sua posição adicionada à lista *notPrunedEntries*. Note-se que o algoritmo verifica se os nós filhos são do tipo índice ou do tipo folha antes de tentar podá-los. Os nós filhos são do tipo índice quando *childRadii* não é *null*.

Se a lista notPrunedEntries não estiver vazia, então não foi possível podar a i-ésima entrada por antecipação considerando o elemento representante. Portanto, deve-se tentar realizar a poda por antecipação considerando o pivô antecipado da i-ésima entrada. Assim, após calcular a distância do pivô antecipado até o elemento de consulta, as Linhas 30–44 tentam podar a i-ésima entrada através da desigualdade triangular pelo pivô. Note-se que, na técnica ACIR, este passo é realizado assim que o nó atual é visitado, pelo fato de não ser possível fazê-lo por antecipação. Sempre que um nó filho for podado neste passo, ele é retirado da lista notPrunedEntries.

Por fim, se a lista notPrunedEntries não estiver vazia, as Linhas 45–49 chamam o método RangeQuery recursivamente, passando a lista notPrunedEntries como parâmetro, que levará o nome entriesToCheck dentro da chamada recursiva, juntamente ao restante dos parâmetros requeridos. Note-se também que a variável *childIndex*, que possui o nome *parentIndex* dentro da chamada recursiva e é utilizada para verificar se existe intersecção entre cada entrada e a região de busca, é atualizada na Linha 52.

O Algoritmo 5 inicia-se atribuindo ∞ a *range*, instanciando o nó raiz e formando a lista *entriesToCheck*. Visto que na k-NN query o raio de abrangência não é conhecido no início, a ideia é atribuir o valor ∞ e, ao longo da consulta, atualizar seu valor conforme elementos são incluídos como resposta.

Algoritmo 5 k-NN query — Parte 1 — CLAP e ACIR+ACIP

```
1: function KnnQuery(sample, k, result)
2:     range = ∞
3:     pageID = GetRootID()
4:     while pageID ≠ 0 do
5:         node = CreateNode(pageID)
6:         if it is the root node then
7:             for i = 0; i < node.GetNumberOfEntries(); i++ do
8:                 entriesToCheck.Add(i)
9:             end for
10:        end if
11:        if node.GetNodeType() == "INDEX" then
12:            indexNode = (IndexNode)(node)
13:            childRadii = indexNode.GetChildRadii()
14:            childDistances = indexNode.GetChildDistances()
15:            childPivotDistances = indexNode.GetChildPivotDistances()
16:            for i = entriesToCheck.Begin(); i ≠ entriesToCheck.End(); i++ do
17:                object = indexNode.GetObject(i)
18:                distance = GetDistance(object, sample)
19:                if it is the root node then
20:                    checkChildInformation = true
21:                else
22:                    if distance ≤ range + currentElement.childRadii[i] then
23:                        checkChildInformation = true
24:                    else
25:                        checkChildInformation = false
26:                    end if
27:                end if
28:                if checkChildInformation then
29:                    if childRadii ≠ null then
30:                        for j = 0; j < indexNode.GetNumberOfChildren(i); j++ do
31:                            if |distance - childDistances[i][j]| ≤ range + childRadii[i][j] then
32:                                notPrunedEntries.Add(j)
33:                            end if
34:                        end for
35:                    else
36:                        for j = 0; j < indexNode.GetNumberOfChildren(i); j++ do
37:                            if |distance - childDistances[i][j]| ≤ range then
38:                                notPrunedEntries.Add(j)
39:                            end if
40:                        end for
41:                    end if
42:                    if notPrunedEntries.NotEmpty() then
43:                        it = notPrunedEntries.Begin()
44:                        pivot = indexNode.GetEntryPivot(i)
45:                        distance = GetDistance(pivot, sample)
46:                        if childRadii ≠ null then
47:                            while it ≠ notPrunedEntries.End() do
48:                                if |distance - childPivotDistances[i][it]| > range + childRadii[i][it] then
49:                                    notPrunedEntries.Erase(it)
50:                                end if
51:                                it++
52:                            end while
53:                        else
54:                            while it ≠ notPrunedEntries.End() do
55:                                if |distance - childPivotDistances[i][it]| > range then
56:                                    notPrunedEntries.Erase(it)
57:                                end if
58:                                it++
59:                            end while
60:                        end if
61:                        if notPrunedEntries.NotEmpty() then
62:                            Insert this element along with its child information into the queue
63:                            notPrunedEntries.Clear()
64:                        end if
65:                    end if
66:                end if
67:            end for
```

Algoritmo 6 k-NN query — Parte 2 — CLAP e ACIR+ACIP

```
68:         else
69:             leafNode = (LeafNode)(node)
70:             for i = entriesToCheck.Begin(); i ≠ entriesToCheck.End(); i++ do
71:                 object = leafNode.GetObject(i)
72:                 distance = GetDistance(object, sample)
73:                 if distance ≤ range then
74:                     result.AddPair(object, distance)
75:                     if result.GetNumberOfEntries() ≥ k then
76:                         result.Cut(k)
77:                         range = result.GetMaximumDistance()
78:                     end if
79:                 end if
80:             end for
81:         end if
82:         while queue.Pop(currentElement) do
83:             Try to prune the element by using its child information with the potential new range value
84:             if the element was pruned then
85:                 Proceed to pop the next element from the queue
86:             else
87:                 pageID = currentElement.pageID
88:                 break
89:             end if
90:         end while
91:     end while
92: end function
```

Se o nó obtido for do tipo índice, procede-se de maneira semelhante aos algoritmos da Range query. Primeiramente, como o algoritmo da k-NN query não é recursivo, trata-se o nível raiz e os demais níveis no mesmo método, como pode ser visto nas Linhas 19–27. Em seguida, o processo de poda ocorre como nos algoritmos da Range query. A principal diferença é que, após todas as tentativas de poda por antecipação, ao invés de chamar o método KnnQuery recursivamente, insere-se a i-ésima entrada na fila de prioridade *queue* na Linha 62. Essa fila é ordenada decrescentemente pela distância da entrada ao elemento de consulta *sample*. As informações dos nós filhos também são incluídas em cada entrada da fila, para que possam ser usadas mais tarde.

No Algoritmo 6, que é a continuação do Algoritmo 5, os nós do tipo folha também são tratados de maneira semelhante aos algoritmos da Range query. A principal diferença é que, após um elemento ser incluído como resposta, verifica-se se a resposta possui mais de k elementos. Se possuir, mantém-se apenas os k mais próximos e atualiza-se o raio de abrangência, conforme as Linhas 75–78.

Por fim, ainda no Algoritmo 6, as Linhas 82–90 retiram o elemento do começo da fila, juntamente às informações referentes aos nós filhos, e tentam podá-lo novamente com o possível novo valor de *range*. Se o elemento for podado, retira-se o próximo elemento da fila e procede-se da mesma forma até que um elemento não seja podado.

Em comparação aos algoritmos de consulta da Slim-tree original, os algoritmos de consulta das técnicas desta dissertação possuem instruções adicionais, que são referentes ao processo de poda por desigualdade triangular pelos pivôs adicionais. Além disso, requer-se a manipulação de uma maior quantidade de variáveis e parâmetros, devido ao acesso

às informações antecipadas. No entanto, essas operações adicionais permitem um maior poder de poda, levando a um melhor desempenho em consultas.

3.5 Considerações finais

Este capítulo apresentou as contribuições desta dissertação, sua aplicação ao MAM Slim-tree e a descrição dos algoritmos de construção, bem como da Range query e da k-NN query. O próximo capítulo apresenta os experimentos e os resultados obtidos da avaliação das técnicas desenvolvidas neste trabalho.

4 EXPERIMENTOS E RESULTADOS

As técnicas apresentadas nesta dissertação foram submetidas a experimentos envolvendo conjuntos de dados com diferentes dimensionalidades e cardinalidades, empregando-se métricas com diferentes custos computacionais, de forma a possibilitar a avaliação das contribuições em cenários variados. Este capítulo apresenta os resultados obtidos por meio de combinações de quatro conjuntos de dados com três métricas. Para realização dos experimentos, foi utilizada uma máquina equipada com um processador Intel® Core™ i5-2400@3.1GHz, memória RAM de 4GB@1333MHz, HD 500GB SATA III 6Gb/s 7200RPM e sistema operacional Ubuntu Linux 14.04.1 LTS 64-bit.

As técnicas foram implementadas como extensões do MAM Slim-tree, desenvolvido em C++ com a biblioteca Arboretum[1]. Esta é uma biblioteca *open-source* que compreende métodos de acesso direcionados para dados complexos, desenvolvida e mantida pelo Grupo de Bases de Dados e Imagens do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (GBDI-ICMC-USP).

O desempenho das contribuições apresentadas foi comparado à Slim-tree original e à Slim-tree estendida com NN-graph, mecanismo de poda utilizado na M*-tree considerado o estado da arte de técnicas baseadas em múltiplos pivôs locais no momento da elaboração desta dissertação.

Este capítulo está organizado conforme a seguir: a Seção 4.1 descreve os conjuntos de dados utilizados nos experimentos; a Seção 4.2 apresenta os resultados relacionados ao desempenho das contribuições em consultas por similaridade; a Seção 4.3 analisa questões de construção das estruturas.

4.1 Descrição dos conjuntos de dados

Os conjuntos de dados ALOI-T e ALOI-H foram originados da biblioteca de imagens ALOI[2] (*Amsterdam Library of Object Images*) [49]. Ambos os conjuntos são formados por vetores de características extraídos de 108.000 imagens de objetos fotografados várias vezes, variando-se a posição, a iluminação e a combinação das cores. O conjunto ALOI-T é composto por vetores de características de textura Haralick [50, 51] de 140 dimensões, enquanto o conjunto ALOI-H é composto por histogramas de cores de 256 dimensões.

A coleção de imagens CoPhIR[3] [52] é formada por 106 milhões de imagens processa-

[1] Disponível em: <http://www.gbdi.icmc.usp.br/downloads/arboretum>
[2] Disponível em: <http://aloi.science.uva.nl>
[3] Disponível em: <http://cophir.isti.cnr.it>

das do Flickr. Para cada imagem, foram extraídas as características do padrão MPEG-7[4]: *Scalable Color*, *Color Structure*, *Color Layout*, *Edge Histogram* e *Homogeneous Texture*. O vetor de características completo de 282 dimensões foi utilizado para construir o conjunto CoPhIR-1M-WL2. O vetor de características de 64 dimensões que diz respeito à característica *Color Structure* foi utilizado para construir o conjunto CoPhIR-1M-M. Ambos os conjuntos possuem 1 milhão de elementos.

Como os conjuntos ALOI-H e CoPhIR-1M-M dizem respeito a histogramas de cores, foi utilizada sobre eles a métrica Mahalanobis. Sobre o conjunto ALOI-T foi utilizada a métrica L_2, que é a função euclidiana, e sobre o conjunto CoPhIR-1M-WL2 foi utilizada uma função euclidiana ponderada (*Weighted* L_2) com os pesos sugeridos em [53].

4.2 Desempenho em consultas por similaridade

Esta seção apresenta os resultados obtidos em experimentos de desempenho, por meio da execução de consultas aos k vizinhos mais próximos e consultas por abrangência, envolvendo as técnicas CLAP, ACIR, ACIR+ACIP e NN-graph sobre os quatro conjuntos de dados descritos na Seção 4.1 e variando-se k entre os valores 25, 100, 200 e 300. Para cada valor de k, as consultas foram executadas múltiplas vezes, cada vez com um elemento de consulta aleatório, obtendo-se o valor médio de cada atributo avaliado.

Resultados iniciais a respeito das técnicas CLAP e ACIR foram aceitos para publicação no ADBIS 2015 [54]. Esses resultados estão inclusos nos resultados apresentados nesta seção. Uma vez que a técnica CLAP é utilizada em conjunto com ambas as técnicas ACIR e ACIR+ACIP, o termo "CLAP" é omitido nas próximas seções a fim de simplificar a leitura.

4.2.1 Resultados sobre o conjunto ALOI-T

Para a realização dos experimentos sobre o conjunto ALOI-T, foram utilizados 500 elementos de consulta aleatórios. A Figura 10 mostra os resultados das consultas k-NN e a Figura 11 mostra os resultados das consultas por abrangência.

Nas consultas k-NN, o uso das técnicas ACIR e ACIR+ACIP possibilitou ganhos em todos os parâmetros avaliados. Considerando tempo de execução, as técnicas ACIR e ACIR+ACIP apresentaram, respectivamente, ganhos de 28.8% a 39.1% e de 22.6% a 32.1%. Considerando cálculos de distância, as técnicas ACIR, ACIR+ACIP e NN-graph apresentaram ganhos de 32.6% a 47.4%, de 30.8% a 45.2% e de 34.4% a 39%. Considerando acessos a disco, as técnicas ACIR e ACIR+ACIP apresentaram ganhos de 8.9% a 11.6% e de 18.7% a 23.8%. Apesar de apresentar ganho no número de cálculos de distância, a

[4] Disponível em: <http://mpeg.chiariglione.org/standards/mpeg-7>

técnica NN-graph teve pior desempenho em relação a tempo de execução e apresentou empate técnico com a Slim-tree em acessos a disco.

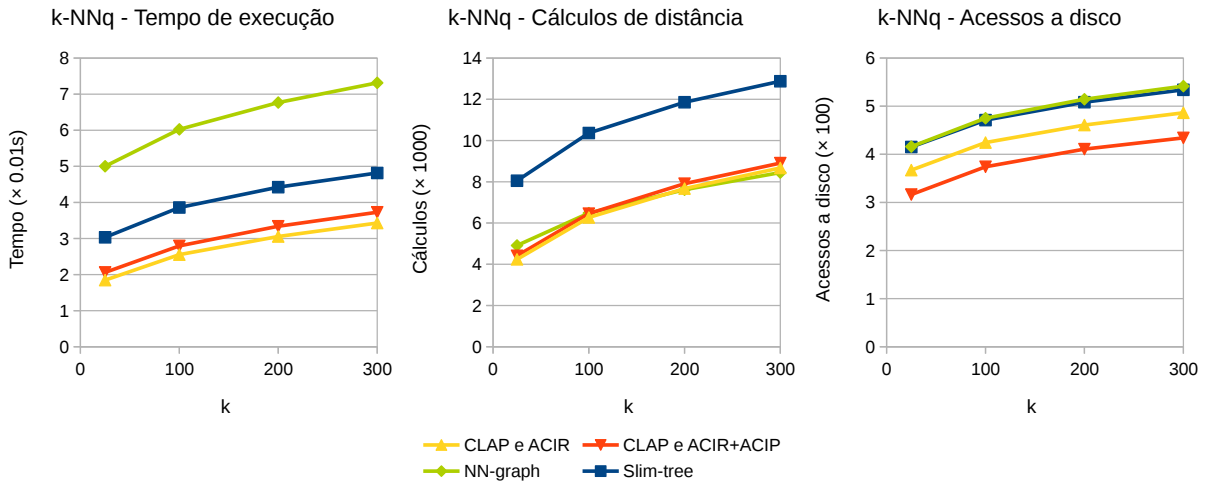


Figura 10 – Resultados de consultas k-NN no conjunto ALOI-T.

Nas consultas por abrangência, embora os valores absolutos sejam menores de um modo geral e os ganhos mais acentuados, o comportamento de todas as técnicas foi similar em relação às consultas k-NN. Considerando tempo de execução, as técnicas ACIR e ACIR+ACIP apresentaram ganhos de 42.5% a 52.8% e de 40.9% a 51.2%. Considerando cálculos de distância, as técnicas ACIR, ACIR+ACIP e NN-graph apresentaram ganhos de 40.1% a 53.5%, de 38.4% a 52.1% e de 39% a 40%. Considerando acessos a disco, as técnicas ACIR e ACIR+ACIP apresentaram ganhos de 9.9% a 12.4% e de 19.4% a 25.3%.

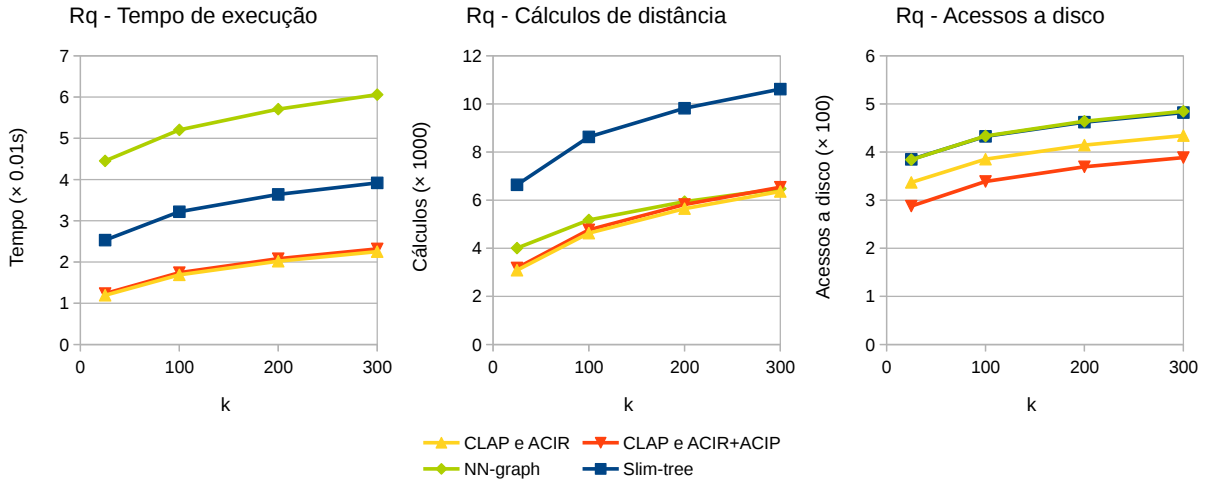


Figura 11 – Resultados de consultas por abrangência no conjunto ALOI-T.

O fato de a técnica NN-graph não reduzir acessos a disco somado ao aumento de informações a serem manipuladas na estrutura, que armazenam a posição do vizinho mais próximo e sua distância de cada elemento em um nó, fez com que a técnica NN-graph não apresentasse ganhos no tempo de execução. Além disso, a redução de cálculos de distância

tem um menor impacto quando a métrica utilizada não é cara computacionalmente, que é o caso da métrica L_2.

4.2.2 Resultados sobre o conjunto ALOI-H

Para a realização dos experimentos sobre o conjunto ALOI-H, foram utilizados 50 elementos de consulta aleatórios. Os resultados são apresentados nas Figuras 12 e 13.

A Figura 12 apresenta os resultados das consultas k-NN. Considerando tempo de execução, as técnicas ACIR, ACIR+ACIP e NN-graph proporcionaram, respectivamente, ganhos de 40.1% a 48%, de 42.1% a 48.2% e de 35.4% a 39%. Considerando cálculos de distância, as técnicas ACIR, ACIR+ACIP e NN-graph apresentaram ganhos de 37.8% a 45.7%, de 39.6% a 45.9% e de 35.5% a 39%. Considerando acessos a disco, as técnicas ACIR e ACIR+ACIP proporcionaram ganhos de 23.2% a 28.6% e de 30.3% a 34.4%.

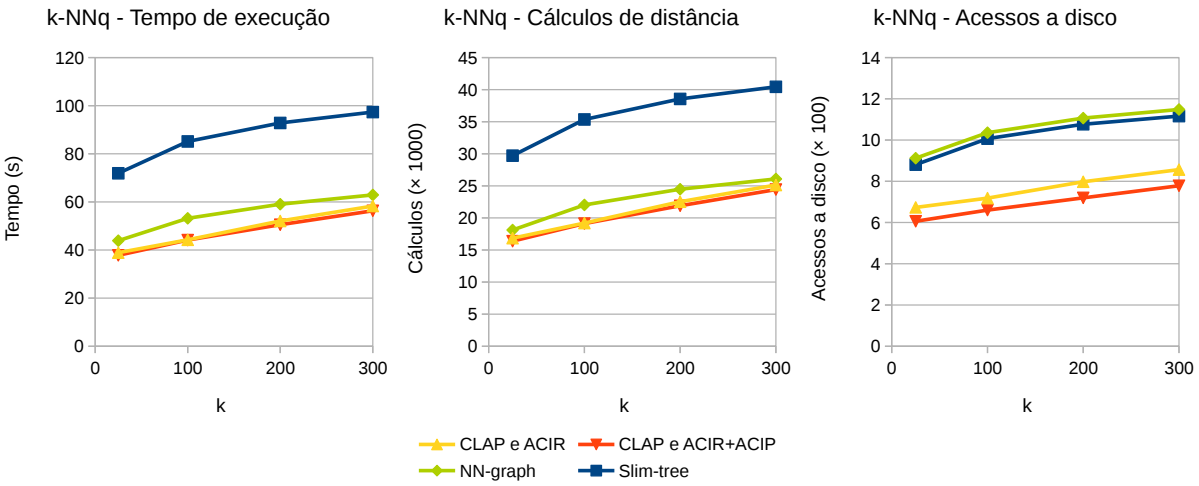


Figura 12 – Resultados de consultas k-NN no conjunto ALOI-H.

A Figura 13 apresenta os resultados das consultas por abrangência. Considerando tempo de execução, as técnicas ACIR, ACIR+ACIP e NN-graph apresentaram, respectivamente, ganhos de 43.1% a 51.1%, de 47.4% a 53.5% e de 38.7% a 40.6%. Considerando cálculos de distância, as técnicas ACIR, ACIR+ACIP e NN-graph apresentaram ganhos de 41% a 49%, de 45.2% a 51.6% e de 39% a 40.9%. Considerando acessos a disco, as técnicas ACIR e ACIR+ACIP apresentaram ganhos de 24.7% a 30.5% e de 33.3% a 37.7%.

Neste conjunto de experimentos, a técnica NN-graph também apresentou ganhos em tempo de execução. Dado o custo computacional quadrático da métrica Mahalanobis, em contraste ao custo linear da métrica L_2, a redução de cálculos de distância proporcionada pela técnica NN-graph teve um maior impacto no tempo de execução das consultas.

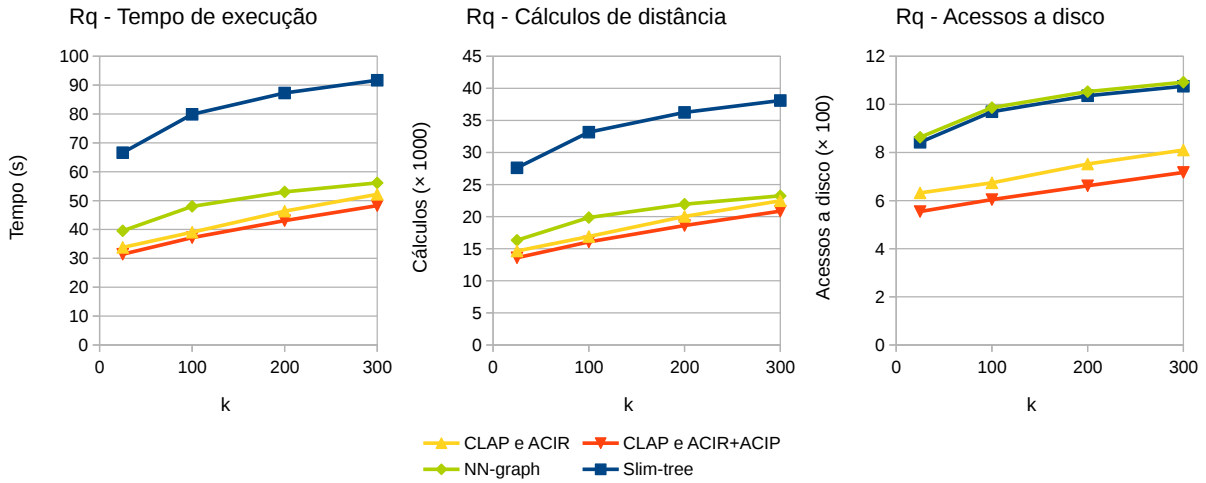


Figura 13 – Resultados de consultas por abrangência no conjunto ALOI-H.

4.2.3 Resultados sobre o conjunto CoPhIR-1M-WL2

Para a execução dos experimentos sobre o conjunto CoPhIR-1M-WL2, foram usados 500 elementos de consulta. Os resultados são apresentados nas Figuras 14 e 15.

A Figura 14 apresenta os resultados das consultas k-NN. Considerando tempo de execução, as técnicas ACIR e ACIR+ACIP apresentaram ganhos de 0.9% a 5% e de 19.1% a 20.7%. Considerando cálculos de distância, as técnicas ACIR, ACIR+ACIP e NN-graph proporcionaram ganhos de 5.9% a 9.1%, de 15% a 16.5% e de 16.8% a 20.7%. Considerando acessos a disco, as técnicas ACIR e ACIR+ACIP proporcionaram ganhos de 2.9% a 3.1% e de 7.5% a 9.8%.

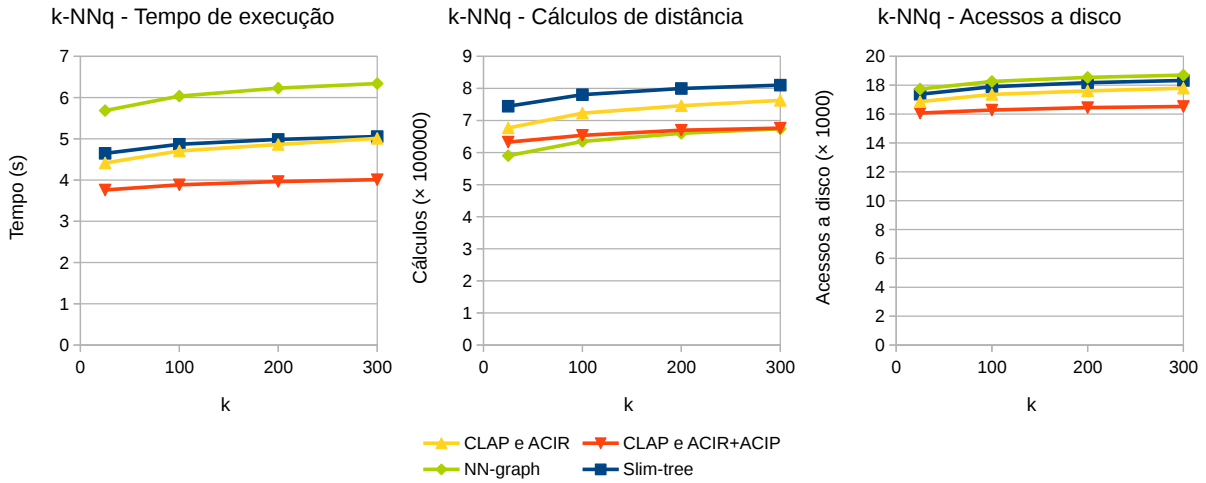


Figura 14 – Resultados de consultas k-NN no conjunto CoPhIR-1M-WL2.

A Figura 15 apresenta os resultados das consultas por abrangência. Considerando tempo de execução, as técnicas ACIR e ACIR+ACIP, respectivamente, proporcionaram ganhos de 3% a 7.2% e de 23.4% a 25.1%. Considerando cálculos de distância, as técnicas ACIR, ACIR+ACIP e NN-graph proporcionaram ganhos de 6.8% a 10.2%, de 16.2% a 18%

e de 18.3% a 22%. Por fim, considerando acessos a disco, as técnicas ACIR e ACIR+ACIP apresentaram ganhos de 3% a 3.2% e de 7.6% a 10.1%.

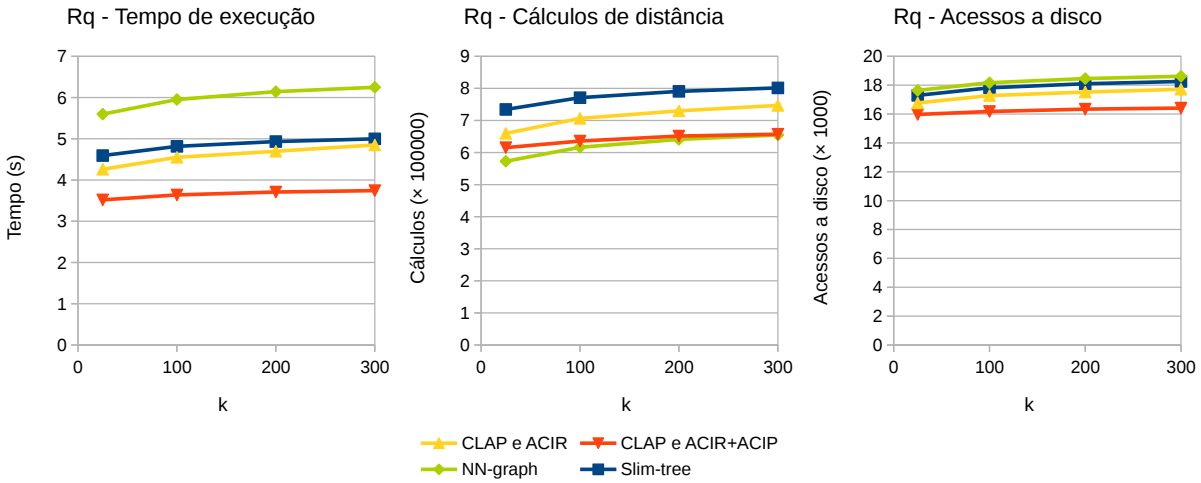


Figura 15 – Resultados de consultas por abrangência no conjunto CoPhIR-1M-WL2.

Novamente, como o custo computacional da distância euclidiana ponderada não é caro, a redução de cálculos de distância proporcionada pela técnica NN-graph não levou a ganhos em tempo de execução, dado seu empate técnico com a Slim-tree em acessos a disco e o custo considerável de manipulação de suas informações adicionais na estrutura.

Nos resultados sobre este conjunto de dados, verifica-se uma diferença mais acentuada entre as técnicas ACIR e ACIR+ACIP. Visto que este conjunto possui uma quantidade maior de elementos do que os conjuntos anteriores, a redução tanto em cálculos de distância quanto em acessos a disco tende a ter mais impacto no tempo de execução das consultas. Como a técnica ACIR+ACIP proporciona uma maior redução em acessos a disco em comparação à técnica ACIR, o que também leva a uma menor quantidade de cálculos de distância, especialmente em conjuntos de dados maiores, seus tempos de execução foram consideravelmente menores do que os tempos de execução da técnica ACIR.

4.2.4 Resultados sobre o conjunto CoPhIR-1M-M

Para a execução dos experimentos sobre o conjunto CoPhIR-1M-M, foram usados 50 elementos de consulta. Os resultados são apresentados nas Figuras 16 e 17.

A Figura 16 apresenta os resultados das consultas k-NN. Considerando tempo de execução, as técnicas ACIR, ACIR+ACIP e NN-graph proporcionaram ganhos de 16.2% a 22.9%, de 16.9% a 24.5% e de 13.3% a 16.3%. Considerando cálculos de distância, as técnicas ACIR, ACIR+ACIP e NN-graph apresentaram ganhos de 12.4% a 19.6%, de 12.6% a 20.6% e de 15.1% a 18.3%. Considerando acessos a disco, as técnicas ACIR e ACIR+ACIP apresentaram ganhos de 5.9% a 7.3% e de 14% a 17%.

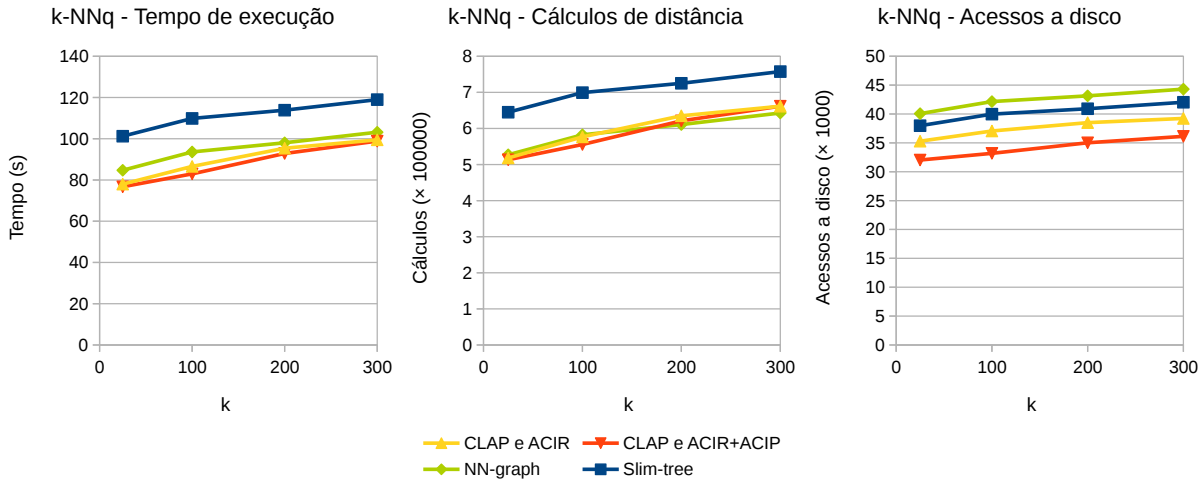


Figura 16 – Resultados de consultas k-NN no conjunto CoPhIR-1M-M.

A Figura 17 apresenta os resultados das consultas por abrangência. Considerando tempo de execução, as técnicas ACIR, ACIR+ACIP e NN-graph apresentaram ganhos de 18.2% a 25.4%, de 19.6% a 26.9% e de 14.5% a 17.2%. Considerando cálculos de distância, as três técnicas apresentaram, respectivamente, ganhos de 14.1% a 21.3%, de 14.4% a 22.8% e de 16.6% a 19.3%. Considerando acessos a disco, as técnicas ACIR e ACIR+ACIP apresentaram ganhos de 5.9% a 7.3% e de 14.2% a 17.4%.

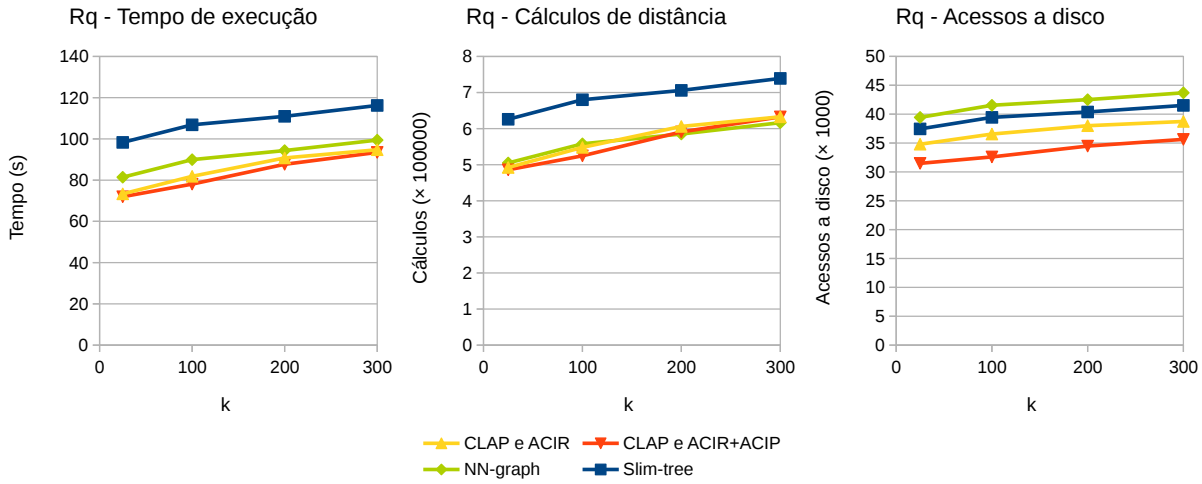


Figura 17 – Resultados de consultas por abrangência no conjunto CoPhIR-1M-M.

Neste conjunto de experimentos, assim como no conjunto ALOI-H, a técnica NN-graph também apresentou ganhos em tempo de execução. Mesmo com um maior número de acessos a disco do que as outras técnicas, inclusive a Slim-tree, os ganhos em cálculos de distância proporcionados pela NN-graph tiveram um maior peso sobre o desempenho das consultas.

4.3 Análise de construção dos métodos de acesso

Esta seção apresenta o impacto das técnicas avaliadas nesta dissertação em termos de tempo de construção das estruturas, tamanho de página utilizado e tamanho do arquivo de dados resultante. Essas informações são apresentadas na Tabela 1.

Note-se que as estruturas estendidas pelas técnicas CLAP, ACIR e ACIR+ACIP utilizaram tamanhos de página diferentes para os nós do tipo índice. No caso das técnicas CLAP e ACIR, utilizou-se o dobro do tamanho de página dos nós do tipo folha, enquanto no caso das técnicas CLAP e ACIR+ACIP utilizou-se o quádruplo do tamanho.

Essa decisão foi tomada de forma a minimizar o *overhead* de informações em nós do tipo índice, acarretado pelo processo de antecipação das técnicas ACIR e ACIR+ACIP. O tamanho de página dos nós do tipo índice, usualmente, será um múltiplo do tamanho de página dos nós do tipo folha, possibilitando o uso do mesmo *buffer pool* no gerenciamento de *buffer* para ambos os tipos de nó. Embora os nós do tipo índice tenham o dobro (ou o quádruplo) do tamanho original ao utilizar as técnicas de antecipação, levando um tempo maior para ler as páginas do disco, o aumento no poder de poda das estruturas possibilitou obter ganhos de desempenho na execução de consultas.

Note-se que os arquivos de dados referentes às técnicas propostas possuem tamanhos similares, inclusive menores do que os arquivos da Slim-tree. Isso pode ser explicado pelo fato de que, com o processo de antecipação de informações, nós do tipo folha passam a armazenar menos informações. Uma vez que a quantidade de nós do tipo folha é muito maior (chegando a mais de 95% da estrutura) do que a quantidade de nós do tipo índice, o fato de armazenar menos informações no nível folha leva a arquivos de dados ligeiramente menores.

Tabela 1 – Informações de construção dos métodos de acesso

Conjunto de dados	Método de acesso	Tempo (s)	Página (KB)	Arquivo
ALOI-T	Slim-tree	40.807	32	103MB
	CLAP e ACIR	43.650	64 (índice) – 32 (folha)	101.2MB
	CLAP e ACIR+ACIP	49.127	128 (índice) – 32 (folha)	100.7MB
	NN-graph	53.166	32	105MB
ALOI-H	Slim-tree	29637.4	64	162MB
	CLAP e ACIR	33289.4	128 (índice) – 64 (folha)	155.9MB
	CLAP e ACIR+ACIP	34532.7	256 (índice) – 64 (folha)	157.3MB
	NN-graph	43101.2	64	160MB
CoPhIR-1M-WL2	Slim-tree	758.633	64	1.4GB
	CLAP e ACIR	871.270	128 (índice) – 64 (folha)	1.334GB
	CLAP e ACIR+ACIP	883.581	256 (índice) – 64 (folha)	1.365GB
	NN-graph	962.539	64	1.4GB
CoPhIR-1M-M	Slim-tree	11964.4	8	449MB
	CLAP e ACIR	13385.5	16 (índice) – 8 (folha)	434MB
	CLAP e ACIR+ACIP	13841.9	32 (índice) – 8 (folha)	439MB
	NN-graph	14502.3	8	473MB

O tempo de construção das estruturas referentes às técnicas propostas foi maior

do que o tempo requerido pela estrutura original, como esperado, uma vez que as técnicas CLAP, ACIR e ACIR+ACIP realizam computações adicionais, como os cálculos de distância necessários para armazenar a distância de cada elemento do nó ao pivô adicional. Entretanto, as estruturas referentes à técnica NN-graph demandaram tempos ainda maiores para serem construídas, permitindo verificar que a manutenção do grafo de vizinhos mais próximos da técnica NN-graph é mais onerosa.

Os tempos de construção das estruturas referentes às técnicas ACIR, ACIR+ACIP e NN-graph foram, respectivamente, de 7% a 14.8%, de 15.7% a 20.4% e de 21.2% a 30.3% maiores do que os tempos de construção da Slim-tree. No entanto, considerando os ganhos de desempenho em consultas por similaridade, os benefícios proporcionados pelas técnicas apresentadas nesta dissertação compensam o aumento nos tempos de construção.

4.4 Considerações finais

Este capítulo descreveu os experimentos de desempenho realizados entre quatro conjuntos de dados e duas métricas, envolvendo as técnicas apresentadas nesta dissertação e a técnica NN-graph, que representa o estado da arte de técnicas baseadas em múltiplos pivôs locais, bem como apresentou os resultados obtidos e analisou questões de construção das estruturas. O capítulo a seguir apresenta as conclusões deste trabalho.

5 CONCLUSÃO

O crescimento no volume de dados complexos tem demandado o desenvolvimento de técnicas mais eficientes, tanto para armazenamento quanto para recuperação. Além de eficiência, aplicações modernas requerem estruturas de indexação capazes de tratar dados complexos independentemente de suas peculiaridades, como distribuição, cardinalidade e dimensionalidade do conjunto de dados em questão. No caso de aplicações *online*, também é especialmente importante o uso de estruturas de indexação que sejam dinâmicas, permitindo adicionar e remover elementos a qualquer momento sem que o desempenho de tais aplicações seja prejudicado.

Nesse contexto, esta dissertação desenvolveu e avaliou novas técnicas visando melhorar o desempenho de MAMs dinâmicos na execução de consultas: a técnica baseada em múltiplos pivôs locais CLAP, que reduz o número de cálculos de distância, e as técnicas de antecipação de informações ACIR e ACIR+ACIP, que reduzem acessos a disco.

As técnicas foram implementadas sobre o MAM Slim-tree e comparadas à técnica NN-graph, considerada o estado da arte de técnicas baseadas em múltiplos pivôs locais. A avaliação das técnicas se deu através de consultas k-NN e consultas por abrangência, variando-se a seletividade das consultas, sobre conjuntos de dados com diferentes cardinalidades, dimensionalidades e com métricas de diferentes custos computacionais.

Nos experimentos que utilizaram uma métrica cara, o tempo de execução de todas as técnicas avaliadas foi inferior ao da Slim-tree. No entanto, o desempenho em consultas por similaridade utilizando-se as técnicas CLAP, ACIR e ACIR+ACIP foi superior ao da técnica NN-graph para todos os conjuntos de dados. Nos experimentos de modo geral, o número de cálculos de distância com as técnicas CLAP, ACIR e ACIR+ACIP foi menor do que com a técnica NN-graph para seletividades menores e levemente maior para seletividades maiores. Em termos de acessos a disco, a técnica NN-graph apresentou empate técnico com a Slim-tree, visto que seu foco é a redução de cálculos de distância.

As técnicas ACIR e ACIR+ACIP mostraram um desempenho parecido, em termos de tempo de execução, na maioria dos conjuntos utilizados. A diferença mais acentuada ocorreu nos experimentos sobre o conjunto cuja cardinalidade e dimensionalidade eram as maiores e que fez uso de uma métrica barata. Devido às características desse conjunto, o tempo de execução de consultas por similaridade tende a ser mais impactado através da redução de cálculos de distância e de acessos a disco. Ambas as variáveis tiveram uma maior redução com a técnica ACIR+ACIP, o que levou a menores tempos de execução.

De modo geral, os maiores ganhos das técnicas CLAP, ACIR e ACIR+ACIP foram para seletividades menores. Contudo, os ganhos foram expressivos em todos os cenários

avaliados. Além disso, o fato de os maiores ganhos terem sido para seletividades menores pode contribuir positivamente para o desempenho de diversas aplicações reais. Sistemas de buscas por similaridade sobre imagens médicas, por exemplo, são aplicações em que o usuário pode estar mais interessado em recuperar uma quantidade razoavelmente pequena de elementos similares a um determinado elemento fornecido.

Um dos possíveis trabalhos futuros diz respeito à avaliação do impacto decorrente da utilização de mais de 1 pivô adicional local pela técnica CLAP. Nesse caso, o aumento no número de pivôs adicionais justificaria o uso de tamanhos de página maiores no nível índice da estrutura. Um possível desafio nessa abordagem seria lidar com o *overhead* de informações antecipadas, que passaria a ser maior do que nas técnicas com 1 pivô adicional. Entretanto, o maior número de informações no nível índice da estrutura resultaria em um maior poder de poda, permitindo descartar, inclusive, subárvores inteiras no nível índice. Assim, a análise desse impacto seria interessante e poderia trazer bons resultados.

Outros trabalhos futuros poderiam integrar as técnicas desenvolvidas a ferramentas de busca por similaridade e até mesmo a SGBDRs. No caso de um SGBDR, por exemplo o Oracle, é permitido ao usuário criar seus próprios tipos de índice e implementar funções que invoquem a execução de estruturas de indexação externas ao SGBDR. Em aplicações *online* que fazem uso de SGBDRs para manipular dados complexos, seria possível melhorar o desempenho de suas estruturas de indexação aplicando-se as técnicas desenvolvidas nesta dissertação.

REFERÊNCIAS

[1] BARRIOS, J. M.; BUSTOS, B. Automatic weight selection for multi-metric distances. In: *Proceedings of the 4th International Conference on Similarity Search and Applications*. New York, NY, USA: ACM, 2011. (SISAP '11), p. 61–68. ISBN 978-1-4503-0795-6. Disponível em: <http://doi.acm.org/10.1145/1995412.1995425>.

[2] FALOUTSOS, C. *Searching Multimedia Databases by Content*. [S.l.]: Kluwer Academic Publishers, 1996. 1–155 p. (Advances in Database Systems, v. 3). ISBN 978-0-7923-9777-9.

[3] UHLMANN, J. K. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, Elsevier Science Publishers B.V., v. 40, n. 4, p. 175–179, 1991. ISSN 0020-0190. Disponível em: <http://www.sciencedirect.com/science/article/pii/002001909190074R>.

[4] YIANILOS, P. N. Excluded middle vantage point forests for nearest neighbor search. In: *Proceedings of the 6th DIMACS Implementation Challenge: Near Neighbor Searches*. Baltimore, MD, USA: [s.n.], 1999. (ALENEX '99). Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.2373&rep=rep1&type=pdf>.

[5] KASTER, D. S. et al. Nearest neighbor queries with counting aggregate-based conditions. *Journal of Information and Data Management*, v. 2, n. 3, p. 401–416, 2011. Disponível em: <http://seer.lcc.ufmg.br/index.php/jidm/article/view/159>.

[6] BARIONI, M. C. N. et al. Querying multimedia data by similarity in relational DBMS. In: YAN, L.; MA, Z. (Ed.). *Advanced Database Query Systems: Techniques, Applications and Technologies*. Hershey, PA, USA: IGI Global, 2011. p. 323–359.

[7] WILSON, D. R.; MARTINEZ, T. R. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, AI Access Foundation, USA, v. 6, n. 1, p. 1–34, 1997. ISSN 1076-9757. Disponível em: <http://dl.acm.org/citation.cfm?id=1622767.1622768>.

[8] HAFNER, J. et al. Efficient color histogram indexing for quadratic form distance functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE Computer Society, Washington, DC, USA, v. 17, n. 7, p. 729–736, 1995. ISSN 0162-8828. Disponível em: <http://dx.doi.org/10.1109/34.391417>.

[9] DEZA, M. M.; DEZA, E. Encyclopedia of distances. In: *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009. p. 1–583. ISBN 978-3-642-00233-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-00234-2_1>.

[10] LONG, F.; ZHANG, H.; FENG, D. D. Fundamentals of content-based image retrieval. In: FENG, D. D.; SIU, W.-C.; ZHANG, H. (Ed.). *Multimedia Information Retrieval and Management*. Springer Berlin Heidelberg, 2003, (Signals and Communication Technology). p. 1–26. ISBN 978-3-642-05533-1. Disponível em: <http://dx.doi.org/10.1007/978-3-662-05300-3_1>.

[11] ZEZULA, P. et al. *Similarity Search: The Metric Space Approach*. 1st. ed. [S.l.]: Springer Publishing Company, Inc., 2006. (Advances in Database Systems, v. 32). ISBN 978-0387-29146-8.

[12] BAYER, R.; MCCREIGHT, E. M. Organization and maintenance of large ordered indexes. *Acta Informatica*, Springer-Verlag, v. 1, n. 3, p. 173–189, 1972. ISSN 0001-5903. Disponível em: <http://dx.doi.org/10.1007/BF00288683>.

[13] COMER, D. The ubiquitous B-tree. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 11, n. 2, p. 121–137, 1979. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/356770.356776>.

[14] ROBINSON, J. T. The K-D-B-tree: A search structure for large multidimensional dynamic indexes. In: *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 1981. (SIGMOD '81), p. 10–18. ISBN 0-89791-040-0. Disponível em: <http://doi.acm.org/10.1145/582318.582321>.

[15] GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. *ACM SIGMOD Record*, ACM, New York, NY, USA, v. 14, n. 2, p. 47–57, 1984. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/971697.602266>.

[16] SELLIS, T. K.; ROUSSOPOULOS, N.; FALOUTSOS, C. The R+-tree: A dynamic index for multi-dimensional objects. In: *Proceedings of the 13th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987. (VLDB '87), p. 507–518. ISBN 0-934613-46-X. Disponível em: <http://dl.acm.org/citation.cfm?id=645914.671636>.

[17] BECKMANN, N. et al. The R*-tree: An efficient and robust access method for points and rectangles. *ACM SIGMOD Record*, ACM, New York, NY, USA, v. 19, n. 2, p. 322–331, 1990. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/93605.98741>.

[18] HELLERSTEIN, J. M.; NAUGHTON, J. F.; PFEFFER, A. Generalized search trees for database systems. In: *Proceedings of the 21th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995. (VLDB '95), p. 562–573. ISBN 1-55860-379-4. Disponível em: <http://dl.acm.org/citation.cfm?id=645921.673145>.

[19] PAPADIAS, D.; MAMOULIS, N.; THEODORIDIS, Y. Processing and optimization of multiway spatial joins using R-trees. In: *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. New York, NY, USA: ACM, 1999. (PODS '99), p. 44–55. ISBN 1-58113-062-7. Disponível em: <http://doi.acm.org/10.1145/303976.303981>.

[20] GAEDE, V.; GUNTHER, O. Multidimensional access methods. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 30, n. 2, p. 170–231, 1998. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/280277.280279>.

[21] ASLANDOGAN, Y. A.; YU, C. T. Techniques and systems for image and video retrieval. *IEEE Transactions on Knowledge and Data Engineering*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 11, n. 1, p. 56–63, 1999. ISSN 1041-4347. Disponível em: <http://dx.doi.org/10.1109/69.755615>.

[22] LEW, M. S. et al. Content-based multimedia information retrieval: State of the art and challenges. *ACM Transactions on Multimedia Computing, Communications, and Applications*, ACM, New York, NY, USA, v. 2, n. 1, p. 1–19, 2006. ISSN 1551-6857. Disponível em: <http://doi.acm.org/10.1145/1126004.1126005>.

[23] BURKHARD, W. A.; KELLER, R. M. Some approaches to best-match file searching. *Communications of the ACM*, ACM, New York, NY, USA, v. 16, n. 4, p. 230–236, 1973. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/362003.362025>.

[24] YIANILOS, P. N. Data structures and algorithms for nearest neighbor search in general metric spaces. In: *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1993. (SODA '93), p. 311–321. ISBN 0-89871-313-7. Disponível em: <http://dl.acm.org/citation.cfm?id=313559.313789>.

[25] BOZKAYA, T.; OZSOYOGLU, M. Distance-based indexing for high-dimensional metric spaces. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 1997. (SIGMOD '97), p. 357–368. ISBN 0-89791-911-4. Disponível em: <http://doi.acm.org/10.1145/253260.253345>.

[26] BAEZA-YATES, R. A. et al. Proximity matching using fixed-queries trees. In: *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*. London, UK: Springer-Verlag, 1994. (CPM '94), p. 198–212. ISBN 3-540-58094-8. Disponível em: <http://dl.acm.org/citation.cfm?id=647814.738307>.

[27] CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree: An efficient access method for similarity search in metric spaces. In: *Proceedings of the 23rd International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers, Inc., 1997. (VLDB '97), p. 426–435. ISBN 1-55860-470-7. Disponível em: <http://dl.acm.org/citation.cfm?id=645923.671005>.

[28] TRAINA-JR., C. et al. Slim-trees: High performance metric trees minimizing overlap between nodes. In: *Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology*. London, UK: Springer-Verlag, 2000. (EDBT '00), p. 51–65. ISBN 3-540-67227-3. Disponível em: <http://dl.acm.org/citation.cfm?id=645339.650146>.

[29] TRAINA-JR., C. et al. Fast indexing and visualization of metric data sets using Slim-trees. *IEEE Transactions on Knowledge and Data Engineering*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 14, n. 2, p. 244–260, 2002. ISSN 1041-4347. Disponível em: <http://dx.doi.org/10.1109/69.991715>.

[30] SOUZA, J. A.; RAZENTE, H. L.; BARIONI, M. C. N. Faster construction of ball-partitioning-based metric access methods. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2013. (SAC '13), p. 8–12. Disponível em: <http://doi.acm.org/10.1145/2480362.2480365>.

[31] VIEIRA, M. R. et al. DBM-tree: Trading height-balancing for performance in metric access methods. *Journal of the Brazilian Computer Society*, Springer-Verlag, v. 11, n. 3, p. 37–51, 2005. ISSN 0104-6500. Disponível em: <http://dx.doi.org/10.1007/BF03192381>.

[32] VIEIRA, M. R. et al. DBM-tree: A dynamic metric access method sensitive to local density data. *Journal of Information and Data Management*, v. 1, n. 1, p. 111–127, 2010. ISSN 2178-7107. Disponível em: <http://seer.lcc.ufmg.br/index.php/jidm/article/view/22>.

[33] SKOPAL, T.; HOKSZA, D. Improving the performance of M-tree family by nearest-neighbor graphs. In: IOANNIDIS, Y.; NOVIKOV, B.; RACHEV, B. (Ed.). *ADBIS 2007*. Heidelberg: Springer, 2007, (LNCS, v. 4690). p. 172–188. ISBN 978-3-540-75184-7.

[34] POLA, I. R. V.; TRAINA-JR., C.; TRAINA, A. J. M. The MM-tree: A memory-based metric tree without overlap between nodes. In: IOANNIDIS, Y.; NOVIKOV, B.; RACHEV, B. (Ed.). *ADBIS 2007*. Heidelberg: Springer, 2007, (LNCS, v. 4690). p. 157–171. ISBN 978-3-540-75184-7.

[35] CARELO, C. C. M. et al. Slicing the metric space to provide quick indexing of complex data in the main memory. *Information Systems*, v. 36, n. 1, p. 79–98, 2011. ISSN 0306-4379.

[36] BUSTOS, B.; NAVARRO, G.; CHAVEZ, E. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, v. 24, n. 14, p. 2357–2366, 2003. ISSN 0167-8655. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167865503000655>.

[37] FILHO, R. F. S. et al. Similarity search without tears: The OMNI-family of all-purpose access methods. In: *Proceedings of the 17th International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2001. (ICDE '01), p. 623–630. ISBN 0-7695-1001-9. Disponível em: <http://dl.acm.org/citation.cfm?id=645484.656543>.

[38] TRAINA-JR., C. et al. The OMNI-family of all-purpose access methods: A simple and effective way to make similarity search more efficient. *The VLDB Journal*, Springer-Verlag, v. 16, n. 4, p. 483–505, 2007. ISSN 1066-8888. Disponível em: <http://dx.doi.org/10.1007/s00778-005-0178-0>.

[39] TRAINA-JR., C. et al. How to improve the pruning ability of dynamic metric access methods. In: *Proceedings of the 11th International Conference on Information and Knowledge Management*. New York, NY, USA: ACM, 2002. (CIKM '02), p. 219–226. ISBN 1-58113-492-4. Disponível em: <http://doi.acm.org/10.1145/584792.584831>.

[40] ESULI, A. PP-Index: Using permutation prefixes for efficient and scalable approximate similarity search. In: *7th Workshop on Large-Scale Distributed Systems for Information Retrieval*. [S.l.: s.n.], 2009. (LSDS-IR '09), p. 17–24.

[41] ESULI, A. MiPai: Using the PP-Index to build an efficient and scalable similarity search system. In: *Proceedings of the 2nd International Workshop on Similarity Search and Applications*. Washington, DC, USA: IEEE Computer Society, 2009. (SISAP '09), p. 146–148. ISBN 978-0-7695-3765-8. Disponível em: <http://dx.doi.org/10.1109/SISAP.2009.14>.

[42] ESULI, A. PP-Index: Using permutation prefixes for efficient and scalable similarity search (extended abstract). In: *Proceedings of the 18th Italian Symposium on Advanced Database Systems*. [S.l.: s.n.], 2010. (SEBD '10), p. 318–325.

[43] ESULI, A. Use of permutation prefixes for efficient and scalable approximate similarity search. *Information Processing & Management*, v. 48, n. 5, p. 889–902, 2012. ISSN 0306-4573. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306457310001019>.

[44] SKOPAL, T.; POKORNY, J.; SNASEL, V. Nearest neighbours search using the PM-tree. In: ZHOU, L.-Z.; OOI, B. C.; MENG, X. (Ed.). *Database Systems for Advanced Applications*. Springer Berlin Heidelberg, 2005, (Lecture Notes in Computer Science, v. 3453). p. 803–815. ISBN 978-3-540-25334-1. Disponível em: <http://dx.doi.org/10.1007/11408079_73>.

[45] SKOPAL, T. Pivoting M-tree: A metric access method for efficient similarity search. In: *DATESO*. Desna, Cerna Ricka, Czech Republic: [s.n.], 2004. p. 27–37.

[46] SKOPAL, T.; POKORNY, J.; SNASEL, V. PM-tree: Pivoting metric tree for similarity search in multimedia databases. In: *ADBIS (Local Proceedings)*. Budapest, Hungary: Hungarian Academy of Sciences, 2004.

[47] LOKOC, J. et al. Cut-region: A compact building block for hierarchical metric indexing. In: *Proceedings of the 5th International Conference on Similarity Search and Applications*. Berlin, Heidelberg: Springer-Verlag, 2012. (SISAP '12), p. 85–100. ISBN 978-3-642-32152-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-32153-5_7>.

[48] LOKOC, J. et al. On indexing metric spaces using cut-regions. *Information Systems*, v. 43, n. 0, p. 1–19, 2014. ISSN 0306-4379. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306437914000258>.

[49] GEUSEBROEK, J.-M.; BURGHOUTS, G. J.; SMEULDERS, A. W. M. The Amsterdam Library of Object Images. *International Journal of Computer Vision*, Kluwer Academic Publishers, v. 61, n. 1, p. 103–112, 2005. ISSN 0920-5691. Disponível em: <http://dx.doi.org/10.1023/B%3AVISI.0000042993.50813.60>.

[50] HARALICK, R. M.; SHANMUGAM, K. S.; DINSTEIN, I. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, IEEE Computer Society, SMC-3, n. 6, p. 610–621, 1973. ISSN 0018-9472.

[51] HARALICK, R. M. Statistical and structural approaches to texture. *Proceedings of the IEEE*, IEEE Computer Society, v. 67, n. 5, p. 786–804, 1979. ISSN 0018-9219.

[52] BOLETTIERI, P. et al. CoPhIR: A test collection for content-based image retrieval. *Computing Research Repository*, abs/0905.4627v2, 2009.

[53] BATKO, M.; KOHOUTKOVA, P.; NOVAK, D. CoPhIR image collection under the microscope. In: *2nd International Workshop on Similarity Search and Applications*. Washington, DC: IEEE Computer Society, 2009. p. 47–54. ISBN 978-0-7695-3765-8.

[54] OLIVEIRA, P. H.; TRAINA-JR., C.; KASTER, D. S. Improving the pruning ability of dynamic metric access methods with local additional pivots and anticipation of information. In: *ADBIS 2015*. Heidelberg: Springer, 2015, (LNCS, v. 9282). ISBN 978-3-319-23134-1.

TRABALHOS PUBLICADOS PELO AUTOR

Trabalhos publicados pelo autor durante o programa.

1. OLIVEIRA, P. H.; ZARPELÃO, B. B.; KASTER, D. S.. **Aumentando o desempenho de análises de dados de fluxos IP com bancos de dados orientados a coluna**, SBSI 2014 — X Simpósio Brasileiro de Sistemas de Informação, Abril/2014, UEL, p. 172–183, ISSN 2177-885X (Qualis CC 2012, B4)

2. OLIVEIRA, P. H.; KASTER, D. S.. **Desenvolvimento de um método de acesso métrico dinâmico eficiente baseado em pivôs adicionais**, WTDBD 2014 — XIII Workshop de Teses e Dissertações em Banco de Dados, Outubro/2014, SBC, p. 318–324, ISSN 2316-5170

3. OLIVEIRA, P. H.; TRAINA JR., C.; KASTER, D. S.. **Improving the Pruning Ability of Dynamic Metric Access Methods with Local Additional Pivots and Anticipation of Information**, ADBIS 2015 — 19th East-European Conference on Advances in Databases and Information Systems, Setembro/2015, Springer, ISBN 978-3-319-23134-1 (Qualis CC 2012, B1)