Universidade Estadual de Londrina

LEANDRO CAVALARI SOARES

UM MÉTODO DE ACESSO MÉTRICO PARA CONSULTAS POR SIMILARIDADE COM CONDIÇÕES ADICIONAIS

Londrina
2014

LEANDRO CAVALARI SOARES

UM MÉTODO DE ACESSO MÉTRICO PARA CONSULTAS POR SIMILARIDADE COM CONDIÇÕES ADICIONAIS

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Daniel dos Santos Kaster

Londrina
2014

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

Dados Internacionais de Catalogação-na-Publicação (CIP)

S676m Soares, Leandro Cavalari.
Um método de acesso métrico para consultas por similaridade com condições adicionais / Leandro Cavalari Soares. – Londrina, 2014.
89 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, 2014.
Inclui bibliografia.

1. Banco de dados – Gerência – Teses. 2. Estruturas de dados (Computação) – Teses. 3. Organização de arquivos (Computação) – Teses. 4. Sistemas multimídia – Teses. 5. 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

LEANDRO CAVALARI SOARES

UM MÉTODO DE ACESSO MÉTRICO PARA CONSULTAS POR SIMILARIDADE COM CONDIÇÕES ADICIONAIS

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

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

Prof. Dr. Renato Bueno
Universidade Federal de São Carlos – UFScar

Prof. Dr. Sylvio Barbon Junior
Universidade Estadual de Londrina – UEL

Profa. Dra. Jandira Guenka Palma
Universidade Estadual de Londrina – UEL

Londrina, 30 de abril de 2014

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus pela oportunidade e capacidade que me foram concedidas para o desenvolvimento deste projeto de vida!

O apoio dos meus pais Érica Regina e João Batista, familiares, namorada Taciana e amigos foi muito importante, pois me acompanharam de perto, compartilhando tanto das conquistas quanto dos momentos em que o cansaço e o sono não cooperavam, fazendo até com que eles não tivessem de minha parte a atenção merecida.

Agradeço ao amigo e orientador Daniel dos Santos Kaster, pelo comprometimento desde a orientação pré-mestrado iniciada a distância, continuada de maneira parcial e concorrente à função que desempenho como colaborador da Veltec S/A. Seu compartilhamento de valores de vida e experiência profissional fazem, naturalmente, a diferença na vida dos formandos.

Agradeço também ao professor Adilson Luiz Bonifácio pelo apoio no início do programa, bem como ao amigo Giovani Benedetti Penha e companheiros da Veltec S/A que também me proporcionaram as condições necessárias para a evolução educacional e profissional.

Keep walking, Johnnie Sino Walker!

(Daniel dos Santos Kaster)

SOARES, L. C.. **Um Método de Acesso Métrico para Consultas por Similaridade com Condições Adicionais**. 89 f. Dissertação de Mestrado (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina–PR, 2014.

RESUMO

O rápido crescimento da massa de dados complexos na atualidade, tais como imagens, vídeos e séries temporais, intensifica a importância do desenvolvimento de estratégias de busca eficientes para este tipo de dados. Aplicações que lidam com dados complexos aplicam consultas por similaridade na recuperação dos mesmos, combinando condições de similaridade com condições associadas a atributos de tipos de dados tradicionais. Existem diversas estruturas de indexação para consultas por similaridade, no entanto, grande parte delas não trabalha com dados tradicionais como condição de busca. As estruturas existentes que respondem a consultas combinando condições contendo tanto atributos complexos quanto tradicionais, em geral, suportam apenas condições baseadas em palavras-chave. Esta dissertação de mestrado propõe um novo método de acesso métrico, chamado *cx-Sim tree* (*condition-extended Similarity tree*), para executar eficientemente consultas por similaridade com condições adicionais gerais (não somente baseadas em palavras-chave) sobre dados complexos. A *cx-Sim tree* é um índice composto que tem quatro variações de implementação, contendo um atributo complexo e um ou mais atributos tradicionais. Experimentos sobre bases de dados complexos reais para validar comparativamente o método mostraram que ele obteve maior desempenho que as abordagens existentes para consultas por similaridade com condições simples e que as variações desenvolvidas cobrem diferentes situações considerando-se consultas com condições compostas, possibilitando recuperação rápida de dados em todas as situações.

Palavras-chave: Métodos de acesso métrico. Consultas por similaridade estendidas com condições. Banco de dados multimídia.

SOARES, L. C.. **A Metric Access Method for Similarity Queries with Additional Conditions**. 89 p. Master's Thesis (Master in Science in Computer Science) – State University of Londrina, Londrina–PR, 2014.

ABSTRACT

The fast growth of complex data repositories, such as images, videos and time series, in recent years is intensifying the importance of developing efficient search strategies over these data types. Applications that deal with complex data employ similarity queries to retrieve data, often combining similarity conditions with conditions over other associated attributes of traditional data types. There are several indexing structures for answering similarity queries, however most of them do not work when there are additional search conditions. The existing structures that answer queries combining conditions over complex and traditional attributes, in general, support only keyword-based conditions. This master's thesis proposes a new metric access method, called *cx-Sim tree* (condition-extended Similarity tree), to efficiently execute similarity queries with additional general conditions (not only keyword-based) over complex data. The *cx-Sim tree* is a composite index that has four implementation variations, containing one complex attribute and or more traditional attributes. Experiments over real complex databases to validate comparatively the method shown that it outperformed existing approaches regarding similarity queries with simple conditions and that the developed variations cover different situations regarding queries with composite conditions, allowing fast data retrieval in every situation.

Keywords: Metric access methods. Condition-extended similarity queries. Multimedia databases.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

bfr	Blocking Factor
CBR	Content-Based Retrieval
ck-NN	Consulta aos Vizinhos Mais Próximos com Condições
cx-Sim	Condition-eXtended Similarity tree
DICOM	Digital Imaging and Communications in Medicine
DIR-tree	Document similarity enhanced Inverted file R-tree
GBDI	Grupo de Banco de Dados e Imagens
ICMC	Instituto de Matemática e Ciência da Computação
IR^2-tree	Information Retrieval R-tree
ISAM	Indexed Sequential Access Method
JIDM	Journal of Information and Data Management
k-NN	Consulta aos Vizinhos Mais Próximos
LSH	Locality-Sensitive Hashing
MA	Método de Acesso
MAE	Método de Acesso Espacial
MAEP	Método de Acesso Espacial Pontual
MAENP	Método de Acesso Espacial Não-Pontual
MAM	Método de Acesso Métrico
MST	Minimum Spanning Tree
PLSH	Parallel Locality-Sensitive Hashing
Rq	Range query
SBBD	Simpósio Brasileiro de Banco de Dados
SGDB	Sistema Gerenciador de Banco de Dados
S2I	Spatial Inverted Index
USP	Universidade de São Paulo

SUMÁRIO

1 INTRODUÇÃO

O avanço tecnológico dos dispositivos eletrônicos nos últimos anos, considerando a popularização dos dispositivos móveis (sensores, smartphones, tablets e notebooks), intensificou a necessidade por uma administração efetiva da massa de dados complexos. O conceito de dados complexos neste trabalho refere-se a dados que não são representáveis por dados tradicionais (caracteres alfanuméricos), tais como dados multimídia e científicos, séries temporais e dados georreferenciados. Considerando-se apenas as mídias sociais mais populares da atualidade (Facebook, Instagram, LinkedIn e Youtube), pode-se verificar que o volume de dados complexos que trafega pela internet por minuto é muito grande. A área médica é outro exemplo onde são geradas enormes quantidades de dados de diversos tipos (imagens, vídeos, séries temporais etc), por meio da crescente disponibilidade de equipamentos para variados tipos de exames médicos, como radiografias digitais, eletrocardiogramas, tomografias computadorizadas e outros.

Planejar o armazenamento e a recuperação dos dados de maneira eficiente e eficaz ainda é um desafio de pesquisa. A eficiência atingida pelos Sistemas Gerenciadores de Banco de Dados (SGBDs) da atualidade para manusear dados tradicionais tem sido motivação para a inclusão de mecanismos para manusear dados complexos nestes sistemas, permitindo um gerenciamento integrado de dados tradicionais e complexos. Dentre os inúmeros desafios para realizar tal integração, o problema abordado nesta dissertação é permitir a execução eficiente de consultas sobre dados complexos utilizando condições adicionais, tipicamente envolvendo dados tradicionais.

1.1 Motivação

Os operadores relacionais ($<$, $\leq$, $\geq$, $>$) não são comumente aplicáveis a dados complexos, pois é impossível ordená-los diretamente, a menos que o operador de comparação seja aplicado a uma informação associada – não complexa – (nome, data, etc.) ou a uma característica classificável, extraída dos dados complexos [1]. Por exemplo, de uma forma geral, não faz sentido verificar se uma imagem é maior ou menor do que outra, considerando-se a semântica do conteúdo visual representado. Portanto, as buscas por dados complexos são comumente executadas através de um operador baseado em similaridade, cujo objetivo é comparar dados considerando-se relações de similaridade definidas sobre o conteúdo do dado [2]. A aplicabilidade destes operadores em um domínio de dados específico requer a existência de uma função que calcula a similaridade entre pares de elementos. Esta função é geralmente uma função de distância, cujo resultado é o inverso da similaridade entre os elementos comparados, onde 0 (zero) denota elementos idênticos.

Consultas por similaridade têm sido amplamente exploradas na literatura. Entretanto, quando aplicadas sobre dados complexos, a maioria das abordagens as considera como operações isoladas, não se beneficiando de condições adicionais de filtragem. Em geral, dados complexos possuem metadados que os descrevem e/ou são associados a outros dados em um banco de dados. Desta forma, é interessante proporcionar meios de definir consultas envolvendo condições adicionais às condições baseadas em similaridade no predicado de busca e meios para executar essas consultas de maneira eficiente.

A combinação de atributos complexos e tradicionais na mesma instrução possibilita a construção de consultas de maior valor agregado. Considerando-se a consulta *Q1* a seguir, cujo domínio dos dados complexos é multimídia, percebe-se que analisando-se apenas o conteúdo do dado complexo (imagens de ultrasonografia – Figura 1) a condição adicional (idade) presente na consulta não pode ser resolvida.

Q1 – Retorne as k imagens de ultrassonografia mais similares à imagem de consulta i_q dentre exames realizados em pacientes de 35 a 40 anos

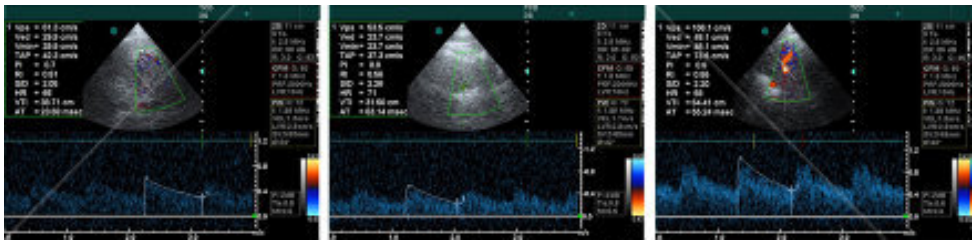


Figura 1 – Representação de similaridade com dado multimídia.

Um segundo exemplo está representado na consulta *Q2*, cujo domínio dos dados complexos é geográfico (Figura 2), onde constata-se novamente que analisando-se os dados de localização geográfica as condições adicionais (tempo e dispositivo móvel) presentes na consulta não podem ser resolvidas.

Q2 – Retorne as postagens de uma mídia social submetidas através de um smartphone, nas últimas 24 horas e num raio de 1 km da coordenada geográfica do hotel Golden Tulip Recife Palace

Esses exemplos mostram que instruções compostas por expressões de busca elaboradas envolvendo atributos complexos e tradicionais possibilitam representar consultas úteis em diversos domínios de aplicação. Embora algumas variações de consultas combinando condições baseadas em atributos complexos e em atributos associados sejam

Figura 2 – Representação da região geográfica de interesse para a consulta *Q2*.

executáveis em SGBDs enriquecidos com funções que permitam representar consultas por similaridade (por exemplo, a abordagem adotada pelo módulo FMI-SiR [3]), o desempenho dessas consultas é reduzido por não possuírem estruturas de indexação compostas que contenham atributos complexos e tradicionais, exigindo acessos adicionais à tabela de dados para responder à consulta. Há algumas estruturas de indexação propostas para executar consultas combinando similaridade e atributos tradicionais, no entanto os critérios de busca tradicionais são limitados a condições baseadas em palavras-chave.

O objetivo desta dissertação foi propor um novo método de acesso métrico que possibilitasse operar dados complexos e tradicionais em conjunto no predicado de consultas a fim de executar de um modo eficiente consultas por similaridade estendida com condições gerais sobre atributos tradicionais (i.e. não limitadas a condições baseadas em palavras-chave). A proposta desenvolvida, denominada *cx-Sim tree* (*condition-extended Similarity tree* – árvore para similaridade estendida com condições), é uma abordagem híbrida que permite testar diretamente no índice ambos os critérios – o condicional e a similaridade – beneficiando-se também das possibilidades de filtragem sobre os atributos de tipos de dados tradicionais. A dissertação apresenta os detalhes da estrutura proposta, bem como algumas variações de implementação, e resultados de experimentos executados com conjuntos de dados reais. Os resultados alcançados nos experimentos mostram que a *cx-Sim tree* teve desempenho superior às abordagens existentes para execução de consultas por similaridade com condições gerais, confirmando a sua eficiência.

1.2 Organização do trabalho

Esta dissertação está organizada como apresentado a seguir:

- o Capítulo 2 apresenta a fundamentação teórica sobre a execução de consultas por similaridade;
- o Capítulo 3 apresenta as principais estruturas de indexação para acelerar o processamento de consultas por similaridade, incluindo abordagens para execução de consultas com condições adicionais;
- o Capítulo 4 apresenta o método de acesso métrico proposto *cx-Sim tree* e variações de implementação desenvolvidas;
- o Capítulo 5 mostra os experimentos executados para comparar o método proposto com as soluções existentes e discute os resultados obtidos;
- finalmente, o Capítulo 6 apresenta as conclusões e perspectivas futuras para a pesquisa.

2 FUNDAMENTOS SOBRE CONSULTAS COM DADOS COMPLEXOS

À medida que o uso de tipos não tradicionais de dados continua em expansão (tais como imagens, vídeos, XML e outros), SGBDs tradicionais encontram-se em constante processo de atualização a fim de processá-los eficientemente. No contexto deste trabalho, tipos de dados que não são representáveis por dados tradicionais (caracteres alfanuméricos) são denominados dados complexos, que demandam operadores específicos de comparação.

Tipos de dados tradicionais, tais como números, caracteres e datas, podem ser manipulados por meio de operadores relacionais de comparação ($<$, $\leq$, $\geq$, $>$), devido a existência de uma relação de ordem total no domínio dos dados. A relação de ordem possibilita comparar qualquer par de objetos distintos e definir a precedência de um sobre o outro. Exemplos de relação de ordem total são: a relação de ordem numérica em domínios numéricos e a relação de ordem lexicográfica em domínios de cadeias curtas de caracteres. Por outro lado os dados complexos, tais como dados multimídia e espaciais, não têm uma relação de ordem total definida e, além disso, os operadores $=$ e $\neq$ são praticamente sem utilidade. Isto é evidenciado, por exemplo, na gravação da mesma situação sobre diferentes condições de posicionamento e iluminação: as imagens podem ser visualmente similares mas a possibilidade de serem exatamente iguais é praticamente nula. Sendo assim, ter condições de aferir o grau de similaridade nessas situações é crucial.

O capítulo em questão aborda os fundamentos das consultas sobre dados complexos, apresentando na Seção 2.1 as consultas por similaridade, considerando suas variações mais comuns (Seção 2.1.1) e uma extensão que considera condições de busca adicionais (Seção 2.1.2). Na Seção 2.2 é apresentada a noção de espaço de similaridade, considerando o Espaço Vetorial (Seção 2.2.1) e o Espaço Métrico (Seção 2.2.2).

2.1 Consultas por Similaridade

As consultas por similaridade mostram-se como a solução mais apropriada para a recuperação de dados complexos. De uma forma geral, uma consulta por similaridade é uma busca dentre elementos pertencentes a um espaço de similaridade (seção 2.2) que, segundo um critério de similaridade, sejam mais parecidos com um ***elemento de referência*** s_q, também chamado de *elemento de consulta*. Essa busca ocorre aos pares, ou seja, cada elemento do espaço de similaridade é comparado diretamente ao s_q e, como resultado, são selecionados apenas os que atendem ao critério de similaridade. A comparação entre o par de elementos ocorre a partir das informações inerentes ao conteúdo

do elemento de referência e os elementos armazenados, o que permite efetuar uma recuperação baseada em conteúdo (***Content-Based Retrieval*** – ***CBR***). Dessa maneira, a consulta é executada considerando o significado do conteúdo intrínseco dos elementos e não as metainformações associadas aos mesmos.

Para aferir a similaridade entre os dados são necessários, previamente, extrair um vetor de características de cada elemento, cuja função é representar o conteúdo do dado em si e proporcionar a execução das comparações (Figura 4). Por exemplo, se o dado complexo é uma imagem, as técnicas de recuperação por conteúdo (*Content-Based Image Retrieval*) objetivam consultar imagens com base no seu conteúdo visual e, portanto, utilizam-se de técnicas de processamento de imagens para a extração de características. É necessário, também, definir uma ***função de distância*** – δ, também chamada *função de dissimilaridade*, que afira o grau de (dis)similaridade entre pares de elementos. Prezando pela qualidade do resultado apresentado, é importante que estes dois pré-requisitos sejam atendidos por especialistas no domínio em questão.

Uma vez atendidos os pré-requisitos acima, a aferição do grau de similaridade entre pares de dados complexos é obtida por meio da aplicação da função de distância sobre os respectivos vetores de características. Como resultado esta função retorna um número que indica quão similaridades são os elementos comparados. Por definição, resultados iguais a 0 (zero) denotam elementos idênticos, resultados muito próximos a 0 (zero) denotam elementos similares e, quanto mais distantes de 0 (zero) forem, maior é o grau de dissimilaridade dos elementos.

Na seção 2.1.1 serão apresentadas as variações mais comuns de consultas por similaridade.

2.1.1 Tipos Básicos de Consultas por Similaridade

Existem diversas variações de consultas por similaridade, incluindo seleções, junções por similaridade e consultas por similaridade agregada. As variações mais comuns são as consultas por abrangência (*Range queries*) e aos vizinhos mais próximos (*k-Nearest Neighbor queries*)[4].

Para ambos os casos, considere um domínio de dados complexos $\mathbb{S}$, um conjunto de dados $S \subseteq \mathbb{S}$, tal que $S = \{s_1, s_2, s_3, ..., s_n\}$, um elemento de referência $s_q \in \mathbb{S}$ e uma função de distância δ definida sobre $\mathbb{S}$.

2.1.1.1 Consulta por Abrangência

Considerando um raio de cobertura ξ, uma consulta por abrangência (***Range query*** – Rq) tem por objetivo retornar todo elemento $s_i \in S$ cuja distância de s_q seja menor ou igual a uma distância máxima ξ [5]. Formalmente, objetiva-se retornar o sub-

conjunto $R \subseteq S$ que atenda a:

$$Rq(\delta, s_q, \xi) = R = \{s_i | \delta(s_i, s_q) \leq \xi\} \tag{2.1}$$

A Figura 3(a) ilustra uma Rq com a função de distância Euclidiana L_2, onde são retornados os elementos localizados na região delimitada pelo raio ξ a partir do elemento de consulta s_q. Um exemplo de consulta que pode ser representado por uma Rq é a consulta *Q3*, onde: $s_q = C$, o universo $\mathbb{S}$ são coordenadas geográficas, o subconjunto $S \subseteq \mathbb{S}$ é uma base de dados de pizzarias e o raio de busca é $\xi = 5$ Km.

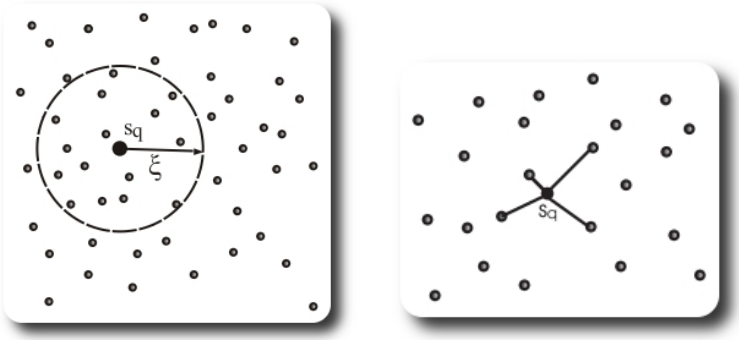
Q3 – Retorne as pizzarias que encontram-se num raio de até 5 Km da coordenada C

2.1.1.2 Consulta aos k-Vizinhos mais Próximos

Considerando um número inteiro positivo k, uma consulta aos k-vizinhos mais próximos (***k*-Nearest Neighbor query*** – k-NNq) recupera os k elementos $s_i \in S$ mais similares a s_q no que diz respeito a δ [5]. Formalmente, objetiva-se retornar o subconjunto $K \subseteq S$ que atenda a:

$$kNNq(\delta, s_q, k) = K = \{s_i \in S | \forall s_j \in S - K, \delta(s_i, s_q) \leq \delta(s_j, s_q)\}, \tag{2.2}$$

A Figura 3(b) demonstra uma k-NNq com a função de distância Euclidiana L_2, onde são retornados os $k = 4$ elementos mais próximos (ilustrativamente conectados) ao elemento de consulta s_q.



(a) (b)

Figura 3 – Exemplos de consultas por similaridade no espaço $\mathbb{R}^2$: (a) Range Query e (b) k-Nearest Neighbor query.

Consultas k-NNq são frequentemente usadas em sistemas de recuperação por conteúdo. Isto se deve ao fato de que, em geral, é mais fácil para o usuário definir o tamanho desejado do resultado (k) do que o limiar de dissimilaridade adequado à sua intenção de busca (ξ), pois é preciso conhecer muito bem o conjunto de dados para definir o valor de ξ para cada busca [6] [7]. Um exemplo de k-NNq é a consulta *Q4*, onde: $s_q = I$ e $k = 10$.

Q4 – Em uma base de imagens de ultrassonografia, retorne as 10 imagens mais similares à imagem de referência I

2.1.2 Consultas por Similaridade Estendidas com Condições

Uma consulta por similaridade pode ser estendida com condições de busca adicionais, definidas sobre atributos associados aos dados complexos. O tipo trivial de condição de busca é a baseada em palavras-chave, que verifica se um elemento é marcado com um dado conjunto de palavras-chave. No entanto, existem muitas consultas que necessitam de condições gerais, isto é, não somente baseadas em palavras-chave. As consultas apresentadas no Capítulo 1 são exemplos de consultas por similaridade estendidas com condições:

- *Q1 – Retorne as k imagens de ultrassonografia mais similares à imagem de consulta i_q dentre exames realizados em pacientes de 35 a 40 anos*;
- *Q2 – Retorne as postagens de uma mídia social submetidas através de um smartphone, nas últimas 24 horas e num raio de 1 km da coordenada geográfica do hotel Golden Tulip Recife Palace.*

Uma operação de similaridade Rq, estendida através de uma condição adicional composta por uma típica expressão condicional do operador de seleção da álgebra relacional (σ), pode ser representada por uma seleção conjuntiva (ou por seleções em cascata) e otimizada com o uso de regras de equivalência válidas para a seleção relacional, como também para as consultas por abrangência [8]. Em contrapartida, considerando-se uma k-NNq como a operação de similaridade, esta hipótese não é válida porque a k-NNq não é comutativa com a seleção relacional.

Kaster[9] propôs que uma consulta aos k-vizinhos mais próximos pudesse ser estendida através da inclusão de condições adicionais que modificam a busca, introduzindo uma nova operação de busca por similaridade que estende o operador básico k-NN, chamada consulta aos k-vizinhos mais próximos estendida por condições (*condition-extended k-Nearest Neighbor query* – ${}_c k$-NNq). Uma ${}_c k$-NNq trata-se de uma busca exata que recupera os k elementos mais próximos a um elemento de referência que satisfaz a condição

adicional c fornecida. Esta nova operação permite responder consultas por similaridade não representáveis até então, como, por exemplo, as consultas *Q1* e *Q2*. A condição c pode ser sobre atributos de diversos tipos, sejam eles dados complexos ou tradicionais, permitindo aplicar operadores de comparação relacional e também combinação de palavras-chave. Ele também propôs algoritmos para executar variações de ${}_c k$-NNq com execução sequencial e usando métodos de acesso métrico para indexar dados complexos, discutidos na Seção 3.4.

2.2 Representações de Espaço de Similaridade

Um espaço de similaridade é o espaço em que dados complexos são mapeados com base em suas caracterítisticas intrínsecas e na função de dissimilaridade considerada. O dado georreferenciado, por exemplo, é representado como um ponto, linha ou polígono no espaço geográfico definido pela latitude e longitude do elemento, ou de suas partes, pois um polígono geralmente é representado como uma sequência de pontos [10]. O dado multimídia, por sua vez, é representado por um conjunto de características extraídas do conteúdo do dado (Figura 4). Apesar de ser considerada uma representação limitada para definir a semântica do dado, esta sumarização do conteúdo denominada ***vetor de características*** tem por objetivo o representar numericamente. Sendo assim, um dado multimídia pode ser representado como um ponto no espaço de características, de acordo com o valor extraído para cada uma das características. A extração de atributos relevantes que comporão a representação numérica do dado complexo, ou seja, o vetor de características, é executada por algoritmos conhecidos como ***extratores de características***.

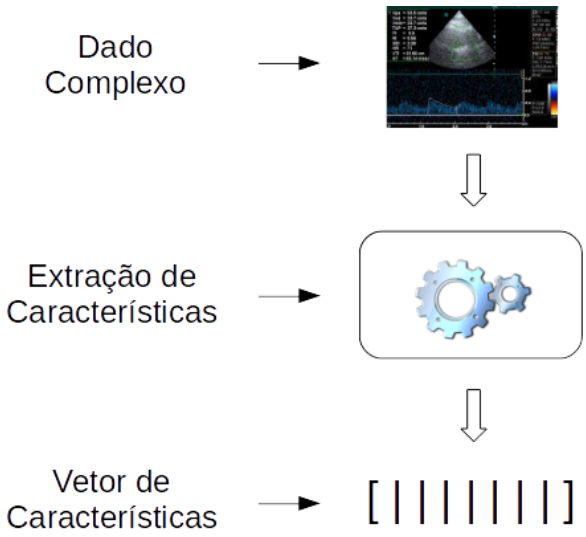


Figura 4 – Vetor de características: a representação numérica de um dado complexo.

A extração de características ocorre de maneira particular para cada um dos possíveis dados multimídia. As imagens usualmente são representadas por padrões de cor,

textura e forma. As primeiras formas de extração de características foram na sua maioria de escopo global, porém as pesquisas mais recentes têm se concentrado em fazer a extração a partir de regiões das imagens (escopo local) [11]. A representação de imagens baseada em regiões aproxima-se mais da percepção humana do que a representação de imagens inteiras [12]; Para possibilitar a extração de características a partir de regiões, faz-se necessário uma fase anterior de segmentação da imagem para a identificação dos elementos presentes na cena. Uma vez segmentada, a extração de caracterítisticas prossegue por cor, textura ou forma, tendo como exemplos de extratores respectivamente: o histograma normalizado [13], a transformada de *wavelet* [14] e os momentos de Zernike [15].

As relações de similaridade entre os dados passam a ser possíveis a partir da representação de dados complexos em um espaço de características. Segundo Gauker[16], na filosofia e na psicologia a mente humana define um hiperespaço de representação. Neste espaço, as dimensões representam formas através das quais os objetos podem ser distinguidos, da seguinte maneira: pontos representam objetos que podem existir e a distância entre eles é inversamente proporcional ao grau de similaridade. Dessa maneira, é possível responder a consultas por dados multimídia que sejam similares a um determinado elemento de consulta (s_q) analisando as distâncias no espaço de similaridade. A aferição do grau de (dis)similaridade ocorre através da aplicação das funções de distância às representações numéricas dos dados complexos, ou seja, aos pares de vetores de características. A combinação de vetores de características e funções de distância constituem *descritores*, que determinam o valor de similaridade entre elementos [17]. Dessa maneira, o par $\langle \mathbb{S}, \delta \rangle$, onde $\mathbb{S}$ é o domínio do vetor de características e δ é a função de distância, constitui-se numa ***instância do espaço de similaridade*** dentre todas as possíveis instâncias do espaço de similaridade.

A relevância do resultado de uma consulta por similaridade depende da instância do espaço de similaridade utilizada, sendo de extrema importância que a definição deste seja oriunda de um profissional experiente no domínio em questão. Para o domínio geográfico, por exemplo, a noção de similaridade é tratada pela distância entre pontos da superfície terrestre, aplicando a distância Euclidiana (L_2) sobre as coordenadas geográficas (vetores de características). Em relação ao domínio multimídia, segundo Bugatti, Traina e Traina Jr.[18], resultados experimentais mostram que encontrar a melhor combinação entre vetor de características e função de distância aprimora a precisão das consultas. Tal situação é análoga a subjetividade da percepção humana a uma cena, pois pessoas diferentes podem ter percepções diferentes a respeito de uma imagem ou vídeo, dependendo do seu interesse e conhecimento prévio. Portanto, a instância "ideal" é referenciada na literatura como espaço (de similaridade) semântico [19].

Na maior parte dos trabalhos encontrados na literatura o espaço de similaridade é um espaço vetorial ou um espaço métrico. As seções 2.2.1 e 2.2.2 apresentam, respec-

tivamente, os fundamentos desses espaços no que tange à representação e execução de consultas por similaridade.

2.2.1 Espaço Vetorial

Considerando que o domínio $\mathbb{S}$ é formado por elementos cuja representação são vetores numéricos, o espaço de similaridade pode ser um ***Espaço Vetorial*** *(com Dimensão Finita)*. A representação dos objetos que compõem um espaço n-dimensional resume-se a n coordenadas de valores reais $[x_1, x_2, x_3, ..., x_n]$, comparáveis com o uso de diversas funções de distância relatadas na literatura [20], dentre elas as da família **Minkowski** – L_p, definidas por:

$$L_p((x_1, ..., x_n), (y_1, ..., y_n)) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \tag{2.3}$$

As diferentes áreas de abrangência de cada métrica da família L_p estão ilustradas na Figura 5. A métrica L_1 (***Manhattan*** ou *City Block*) corresponde ao somatório do módulo das diferenças das coordenadas. Como representado em 5(a), o conjunto de pontos com a mesma distância ξ forma um losângulo com os diâmetros paralelos aos eixos das coordenadas. A métrica L_2 (***Euclidiana***), é a distância amplamente utilizada para calcular distância de vetores e, além disso, como representado em 5(b), o seu conjunto de pontos com a mesma distância ξ forma uma circunferência. A métrica L_∞ (***Chebychev***), trata-se do limite da função 2.3 quando p tende a infinito, sendo definida pela Equação 2.4. Como representado em 5(c), o seu conjunto de pontos com a mesma distância ξ forma um quadrado com os lados paralelos aos eixos das coordenadas.

$$L_\infty((x_1, ..., x_n), (y_1, ..., y_n)) = \max_{i=1}^{n} |x_i - y_i| \tag{2.4}$$

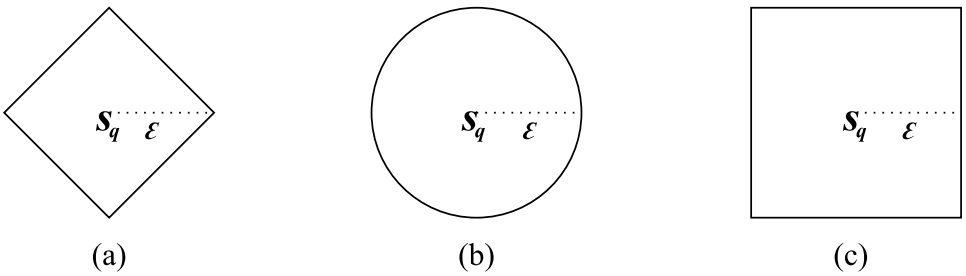


Figura 5 – Áreas de abrangência para as distintas métricas da família L_p (L_1, L_2 e L_∞ respectivamente), considerando um raio de cobertura ξ e o elemento de referência s_q.

A noção de distância no espaço vetorial pode utilizar de propriedades geométricas, tais como os ângulos, as áreas e as distâncias. Estas propriedades são muito usadas nas

propostas de estruturas de indexação e algoritmos para agilizar consultas sobre dados imersos em espaços vetoriais.

2.2.2 Espaço Métrico

Para que a noção de similaridade possa ser aplicada via algoritmos a dados complexos, faz-se necessário o estabelecimento de regras que regem o comportamento do domínio dos dados e sua relação com a função de distância. De acordo com Chávez et al.[4], dado um universo de possíveis elementos $\mathbb{S}$, ou seja, o domínio dos dados, $S = \{s_1, s_2, s_3, ..., s_n\}$ um subconjunto de $\mathbb{S}$ onde as consultas serão efetuadas, e δ uma função:

$$\delta : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}^+ \tag{2.5}$$

responsável por aferir a distância entre elementos, de modo que, quanto mais próximos dois objetos estão, mais similares eles são considerados [2], é possível definir consultas por similaridade.

Em muitas situações, o domínio $\mathbb{S}$ pode ser um espaço vetorial $\mathbb{R}^n$. Entretanto, há dados complexos cujos vetores de características não têm dimensionalidade definida (e.g. palavras) ou cuja dimensionalidade dos elementos não é fixa (e.g. extratores que identificam regiões em uma imagem e extraem características dessas regiões podem ter diferentes dimensionalidades, de acordo com o número de regiões identificadas). Esses exemplos de dados complexos não podem ser representados em um espaço vetorial, mas sim em um espaço métrico.

Um ***Espaço Métrico*** $\mathbb{M}$ é definido por um par $\langle \mathbb{S}, \delta \rangle$, onde $\mathbb{S}$ é um domínio de dados e δ é uma função, denominada *Função de Distância Métrica* ou apenas *Métrica*, que deve atender às propriedades a seguir, $\forall s_1, s_2, s_3 \in \mathbb{S}$:

P1 – Simetria

$$\delta(s_1, s_2) = \delta(s_2, s_1)$$

P2 – Não negatividade

$$0 < \delta(s_1, s_2) < \infty, \text{ se } s_1 \neq s_2 \text{ e } \delta(s_1, s_1) = 0$$

P3 – Desigualdade triangular

$$\delta(s_1, s_2) \leq \delta(s_1, s_3) + \delta(s_3, s_2)$$

Tais propriedades, principalmente a desigualdade triangular, proporcionam a execução de consultas de maneira otimizada através do uso de técnicas de indexação. Estas por sua vez, podam (pela distância) elementos cujo grau de similaridade ao elemento representativo não atinja um valor mínimo.

Como as funções L_1, L_2 da família Minkowski são métricas, um espaço vetorial $\mathbb{R}^n$ associado a uma destas funções de distância forma um espaço métrico. Isto mostra que um espaço métrico permite representar dados complexos cujos vetores de características são dimensionais. Contudo, espaços métricos são capazes de representar também dados que não tem dimensionalidade definida ou fixa. Por exemplo, um domínio de palavras associado à métrica de **Levenshtein** (L_{Edit}) não tem dimensionalidade definida, mas forma um espaço métrico. A métrica $L_{Edit}(s_1, s_2)$ [21] é utilizada para aferir a similaridade entre pares de cadeias de caracteres. Ela retorna a menor quantidade de operações de edição de caracteres (inserções, remoções e substituições) necessárias para transformar a cadeia de caracteres s_1 em s_2.

2.3 Considerações Finais

O capítulo em questão apresentou o modo mais adequado e difundido para se responder a buscas em base de dados complexos: as consultas por similaridade. Foram abordadas tanto as suas variações mais comuns (as consultas por abrangência e aos vizinhos mais próximos), quanto uma nova operação capaz de estender as consultas por similaridade com condições adicionais gerais, a ${}_c k$-NNq. Esta última, por adicionar atributos tradicionais aos dados complexos que compõem a estrutura de indexação, possibilita responder a consultas por similaridade não representáveis até então, como também aplicar operadores de comparação relacional aos dados indexados. Além disso, foi apresentada a noção de espaço de similaridade, ou seja, o espaço onde os dados complexos são mapeados tomando como base suas características intrínsecas e uma função que afira a dissimilaridades entre os dados.

O capítulo 3 abordará métodos de acessos e algoritmos para consulta por similaridade sobre dados complexos, bem como estruturas de dados para integração desses dados com dados de tipos tracionais.

3 MÉTODOS DE ACESSO E ALGORITMOS PARA CONSULTAS POR SIMILARIDADE ENVOLVENDO DADOS COMPLEXOS

As consultas por similaridade podem ser respondidas independente da existência de uma estrutura de indexação previamente criada. Não existindo índice, o conjunto resposta pode ser construído a partir de uma busca sequencial onde todos os elementos do conjunto de dados serão examinados. Segundo Chávez et al.[4], um *algoritmo de indexação* é um procedimento offline para construir previamente uma estrutura de dados (chamada índice) desenvolvida para poupar cálculos de distância ao responder a consultas por similaridade. Essa estrutura de dados pode ser custosa para construir, mas isto será amortizado ao se poupar a execução de cálculos de distância sobre várias consultas na base de dados. O objetivo é projetar algoritmos de indexação eficientes para reduzir o número de avaliações de distância.

Vários tipos de dados complexos são representáveis como vetores de características multidimensionais extraídas a partir do seu conteúdo. Desta forma, eles podem ser representados tanto em um espaço vetorial quanto em um espaço métrico e, portanto, indexados utilizando duas classes de métodos de acesso: Métodos de Acesso Espaciais (MAEs), que pressupõem que os dados estão imersos em um espaço vetorial; ou Métodos de Acesso Métricos (MAMs), que manipulam dados em um espaço métrico. Considerando-se consultas por similaridade estendidas com condições, a maioria dessas consultas em aplicações envolvendo dados complexos utiliza condições baseadas em atributos de tipos tradicionais. Há várias estruturas de indexação para dados de tipos tradicionais, com destaque para a B^+-*tree* e suas variantes e índices *hash*, que podem ser usados para acelerar a execução de consultas por similaridade com condições envolvendo dados tradicionais.

Este capítulo apresenta os principais métodos de Acesso para dados em domínios com relação de ordem total (dados tradicionais) e em domínios multidimensionais (dados complexos). Inicialmente são apresentados métodos de acesso para dados tradicionais (Seção 3.1), seguidos de métodos de acesso espaciais (Seção 3.2), métricos (Seção 3.3) e, finalmente, para a execução de consultas por similaridade com condições (Seção 3.4).

3.1 Métodos de Acesso para Dados Tradicionais

Existem vários métodos de acesso para dados tradicionais, ou seja, para dados cujo domínio apresenta relação de ordem total. Dentre eles estão os índices para arquivos ordenados, *Hashing* e *B-tree* com suas variações, sendo esta última a mais importante.

Segundo Elmasri e Navathe[22], as técnicas de ***Hashing*** são caracterizadas por

proporcionar um acesso muito rápido aos registros para condições de busca por igualdade na chave, também denominada **campo de *hash***. Basicamente, estas técnicas fornecem uma função *h()*, chamada função ***hash***, que, aplicada ao valor do campo de *hash* de um registro, gere o endereço do bloco de disco no qual o registro está armazenado. No melhor caso um acesso pode ter complexidade *O(1)*, mas, caso a função *hash* gere muitas colisões a complexidade passa a ser *O(n)*. Este aumento justifica-se dada a necessidade de uma busca sequencial pelos n elementos que compõem o *bucket* apontado por *h()*. As técnicas de *Hashing* foram desconsideradas nesta dissertação como possível solução para indexação dos dados tradicionais na estrutura proposta no Capítulo 4, pois não atendem de forma eficiente a buscas por intervalos.

A *B-tree*, o mais difundido índice para arquivos ordenados, foi proposto por Bayer e McCreight[23]. Esta trata-se de uma árvore multivias balanceada em relação a altura – h, ou seja, todas as buscas têm o mesmo comprimento da raiz até as folhas – $h \leq O(\log n)$. Consequentemente, suas operações são de complexidade logarítmica. Seus algoritmos de inserção e remoção garantem que, com exceção da raiz, todo nó tenha no mínimo 50% da sua cardinalidade (quantidade de entradas) ocupada. O nó raiz, por sua vez, tem no mínimo uma entrada. Em relação ao grau (número de subárvores), na *B-tree* este é igual a sua cardinalidade mais um.

O procedimento de inserção começa com um percurso da raiz até a folha, onde, nesta última, o novo elemento será adicionado. Com isso, o crescimento da estrutura é *bottom-up*, ou seja, de baixo para cima. A inserção ocorre de maneira ordenada, garantindo que toda subárvore à esquerda de um novo elemento s seja composta por valores menores que s, e que toda subárvore à direta de s seja composta por valores maiores que s.

Caso o nó folha se preencha completamente mediante uma nova inserção (*overflow*), ocorre uma operação de *split* onde um novo nó é criado e as chaves são distribuídas entre o nó corrente, seu pai e o novo nó irmão, de modo que todos fiquem com, ao menos, o preenchimento mínimo (50%). Essa transferência da chave que se encontraria no centro do nó cheio para o nó pai pode ocasionar uma propagação de *splits* que, caso chegue ao nó raiz, ocasionará o aumento da altura da árvore em uma unidade. De maneira análoga, caso a remoção de uma entrada da árvore acarrete em um nó com ocupação menor que o preenchimento mínimo (*underflow*), pode ocorrer desde uma simples redistribuição de chaves entre nós irmãos, até a concatenação de nós irmãos mais a chave central (presente no nó pai). Esta última operação pode ser propagada até a raiz para se garantir o preenchimento mínimo dos nós da árvore e, consequentemente, pode ter sua altura diminuida em uma unidade.

Um dos principais pontos fortes da *B-tree* é a grande capacidade de armazenamento em memória secundária. Segundo Garcia-Molina, Ullman e Widom[24], uma *B-tree* com cardinalidade 255 e grau 256 pode armazenar cerca de 16,7 milhões de chaves em apenas

3 níveis. Consequentemente, se o nó raiz estiver integralmente na memória principal, com apenas 2 acessos a disco é possível recuperar um dado desejado. Em relação à operação de busca, mais dois benefícios podem ser destacados: a redução no número de acessos a disco proporcionada pela poda de subárvores cujas raízes não atendem à condição de busca, como também a redução no número de comparações dentre as chaves em cada nó, devido ao conjunto ser ordenado.

A proposta original da estrutura de dados *B-tree* ganhou duas variações principais, a B^**-tree* e a B^+*-tree*, com contribuições importantes que serão apresentadas a seguir.

A B^**-tree* [25] traz uma melhora em relação a armazenamento, resultando em uma árvore menor (em relação à altura) e, consequentemente, proporciona uma redução no tempo de busca. Tal aprimoramento é devido ao seu algoritmo de inserção trabalhar com um processo de redistribuição local de chaves, postergando o *split* até que 2 nós irmãos estejam completamente cheios. Atingido esse estágio, as chaves dos nós cheios são subdivididas em três novos nós, o que resulta em nós com preenchimento mínimo de 66%.

Já a B^+*-tree* [26], diferentemente das anteriores, apresenta uma distinção entre nós internos e folhas. Os nós internos apresentam apenas uma cópia das chaves necessárias para o percurso da raiz até as folhas, onde realmente são armazenadas as entradas do índice. Os nós folha, além do encadeamento natural da *B-tree*, são ligados da esquerda para a direita em formato de lista encadeada, denominada por Comer[26] como *conjunto sequencial*. Tal organização facilita as buscas sequenciais e por intervalos, o que também atribuiu a estrutura o nome de *método de acesso sequencial indexado* (*Indexed Sequential Access Method – ISAM*).

A B^+*-tree*, por possibilitar acesso eficiente tanto em buscas pontuais quanto em buscas por intervalo, além das demais propriedades da *B-tree*, foi escolhida como a estrutura de indexação para os dados tradicionais do método proposto no Capítulo 4.

3.2 Métodos de Acesso Espaciais

Os Métodos de Acesso Espaciais (MAEs) são utilizados para indexar objetos multidimensionais e, por isso, também são denominados Métodos de Acesso Multidimensionais. Por multidimensional, ou n-dimensional, deve ser entendido o objeto que possa ser localizado por uma série de n coordenadas, sendo n maior que 1. Os dados operados pelos MAEs pertencem ao domínio espacial ou a um espaço de n dimensões, onde cada uma delas é representada por um atributo da relação.

Segundo [27], os MAEs são classificados em:

- **Métodos de Acesso Espaciais Pontuais (MAEPs):** os objetos são considerados como pontos no espaço;

- **Métodos de Acesso Espaciais Não-Pontuais (MAENPs):** os objetos são considerados como regiões, ou seja, apresentam extensão espacial.

O MAEP mais representativo trata-se do ***k-d-B-tree*** [28]. Este é uma combinação da *k-d-tree* [29] com a *B-tree* [23], que divide o espaço em regiões que contemplam aproximadamente o mesmo número de pontos, associando cada região a um nó da árvore. Assim como a *B-tree*, a *k-d-B-tree* é uma árvore multivias, balanceada pela altura e com boa adaptação a distribuição dos dados. Um dos maiores problemas deste método é que a divisão de um nó interno pode ser propagada tanto a árvore acima quanto a de baixo. Tal fato, segundo Gaede e Günther[27], não garante a ocupação mínima de um nó, podendo gerar nós vazios.

Já o MAENP mais representativo é o ***R-tree*** [30], que pode ser visto como uma adaptação do *B-tree* para indexar dados multidimensionais não-pontuais. Os dados também são inseridos nas folhas, mas a intersecção das regiões definidas pelas subárvores de um mesmo nível pode não ser vazia. Nos nós folha, cada entrada possui um código de identificação do objeto (*object identification – OId*) e um retângulo mínimo que o envolve (*Minimum Bounding Rectangle – MBR)*. Os nós internos, por sua vez, são formados pelo MBR que envolve toda a sua subárvore e um ponteiro para ela. Em relação às operações de inserção e remoção que podem ocasionar a reorganização da árvore, o principal objetivo é identificar a subárvore cujo MBR vai aumentar menos de volume, buscando reduzir a sobreposição.

Para dados de baixa dimensionalidade, os MAEs como os apresentados funcionam de maneira eficiente. Embora exista a possibilidade de expansão para dimensões mais altas, eles geralmente requerem tempo e/ou espaço que crescem exponencialmente com o aumento no número das dimensões [4]. Bozkaya e Ozsoyoglu[31], inclusive, citam que experimentos mostraram que o *R-tree* se torna ineficiente para espaços com mais de 20 dimensões. Berchtold, Keim e Kriegel[32] apresentaram uma evolução ao *R-tree* denominado ***X-tree***. Esta propõe o uso de um supernó de tamanho variado para tratar a questão da sobreposição de MBRs quando atingido um grau elevado. Experimentos relatados pelos autores mostram que desempenho do *X-tree* é uma ordem de magnitude superior ao do *R-tree* para dimensionalidades moderadas, embora degrade consideravelmente para altas dimensionalidades. Além disso, uma outra desvantagem importante para se mencionar é que os MAEs não conseguem indexar dados adimensionais.

3.3 Métodos de Acesso Métricos

Como citado anteriormente, dados complexos podem ser indexados utilizando métodos de acesso espaciais. No entanto, estas estruturas degradam rapidamente com o aumento do número de dimensões. Esta seção apresenta os principais Métodos de Acesso

Métricos (MAMs) encontrados na literatura. Além dos MAMs serem capazes de indexar dados dimensionais e dados sem dimensão definida ou fixa, de uma forma geral, eles sofrem menor degradação de desempenho do que MAEs com o aumento da dimensionalidade de dados dimensionais.

Analisando as linhas de pesquisa relacionadas a consultas por similaridades com dados complexos, percebe-se que a maior parte das propostas de métodos de acesso que melhor manipulam dados complexos enquadra-se em duas famílias:

- **Buscas Aproximadas:** família que utiliza heurísticas que proporcionam respostas aproximadas de forma eficiente, isto é, sem a garantia de encontrar todos os elementos que deveriam ser retornados;
- **Buscas Exatas:** família que garante respostas exatas, mas com um custo de busca maior.

Os métodos da família de buscas aproximadas normalmente apresentam rápidos tempos de resposta em detrimento da exatidão dos resultados [6]. Especificamente, estes recuperam resultados aproximados em tempo sublinear [33]. Um dos tipos mais difundidos de MAMs para buscas aproximadas são os baseados em *hashing* sensível à localidade (*Locality-Sensitive Hashing – LSH*). Por exemplo, Slaney e Casey[34] usam *LSH* em projeções k-scalares para buscar rapidamente elementos similares em grandes bases de dados. Esta técnica foi empregada na identificação de páginas web duplicadas, músicas e de objetos dentro de imagens. Kulis, Jain e Grauman[33] apresentam um método que proporciona uma ponderação entre a representação da imagem e a métrica de distância na qualidade dos resultados. Tal ponderação é possibilitada com o ajuste fino nos parâmetros do hash (número de buckets por hash, número de funções de hash, noção de perto e longe, probabilidade de colisão etc) e permite disponibilizar o resultado em tempo sublinear. Sundaram et al.[35], por sua vez, apresentam uma variação de *LSH* denominado *Parallel LSH* (*PLSH*) cujo objetivo é alcançar, por meio de paralelização, maior eficiência e escalabilidade ao executar consultas com múltiplos nós e cores. Apesar de apresentar rápido tempo de resposta, a principal limitação desta família é que o resultado das consultas são aproximados.

A família de métodos para buscas exatas utiliza-se de propriedades do espaço métrico, descrito na Seção 2.2.2, para reduzir o espaço de busca. Existem diversos Métodos de Acesso Métricos (MAMs) para possibilitar acelerar consultas sobre dados complexos, especialmente os dados multimídia. Estas estruturas de indexação possibilitam minimizar o número de comparações (cálculos de distância) e o número de acessos a disco durante o processamento da consulta usando propriedades do espaço métrico, particularmente a desigualdade triangular [36]. Tal resultado é obtido subdividindo o conjunto de dados

em blocos ou nós e, para cada um deles, um elemento é eleito como representativo do bloco/nó que será utilizado para indexar o grupo. Muitas destas estruturas funcionam baseadas no descarte de elementos usando a propriedade de desigualdade triangular.

Existem diversas propostas de métodos de acesso métricos estáticos e dinâmicos. O primeiro MAM dinâmico (isto é, um MAM que não degenera mediante inserções e remoções, mantendo a eficiência de busca) foi a *M-tree* [37]. Esta é uma árvore com crescimento bottom-up e com dois tipos de nós (índice e folha), que permanece balanceada com inserções e remoções, eximindo a necessidade de reorganizações periódicas. Outros MAMs dinâmicos largamente utilizados são a *Slim-tree* [36] e a *DBM-tree* [38]. A *Slim-tree* é uma evolução da *M-tree*, apresentando como melhorias a minimização da sobreposição entre nós e um algoritmo rápido de split baseado em *Minimum Spanning Tree (MST)*, enquanto a *DBM-tree* explora a variação de densidade dos elementos no conjunto de dados permitindo a criação, de uma maneira controlada, de árvores desbalanceadas. Traina Jr. et al.[39] também propuseram uma família de MAMs chamada *Omni Family*, que elege um conjunto de elementos representativos globais (a base *Omni-Foci*) para todo o conjunto de dados e realiza a poda pela distância entre cada elemento e os *focus* na base *omni-foci*. Tal abordagem permite a redução no número de cálculos de distância e acessos a disco, proporcionando alta performance às buscas por similaridade. O estudo de Yousri, Ismail e Kamel[40] minimiza as comparações relativas a dados com alta dimensionalidade pelos nós folha com tamanho variável, o que resultou em uma performance 55% melhor que uma busca tradicional onde a distância é calculada para todos os elementos presentes nos nós folha da estrutura de indexação. Finalmente, McFee e Lanckriet[41] aplicaram variações da *k-d-tree* para identificar a similaridade em conteúdo musical. Esta última abordagem leva a um *tradeoff* entre exatidão e complexidade na execução de uma consulta do tipo k-NN.

3.4 Métodos de Acesso e Algoritmos para Consultas por Similaridade com Condições

Os artigos supracitados tratam a noção de similaridade considerando apenas dados complexos. Ou seja, as buscas por similaridade são consideradas como operações isoladas. Frente a isso, esta seção apresenta trabalhos que apoiam-se na combinação de dados tradicionais e complexos nas estruturas de indexação para executar consultas por similaridade (Figura 6). Esses trabalhos podem ser divididos em duas classes de acordo com o tipo de condição associada a consulta por similaridade: trabalhos que suportam apenas condições baseadas em palavras-chave e trabalhos que suportam condições gerais.

Os trabalhos que suportam somente condições baseadas em palavras-chave empregam arquivos de assinatura ou arquivos invertidos. Um arquivo de assinatura é um

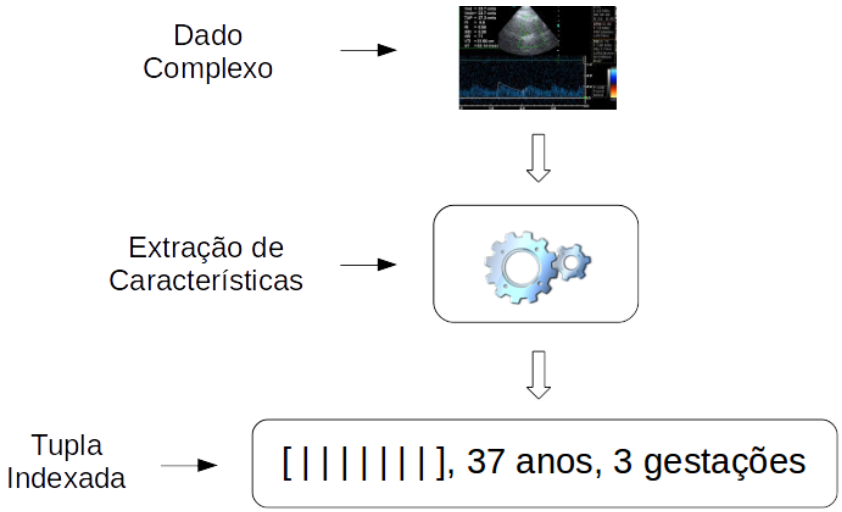


Figura 6 – Estrutura de indexação que combina dados complexos e dados tradicionais.

bitmap associado a um nó da árvore onde cada bit corresponde a uma palavra-chave. Um bit 1 indica que há ao menos um elemento que tem a respectiva palavra-chave no nó ou na subárvore cuja raíz é o nó em questão. Já um bit 0 indica que o elemento não existe. O arquivo de assinatura de um nó índice é dado pela aplicação da operação OR, bit a bit, entre os arquivos de assinatura dos nós filhos, permitindo podar subárvores ou nós que não têm elementos com a(s) palavra(s)-chave desejada(s) durante a busca. Exemplos de estruturas que empregam arquivos de assinatura são a *RS-tree* [42], a *SPY-TEC+* [43] e a $IR^2 - tree$ *(Information Retrieval R-tree)* [44]. Um arquivo invertido é uma estrutura eficiente para a recuperação de informações textuais que proporciona o estabelecimento de um *rank* das informações recuperadas baseado nas palavras-chave. Exemplos de estruturas que empregam arquivos invertidos para acelerar consultas espaciais baseadas em palavras-chave são: a *DIR-tree (Document similarity enhanced Inverted file R-tree)* [45] e a *S2I (Spatial Inverted Index)* [46]. Outra estrutura que emprega arquivos invertidos é a *IQ-tree* [47], cujo foco é nas consultas espaciais contínuas baseadas em palavras-chave sobre *streams* de dados.

No que diz respeito a trabalhos com suporte a consultas por similaridade com condições gerais, não foi encontrado na literatura método de acesso desenvolvido especificamente para responder a tais consultas. O trabalho de Kaster et al.[3] propôs novos algoritmos a partir de um método de acesso existente que aprimora a performance das consultas por similaridade com condições gerais. Estes algoritmos respondem a consultas por similaridade estendidas com condições (Seção 2.1.2) e foram implementados na *Slim-tree* (embora possam ser implementados também em outras estruturas, tais como a *R-tree* ou a *M-tree*), em duas abordagens: *Table-Slim* e *Covering-Slim*.

A abordagem ***Table-Slim*** [3] (Figura 7) utiliza-se de uma *Slim-tree* sem modi-

ficações para armazenar o vetor de características e o identificador de linha (*rowId*) de cada tupla indexada. A execução de uma consulta por similaridade atravessa a estrutura podando os nós/elementos que não satisfazem o critério de similaridade. Se o algoritmo alcança uma entrada no índice que satisfaz o critério de similaridade, ele necessita checar a condição de busca adicional. Ao fazê-lo, o algoritmo acessa a tabela de dados para recuperar a tupla referenciada pelo *rowId* armazenado na entrada, pois o atributo requisitado na condição não está dentro do índice. Se o elemento satisfaz a condição adicional, ele é incluído no conjunto resposta e a busca continua. A principal fragilidade da abordagem *Table-Slim* está no fato de demandar alto número de acessos a disco, no entanto, ele é mais rápido que a busca sequencial para situações de seletividade moderada da condição adicional. Além disso, o método *Table-Slim* só necessita do índice no atributo complexo, sendo capaz de responder a consultas cuja condição envolve qualquer outro atributo da tabela de dados.

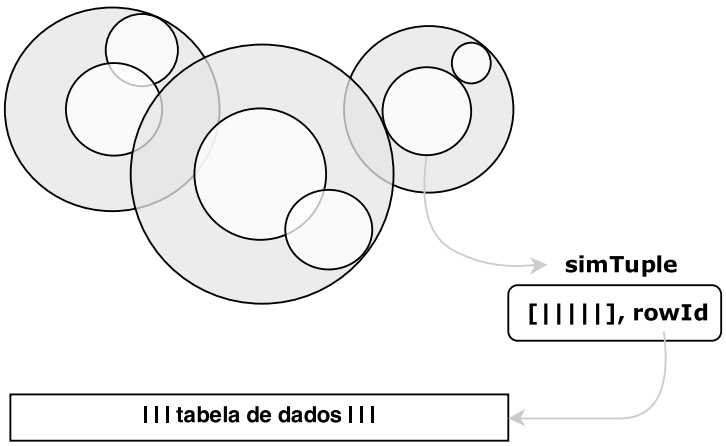


Figura 7 – Representação da abordagem *Table-Slim*, demonstrando o fluxo entre a *Slim-tree* (representada por seus arcos/nós), uma tupla indexada e a tabela de dados de onde os atributos tradicionais são coletados para validação.

A abordagem ***Covering-Slim*** [3] (Figura 8) modifica a *Slim-tree* para ser um índice de cobertura para uma consulta do tipo ${}_c k$-NN. Um índice de cobertura [9] é tal que pode responder a uma consulta sem realizar pesquisas adicionais na tabela de dados. Isso torna-se possível no *Covering-Slim* com o armazenamento de atributo(s) incluído(s) na condição adicional, juntamente com o vetor de características e o *rowId*, nas folhas do índice na *Slim-tree*. A execução de uma consulta por similaridade é idêntica a execução da abordagem *Table-Slim*, no entanto, a condição adicional é verificada diretamente no índice, reduzindo drasticamente o número de acessos a disco. A abordagem *Covering-Slim* é sempre mais rápida que a *Table-Slim* e muito mais rápida que a busca sequencial para situações de seletividade moderada a alta da condição adicional. Além disso, ela pode somente ser empregada para responder a consultas estendidas com condições se um índice

de cobertura apropriado estiver disponível, mas desempenha pior que a busca sequencial para situações de alta seletividade da condição adicional.

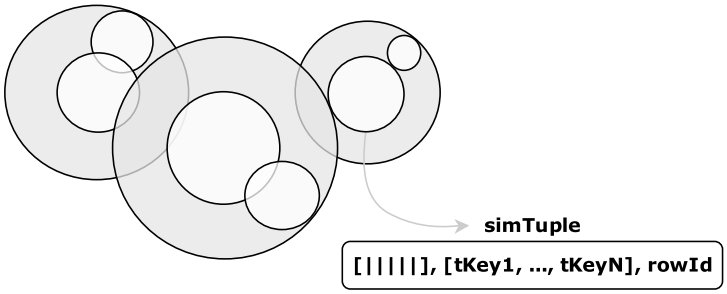


Figura 8 – Representação da abordagem *Covering-Slim*, demonstrando o fluxo entre a *Slim-tree* (representada por seus arcos/nós) e uma tupla indexada de onde os atributos tradicionais são diretamente validados.

3.5 Considerações Finais

Este capítulo apresentou os principais Métodos de Acesso (MA) para dados em domínios com relação de ordem total (dados tradicionais) e em domínios multidimensionais (dados complexos). Em relação aos dados tradicionais, o B^+*-tree* é o método que melhor se adequa aos objetivos deste trabalho em relação a realizar de maneira eficiente buscas tanto pontuais quanto por intervalo para verificar condições adicionais de uma consulta por similaridade. Os MAEs, por sua vez, apesar de também se enquadrarem aos dados complexos não são performáticos com dados de maior dimensionalidade e, portanto, esta pesquisa segue com os MAMs como sendo a abordagem mais apropriada para a indexação de dados complexos.

Também foi apresentado que os métodos de acesso para consultas por similaridade estendidas com condições são restritos a condições baseadas em palavras-chave ou são algoritmos específicos desenvolvidos sobre um MAM. Entretanto, esta pesquisa baseia-se na hipótese de que uma estrutura especializada é mais eficiente do que as abordagens existentes. O capítulo a seguir abordará o método proposto para a execução de consultas por similaridade com condições adicionais.

4 O MÉTODO PROPOSTO: CX-SIM TREE

A eficiência atingida pelos SGBDs no manuseio de dados tradicionais tem motivado estudos para o gerenciamento de dados complexos nesses sistemas. Embora a recuperação de dados complexos por conteúdo nos SGBDs atuais ainda não tenha atingido o suporte desejado, é possível adicionar condições complementares de busca às consultas. A associação de atributos tradicionais aos dados complexos, também considerados no predicado das consultas por similaridade, possibilita aprimorar a extração de informações relevantes de bases de dados compostas por dados complexos. Este trabalho parte do pressuposto que é possível aprimorar o desempenho de consultas combinando condições baseadas em similaridade e condições baseadas em dados associados utilizando-se uma estrutura composta que indexe simultaneamente dados complexos e dados de tipos tradicionais. Como citado na Seção 3.4, existem na literatura algumas propostas de estruturas projetadas para executar eficientemente consultas que combinam similaridade com condições adicionais, no entanto, as condições adicionais são limitadas a condições baseadas em palavras-chave.

Este capítulo apresenta um novo método de acesso métrico, chamado *cx-Sim tree* (*Condition-eXtended Similarity tree* – árvore para similaridade estendida com condições), que é a principal contribuição deste trabalho de mestrado (resultados parciais foram publicados em [48]). A *cx-Sim tree* é uma estrutura de indexação composta (i.e. indexa múltiplos atributos), dinâmica e projetada para armazenamento em disco, que tem por objetivo otimizar a execução de consultas por similaridade através da adição de condições gerais a dados complexos. A Seção 4.1 apresenta os fundamentos do método proposto. A Seção 4.2 descreve a estrutura da *cx-Sim tree* quando apenas um dado tradicional é indexado. Já a Seção 4.3 apresenta variações de organização da estrutura quando mais de um dado tradicional é indexado.

4.1 Fundamentos do Método *cx-Sim tree*

O método *cx-Sim tree* tem como fundamento a composição de duas ou mais estruturas de indexação, sendo uma delas um método de acesso para dados complexos e a(s) outra(s) um método de acesso ordenado (uma B^+-*tree*). A primeira grande vantagem de tal composição é proporcionar à base de dados indexada a noção de ordem total sobre seus elementos, possibilitada unicamente devido à presença do(s) atributo(s) tradicional(ais). Consequentemente, o conjunto de dados como um todo passa a ser comparável utilizando-se operadores relacionais ($<$, $\leq$, $\geq$, $>$), o que não era possível considerando apenas os dados complexos. Esta característica é fundamental para permitir agilizar a execução de consultas por similaridade com condições genéricas envolvendo atributos associados a um

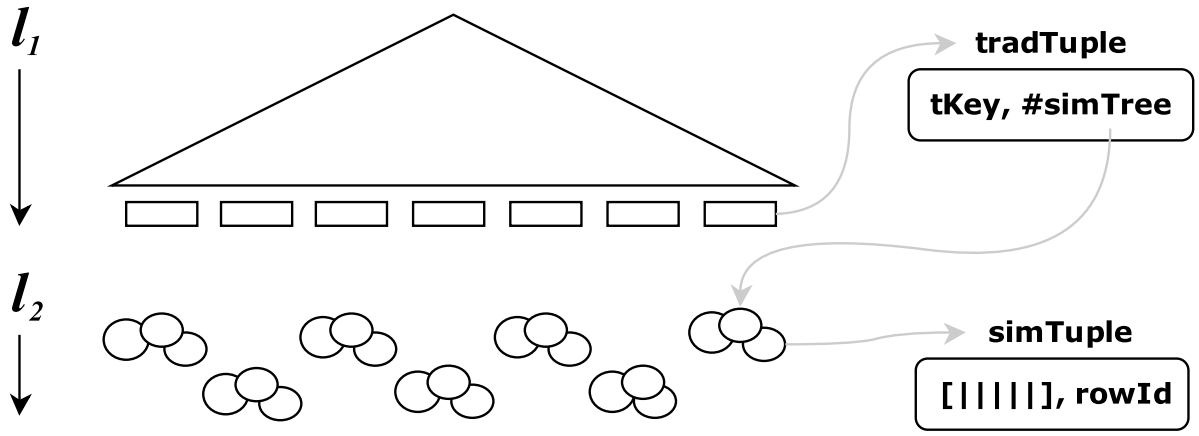


Figura 9 – Representação do método de acesso *cx-Sim tree*, destacando o fluxo da busca indexada através dos arcos cinzas.

dado complexo. Portanto, a *cx-Sim tree* é um método de acesso para a indexação de bases de dados complexas com atributos adicionais ordenáveis.

A segunda grande vantagem que o método proposto oferece na associação de dados tradicionais a dados complexos é o particionamento dos dados complexos. A *cx-Sim tree* mantém uma ou mais estruturas de dados ordenadas no seu nível mais alto, o que torna possível avaliar primeiramente a condição definida sobre o atributo tradicional, para que, posteriormente, a condição definida sobre o atributo complexo seja verificada apenas sobre os dados que tenham potencial real para compor o conjunto resposta. Ou seja, a *cx-Sim tree* faz um particionamento dos dados complexos que compartilham do(s) mesmo(s) atributo(s) tradicional(is). Como citado no Capítulo 3, ao realizar consultas sobre dados complexos existem duas operações onerosas, que são: os acessos a disco e os cálculos de distância. O particionamento dos dados complexos permite que os acessos a disco e os cálculos de distância desnecessários sejam evitados através de uma simples operação de comparação, muito menos onerosa, entre atributos de tipos de dados tradicionais.

A arquitetura da *cx-Sim tree* compreende dois níveis, como ilustrado na Figura 9, definidos conforme segue.

- **Nível** l_1 – Responsável pelo armazenamento dos atributos tradicionais, como também por possibilitar a pré-validação da condição adicional em relação a condição de similaridade. Considerando-se que o atributo tradicional a indexar deve ser, por natureza, ordenável, a estrutura de dados escolhida para armazenar seus valores e possibilitar uma busca eficiente é a B^+-*tree*. Portanto, as buscas em l_1 beneficiam-se de operações em tempo logarítmico. Pelas características apresentadas na seção 3.1, nota-se que não é viável substituir a B^+-*tree* por uma *hash table*, pois apesar desta estrutura proporcionar buscas em tempo constante *O(1)* no melhor caso, não possi-

bilitaria fazer buscas por intervalos de forma eficiente, que são frequentes quando se consideram condições gerais associadas à busca por similaridade (não apenas pontuais ou por palavras-chave), o que por sua vez é a motivação para o desenvolvimento do método proposto.

- **Nível** l_2 – Responsável pelo armazenamento dos atributos complexos, de uma maneira particionada, onde elementos que compartilham da mesma condição tradicional estão armazenados em uma mesma estrutura de indexação dinâmica para dados complexos. Considerando-se que naturalmente existirão diversos valores distintos para os atributos tradicionais, consequentemente os dados complexos estarão organizados neste nível em uma floresta de árvores dinâmicas. Árvores estas disjuntas em relação ao valor do atributo tradicional. Para que a *cx-Sim tree* seja dinâmica, isto é, não degrade com a ocorrência de inserções e remoções de elementos, é preciso que a estrutura de dados que indexa os dados complexos seja dinâmica. Neste sentido, o nível l_2 poderia ser formado por métodos de acesso multidimensionais, como a *R-tree*, ou métodos de acesso métricos, como a *M-tree* ou a *Slim-tree*. Como a *cx-Sim tree* tem por objetivo principal responder a consultas por similaridade, prefere-se utilizar MAMs, que são mais genéricos (indexam dados multidimensionais ou puramente métricos) e sofrem menos com o aumento da dimensionalidade dos dados, embora essa decisão dependa do domínio de dados complexos indexados.

Esta arquitetura genérica pode ter algumas variações, dependendo de quantos atributos tradicionais são armazenados na estrutura. Independentemente da variação, o método proposto tem uma característica que afeta o tamanho do índice: o primeiro nível foi projetado para armazenar apenas valores distintos (ou combinações distintas, caso mais de um atributo tradicional seja armazenado). Assim, o compartilhamento de atributos tradicionais por atributos complexos distintos não implica em duplicação de dados em l_1, gerando um índice menor. O índice é compacto visto que apenas um valor distinto de cada atributo tradicional é armazenado, independentemente da quantidade de elementos que têm este valor como atributo.

Outra característica importante é que o *cx-Sim tree* pode ser utilizado para agilizar consultas por similaridade estendidas com condições mesmo que não contenha todos os atributos usados na condição. Por exemplo, considere uma *cx-Sim tree* que indexe vetores de características de imagens de ultrassonografia e as idades dos pacientes e a consulta *Q5* a seguir:

Q5 – Retorne as k ultrassonografias mais similares a imagem de consulta i_q
dentre exames realizados em pacientes de 35 a 40 anos
e cujo número de gestações seja maior que 2

Observando-se a consulta *Q5*, que é uma consulta por similaridade que tem uma condição adicional composta por 2 termos, percebe-se que dois atributos necessários à sua execução estão presentes na estrutura (idade e vetor de características), mas o terceiro não (número de gestações). Mesmo assim, a *cx-Sim tree* é capaz de executar a consulta, pois, para cada elemento que é candidato a compor a resposta com relação ao critério de similaridade e que satisfaz o primeiro termo da condição, o algoritmo de busca faz um acesso à tabela de dados para verificar o segundo termo da condição. Esta característica é particularmente importante quando a consulta por similaridade é uma k-NN estendida com condições, pois esta operação não é comutativa com outras operações no plano de execução da consulta e precisa ser resolvida em um único algoritmo.

Esta abordagem é idêntica à utilizada nos algoritmos de execução de consultas por similaridade estendidas por condições da variação *Table-Slim*, descrita na Seção 3.4. Na *cx-Sim tree*, as consultas por similaridade com condições compostas são atendidas com a mesma abordagem: uma pós-validação das novas condições adicionais. Desta forma, utilizando-se uma *Slim-tree* em l_2, tal pós-processamento é viabilizado na *cx-Sim tree* através dos algoritmos da variação *Table-Slim*, de modo que a verificação dos atributos tradicionais subsequentes se dê imediatamente após a validação da condição de similaridade e mediante um novo acesso na tabela de dados. Uma vez recuperada a tupla da tabela de dados, pode-se verificar tantos atributos quantos forem desejados. Para possibilitar essa organização, ao invés de uma *Slim-tree* convencional a árvore empregada passa a ser a *Slim-tree* com a abordagem *Table-Slim*, como representado na Figura 10.

Independentemente do tipo de condição adicional a que o método de acesso esteja respondendo, a organização da *cx-Sim tree* baseia-se na existência da relação de ordem total no domínio dos dados tradicionais. Assim, as operações mais custosas durante a execução de uma consulta por similaridade – acessos a disco e cálculo de distância entre atributos complexos – ocorre somente após o filtro no atributo tradicional e de uma maneira assertiva, ou seja, somente sobre dados que realmente têm potencial para compor o conjunto resposta. É importante observar que este pós-processamento (proporcionado pela abordagem adotada no *Table-Slim*) é executado apenas para as consultas com condições adicionais compostas que incluam algum atributo não indexado na *cx-Sim tree*. Para as consultas com condições adicionais simples não faz-se necessário nenhum acesso a disco adicional.

Por fim, a *cx-Sim tree* é dinâmica, isto é, não degrada mediante as atualizações, pois se baseia em árvores dinâmicas em ambos os níveis da estrutura. Vale a pena destacar que o método não é balanceado quanto à altura, pois a altura de cada árvore no nível 2 depende da quantidade de elementos que compartilham o mesmo valor para o atributo tradicional indexado no nível 1. No entanto, este fato não degrada a performance das operações sobre a estrutura, pois se uma partição é elegível ao conjunto resposta com

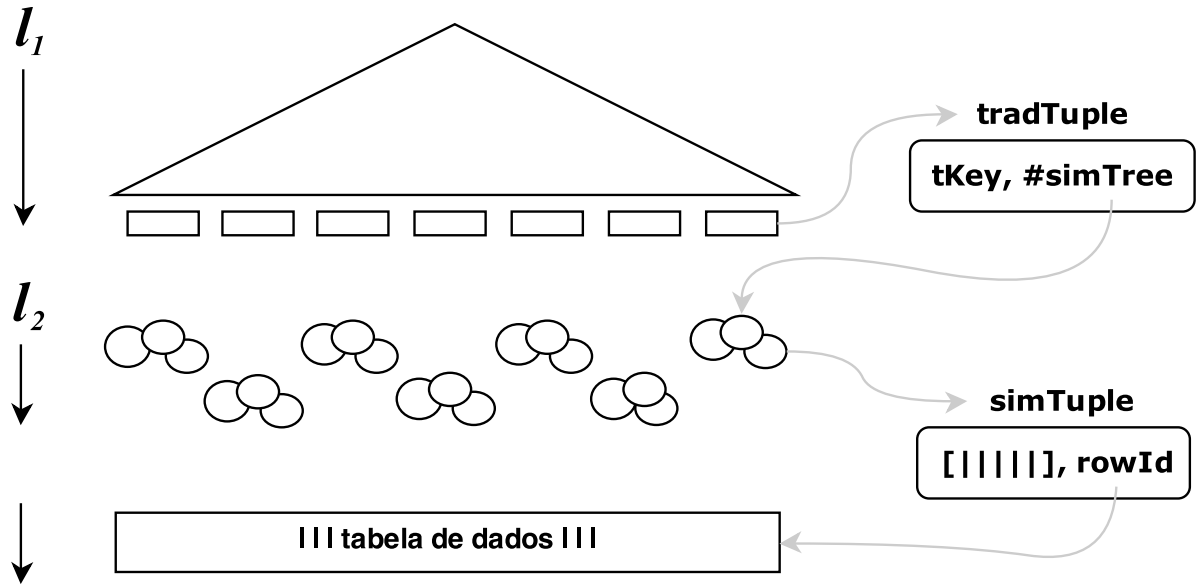


Figura 10 – Representação do método de acesso *cx-Sim tree*, destacando o fluxo da busca com a abordagem *Table-Slim* através dos arcos cinzas.

base no atributo tradicional, todos os elementos armazenados na *Slim-tree* desta partição têm que ser considerados, pois, em geral, não há correlação entre o atributo tradicional e o atributo complexo. Analisando os casos extremos, se cada elemento tem o mesmo valor para o atributo tradicional (ou uma mesma combinação de atributos), a B^+*-tree* terá apenas uma entrada e, consequentemente, existirá apenas uma árvore no nível 2. Considerando que esta árvore é dinâmica e balanceada pela altura, como é o caso da *Slim-tree*, a altura da *cx-Sim tree* é logarítmica no número de elementos indexados, sendo definida pela altura da árvore métrica mais um, que é a altura da B^+*-tree*. No outro extremo, se cada elemento tem um valor distinto (ou combinação distinta) para o atributo tradicional, a altura da *cx-Sim tree* é também logarítmica, sendo definida pela altura da B^+*-tree* mais um, que é a altura da árvore métrica.

As duas próximas seções descrevem detalhes da *cx-Sim tree* quando contém, respectivamente, um ou mais atributos tradicionais.

4.2 Estrutura da *cx-Sim tree* Contendo um Único Atributo Tradicional

Quando a *cx-Sim tree* armazena um atributo tradicional e um atributo complexo, ela é formada por uma B^+*-tree* e um índice para indexar o atributo complexo para cada valor distinto indexado na B^+*-tree*. A estrutura de dados escolhida para o armazenamento dos dados complexos na implementação da *cx-Sim tree* foi a *Slim-tree*, tanto por ser a

opção utilizada nas abordagens *Covering-Slim* e *Table-Slim*, quanto pelos avanços da estrutura em relação à *M-tree* apresentados no Capítulo 3. Desta forma, este capítulo utiliza a *Slim-tree* no detalhamento das variações desenvolvidas da estrutura proposta e seus algoritmos. A seguir estão descritas as operações de inserção, busca e remoção do *cx-Sim tree* quando contém apenas um dado tradicional.

4.2.1 Inserção

A inserção de elementos na *cx-Sim tree* quando contém um atributo tradicional é demonstrada no Algoritmo 1. Cada valor distinto em l_1 faz referência à região de l_2 onde o atributo complexo deve ser adicionado. Desta forma, a inserção primeiramente faz uma busca em l_1 (por igualdade na chave *traditionalValue*), para obter a referência da *Slim-tree* respectiva (linha 3). Se a chave não é encontrada, isto significa que o elemento atual é o primeiro a ser inserido nesta partição. Portanto, a chave *traditionalValue* é inserida na B^+*-tree*, tendo uma referência para uma nova *Slim-tree* que irá indexar os valores do atributo complexo para os elementos desta partição (linhas 4-8). O algoritmo acessa a *Slim-tree* apontada pela pela entrada da B^+*-tree* com chave *traditionalValue* e invoca a função de inserção da *Slim-tree*, fornecendo como parâmetros o dado complexo (*featureVector*) e o endereço físico da tupla (*rowId*) de origem dos atributos (linha 9). Nesta operação, o atributo complexo é inserido identificando, através de um processo recursivo, o nó folha cuja área de cobertura sofra o menor crescimento depois da inserção do novo elemento. Quanto menor o crescimento da área de cobertura do nó, menor a taxa de sobreposição e, consequentemente, melhor a performance da estrutura durante as operações de busca. Caso a inserção na *Slim-tree* seja realizada com sucesso, o algoritmo é finalizado com sucesso (linha 10). Caso contrário, a inserção é finalizada retornando insucesso, removendo a entrada criada na B^+*-tree* nesta chamada da inserção da *cx-Sim tree*, se for o caso (linhas 12-16).

4.2.2 Busca

O Algoritmo 2 representa o método de busca da *cx-Sim tree*, que corresponde à execução de consultas por similaridade (por abrangência ou aos k-vizinhos mais próximos) estendidas com condições. Em resumo, a abordagem do método proposto permite alcançar os subconjuntos dos elementos indexados (partições) que satisfazem a condição adicional de consulta efetuando apenas uma busca na B^+*-tree*. Depois de identificada(s) a(s) partição(s) que contêm os elementos qualificados de acordo com as condições sobre os atributos tradicionais, a busca sobre a(s) respectiva(s) árvore(s) que indexa o atributo complexo é iniciada. Por fim, caso necessário, os conjuntos de resultados de cada busca é mesclado (*merged*) em um único conjunto resposta.

Considerando que a condição da busca por similaridade pode ser composta, o

Algoritmo 1: Operação de Inserção na *cx-Sim tree*

```
   Data: featureVector, traditionalValue, rowId
   Result: True if the insertion is successful or false otherwise
 1 begin
 2     newLevel1 = false;
 3     level2Tree = level1.search(traditionalValue);
 4     if level2Tree is null then
 5         level1.insert(traditionalValue);
 6         level2Tree = new SlimTree(traditionalValue);
 7         newLevel1 = true;
 8     end
 9     if level2Tree.insert(featureVector, rowId) then
10         return true;
11     else
12         if newLevel1 then
13             level1.delete(traditionalValue);
14         end
15     end
16     return false;
17 end
```

algoritmo recebe como entrada um vetor de atributos de tipos tradicionais contendo os atributos usados na condição (traditionalVector). Além deste vetor, o algoritmo tem como parâmetros o vetor de características de referência (featureVector), o tipo da consulta (queryType: k-NN ou *Range query*), a função de distância δ, o limiar de similaridade (k para consulta k-NN ou ξ para *Range query*) e a condição.

Algoritmo 2: Operação de Busca na *cx-Sim tree*

```
   Data: featureVector, traditionalVector, queryType, δ, k|ξ, condition
   Result: The query result set
 1 begin
 2     level2Trees = level1.search(traditionalVector[0], condition);
 3     resultSet = null;
 4     foreach l2Tree in level2Trees do
 5         l2Tree.search(featureVector, traditionalVector[1..N], queryType, δ, k|ξ, condition,
           resultSet);
 6     end
 7     return resultSet;
 8 end
```

O algoritmo incialmente faz uma busca por valores que satisfaçam a (parte da) condição definida sobre o atributo tradicional indexado na *cx-Sim tree* em l_1, denotado por traditionalVector[0] no algoritmo (linha 2). Se os valores são encontrados, o algoritmo inicializa o conjunto de resultados e obtém as referências às *Slim-trees* de l_2 que contêm as partições onde os atributos complexos (que atendem à condição verificada) serão pesquisadas de acordo com o critério de similaridade especificado na consulta (linhas 3-5). Caso a busca por similaridade seja uma *Range query*, o algoritmo atravessa as devidas *Slim-trees* utilizando-se da estratégia *branch-and-bound*, realizando a poda de subárvores tais que suas áreas de cobertura não interceptem a área delimitada pelo raio da consulta

(ξ). Caso a busca por similaridade seja uma k-NN *query*, o raio inicial é setado como infinito, sendo gradualmente reduzido durante a execução da operação. A estrutura é varrida utilizando-se da estratégia *best-first*, que ordena as subárvores para serem visitadas de acordo com a distância mínima entre suas áreas de cobertura e o elemento de busca.

Considerando uma consulta por similaridade com condição adicional simples, os elementos que tivessem validadas as condições tradicional e de similaridade neste ponto da busca seriam adicionados ao conjunto resposta, fornecido como parâmetro para a busca na *Slim-tree*. Caso a condição adicional da consulta por similaridade seja composta, a busca prossegue com o pós-processamento para a validação do(s) outro(s) atributo(s) tradicional(is) não indexados (traditionalVector[1..N]), da seguinte maneira: como o segundo atributo não se encontra no índice, faz-se necessário um acesso a tabela de dados para recuperar a tupla referenciada pelo *rowId*, este sim armazenado no índice. Se o elemento corrente satisfaz a condição adicional para os atributos não indexados, este é adicionado ao conjunto resposta e a busca prossegue.

Uma consulta cuja condição definida sobre o atributo tradicional seja satisfeita por mais que uma entrada em l_1 requer a execução da operação de busca em mais de uma *Slim-tree* de l_2. Nestes casos, o resultado final da consulta é dado através do *merge* dos resultados de todas as buscas. Isso pode ser feito sequencialmente ou em paralelo. Considerando-se a execução sequencial, uma busca alimenta a próxima com o conjunto resposta parcial produzido. Considerando-se a execução em paralelo, as buscas devem compartilhar um conjunto resposta global. Esta abordagem em paralelo permite obter o conjunto resposta final (*merged*), bem como aprimorar a capacidade de poda ao executar *k-NN queries* reduzindo a área de cobertura mais rápido do que a abordagem sequencial.

4.2.3 Remoção

A remoção de elementos do método *cx-Sim tree* usa a operação de busca, como mostrado no Algoritmo 3. Encontrada a árvore em l_2 que contém a partição em que o elemento a ser removido pertence, o algoritmo invoca o método de remoção do elemento na respectiva *Slim-tree* (linhas 2-4). A operação de busca na *Slim-tree* é a *Point query*, pois deseja-se remover o elemento particular que resulta de uma busca pontual. No entanto, isso pode gerar reorganizações na *Slim-tree*, tais como distribuição de elementos entre nós irmãos, troca de representativos, remoção de nós e reinserção de elementos.

Se a remoção na *Slim-tree* deixar a árvore vazia (remoção do nó raiz), a árvore é destruída e o algoritmo propaga esta mudança, removendo o elemento em l_1 que referenciava esta árvore. Ou seja, o valor de um atributo tradicional que referencia uma árvore dinâmica de dados complexos é removido de l_1 quando o último dado complexo referenciado por ele é removido. A remoção do elemento em l_1 pode disparar operações de reorganização na B^+*-tree* para completar a operação de remoção. Tais operações em

l_1 (*unsplit*, distribuição de elementos entre nós irmãos, etc.) podem ser propagadas para a raiz da árvore. Portanto, o custo de remoção é proporcional à altura das árvores.

Algoritmo 3: Operação de Remoção na *cx-Sim tree*

```
   Data: featureVector, traditionalValue
   Result: True if the element was deleted and false otherwise
 1 begin
 2     level2Tree = level1.search(traditionalValue);
 3     if level2Tree is not null then
 4         if level2Tree.delete(featureVector) then
 5             if level2Tree.isEmpty() then
 6                 delete level2Tree;
 7                 level1.delete(traditionalValue);
 8             end
 9             return true;
10         end
11     end
12     return false;
13 end
```

4.3 Estrutura da *cx-Sim tree* Contendo Múltiplos Atributos Tradicionais

A abordagem apresentada na seção anterior é considerada como o caso geral do método proposto. Além disso, mostrou-se flexível ao ponto de responder tanto a consultas por similaridade com condições adicionais simples quanto com condições adicionais compostas. Especificamente para estas últimas, utilizando-se um índice de cobertura pode-se desonerar a validação dos atributos tradicionais efetuada após a verificação da similaridade. Um índice de cobertura é um índice que contém todos os atributos, e possivelmente mais, necessários à execução de uma consulta [9]. Portanto, não é necessário acessar a tabela de dados para responder à consulta e, consequentemente, o processo como um todo torna-se mais eficiente.

Esta seção apresenta três variações da *cx-Sim tree* como um "índice de cobertura". Ressalte-se as aspas no termo índice de cobertura, pois, como apresentado na seção anterior, a *cx-Sim tree* pode ser usada para agilizar consultas por similaridade estendidas com condições mesmo que nem todos os atributos estejam disponíveis no índice. A proposta das variações apresentadas nesta seção é incluir mais atributos tradicionais no índice, de forma a "cobrir mais" as consultas, mesmo que ainda sejam necessários acessos à tabela de dados. As variações têm como fundamento: (i) incluir um atributo tradicional nas entradas da árvore em l_2 (*cx-Sim Covering tree*); (ii) incluir mais uma B^+-*tree* em l_1, indexando um segundo atributo tradicional (*cx-Sim Chained tree*); ou (iii) fazer a B^+-*tree* em l_1 indexar uma combinação de atributos tradicionais, como um índice composto

(*cx-Sim Composite tree*). As seções que se seguem descrevem, respectivamente, estas três variações.

4.3.1 cx-Sim Covering tree

O algoritmo *Covering-Slim* apresentado na Seção 3.4 enquadra-se na condição de índice de cobertura descrita acima, pois os atributos adicionais (a partir do segundo inclusive) são armazenados juntamente ao vetor de características e o *rowId*, nas folhas do índice na *Slim-tree*, não necessitando pesquisas adicionais na tabela de dados. Através dele as consultas por similaridade com condições compostas podem ser atendidas de maneira idêntica a execução da variação *Table-Slim*: com uma pós-validação das novas condições adicionais e imediatamente após a validação da condição de similaridade. No entanto, este pós-processamento é executado diretamente no índice, reduzindo drasticamente o número de acessos a disco pois exime o acesso a tabela de dados.

O *cx-Sim Covering tree*, denominação empregada para esta abordagem e representada na Figura 11, assim como o *cx-Sim tree*, responde a consultas por similaridade com condições adicionais simples ou compostas e, não necessariamente, com apenas dois atributos tradicionais. Ou seja, esta abordagem está apta também a responder a consultas de dois ou mais atributos tradicionais em seu predicado, desde que um índice de cobertura apropriado esteja disponível.

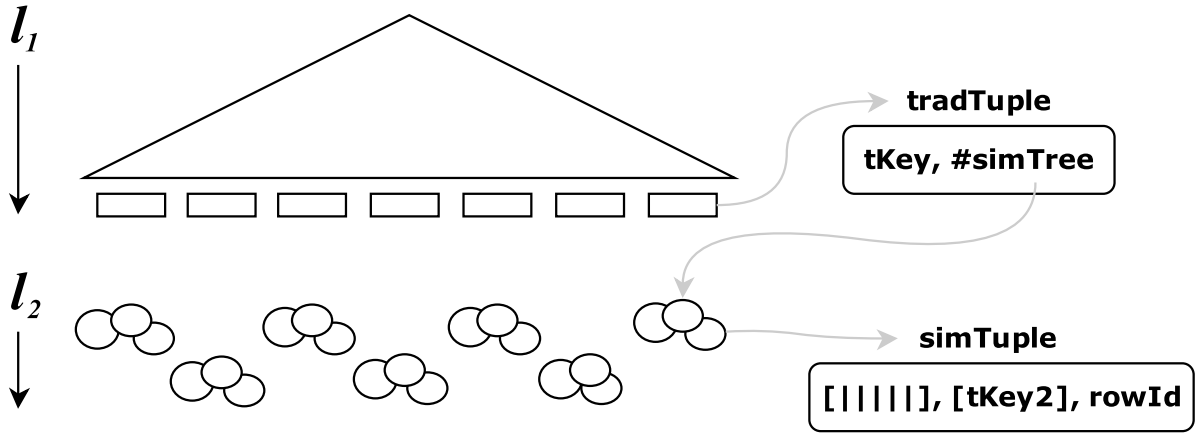


Figura 11 – Representação do método de acesso *cx-Sim Covering tree*, destacando o fluxo da busca indexada através dos arcos cinzas.

Arquiteturalmente, o *cx-Sim Covering tree* é tal que:

- Nível l_1: mantém as mesmas características e propriedades da abordagem *cx-Sim tree*.

- Nível l_2: mantém as mesmas características da abordagem *cx-Sim tree*, como exemplo a organização em uma floresta de árvores métricas dinâmicas, disjuntas em relação ao valor do atributo tradicional, além de armazenar todos os atributos tradicionais complementares. Para cada elemento a compor o índice, todo atributo tradicional a ser comparado nas buscas deve ser armazenado junto ao vetor de características e ao *rowId*, nas folhas do índice na *Slim-tree*. Para possibilitar essa organização, ao invés de uma *Slim-tree* convencional a árvore métrica empregada passa a ser a *Slim-tree* com a abordagem *Covering-Slim*. Consequentemente, durante a descrição do *cx-Sim Covering tree*, ao mencionar a árvore métrica será empregada a nomenclatura *Covering-Slim*.

Quando comparado ao *cx-Sim tree*, o *cx-Sim Covering tree* é equivalentemente comparável em relação a: execução de consultas por similaridade com condição adicional simples, basear-se na existência da relação de ordem total no domínio dos dados tradicionais, evitar acessos a disco e cálculos de distância desnecessários ao validar a condição adicional antes da similaridade, ser dinâmico e não balanceado quanto a altura em sua totalidade, mas sim balanceado em cada uma das suas árvores internas (B^+*-tree* ou floresta de árvores dinâmicas).

A principal vantagem do *cx-Sim Covering tree* trata-se de, no momento da pós-validação do atributo tradicional, não haver um acesso a tabela de dados para cada elemento que já teve a condição de similaridade validada.

Na seção 4.3.1.1 a seguir estão descritas as principais operações do *cx-Sim Covering tree* para a execução de consultas por similaridade estendida com condições.

4.3.1.1 Inserção

A inserção de elementos no *cx-Sim Covering tree* é demonstrada no Algoritmo 4. Num processo semelhante ao do *cx-Sim tree*, cada valor distinto ou range em l_1 faz referência a região de l_2 onde o atributo complexo deve ser adicionado. Então, o atributo complexo junto dos demais atributos tradicionais complementares são inseridos na respectiva Covering-Slim de l_2, identificando através de um processo recursivo o nó folha cuja área de cobertura sofra o menor crescimento depois da inserção do novo elemento. Quanto menor o crescimento da área de cobertura do nó, menor a taxa de sobreposição e, consequentemente, melhor a performance da estrutura durante as operações de busca.

Nesta abordagem o primeiro nível também armazena apenas valores distintos. Portanto, o compartilhamento de atributos tradicionais por atributos complexos distintos não implica em duplicação de dados em l_1, gerando um índice menor.

Algoritmo 4: Operação de Inserção no método *cx-Sim Covering tree*

```
   Data: featureVector, traditionalVector, rowId
   Result: True if the insertion is successful or false otherwise
 1 begin
 2     if !level1.search(traditionalVector[0]) then
 3         level1.insert(traditionalVector[0]);
 4         level2 = new CoveringSlim(traditionalVector[0]);
 5     end
 6     level2 = getCoveringSlim(traditionalVector[0]);
 7     if level2.insert(featureVector, traditionalVector[1..N], rowId) then
 8         return true;
 9     end
10     return false;
11 end
```

4.3.1.2 Busca

O Algoritmo 5 representa a execução de consultas com o uso do *cx-Sim Covering tree* que, assim como o *cx-Sim tree*, beneficia-se de operações em tempo logarítmico ao efetuar as buscar por valores ou ranges que satisfaçam a condição tradicional em l_1 ($<$, $\leq$, $\geq$, $>$, $=$, $\neq$). Encontrando elementos que atendam às condições de busca, são obtidas as referências para as árvores dinâmicas onde os critérios de similaridade serão avaliados.

Algoritmo 5: Operação de Busca no método *cx-Sim Covering tree*

```
   Data: featureVector, traditionalVector, condition, k
   Result: The query result set
1  begin
2      level2Trees = level1.search(traditionalVector[0], condition);
3      resultSet = null;
4      foreach l2Tree in level2Trees do
5          l2Tree.search(featureVector, traditionalVector[1..N], k, resultSet);
6      end
7      return resultSet;
8  end
```

Caso a busca por similaridade seja uma *range query*, o algoritmo atravessa as devidas *Covering-Slims* utilizando-se da estratégia *branch-and-bound*, realizando a poda de subárvores tais que suas áreas de cobertura não interceptem a área delimitada pelo raio da consulta. Caso a busca por similaridade seja uma *k-NN query*, o raio inicial é setado como infinito, sendo gradualmente reduzido durante a execução da operação. A estrutura é varrida utilizando-se da estratégia *best-first*, que ordena as subárvores para serem visitadas de acordo com a distância mínima entre suas áreas de cobertura e o elemento de busca.

Considerando uma consulta por similaridade com condição adicional simples, os elementos que tivessem validadas as condições tradicional e de similaridade neste ponto da busca seriam adicionados ao conjunto resposta. Caso a condição adicional da consulta por similaridade seja composta, a busca prossegue com o pós-processamento para a validação

do(s) outro(s) atributo(s) tradicional(is), neste caso indexados e consideravelmente menos onerosos que o *cx-Sim tree* por não necessitarem de acessos a tabela de dados. Se o elemento corrente também satisfaz a condição adicional para os atributos tradicionais pós-validados, este é adicionado ao conjunto resposta e a busca prossegue.

Uma consulta cuja condição definida sobre o atributo tradicional seja satisfeita por mais que uma entrada em l_1 requer a execução da operação de busca em mais de uma *Covering-Slim* de l_2. Nestes casos, o resultado final da consulta é dado através do *merge* dos resultados de todas as buscas. Isso pode ser feito sequencialmente ou em paralelo. Considerando-se a execução sequencial, uma busca alimenta a próxima com o conjunto resposta parcial produzido. Considerando-se a execução em paralelo, as buscas devem compartilhar um conjunto resposta global. Esta abordagem em paralelo permite obter o conjunto resposta final (*merged*), bem como aprimorar a capacidade de poda ao executar *k-NN queries* reduzindo a área de cobertura mais rápido do que a abordagem sequencial.

4.3.1.3 Remoção

A remoção de elementos do método *cx-Sim Covering tree* é esquematicamente idêntica a do método *cx-Sim tree*, visto que também usa a operação de busca como base. Como esta última operação abstrai as particularidades de identificação dos elementos em cada um dos dois métodos, uma vez identificada a existência da chave desejada basta providenciar a remoção da mesma nos níveis l_1 e l_2, como descrito a seguir.

Encontrada a chave a ser removida de l_1, o algoritmo dispara um processo para remover o elemento na respectiva *Covering-Slim* como mostrado no Algoritmo 6. A operação de busca na *Covering-Slim* é a *Point Query*, pois deseja-se remover o elemento particular que resulta de uma busca pontual. No entanto, isso pode gerar reorganizações na *Covering-Slim*, tais como distribuição de elementos entre nós irmãos, troca de representativos, remoção de nós e reinserção de elementos.

Se a remoção na *Slim-tree* deixar a árvore vazia (remoção do nó raiz), o algoritmo propaga esta mudança para o elemento em l_1 que referencia esta árvore. O valor de um atributo tradicional que referencia uma árvore dinâmica de dados complexos é removido de l_1 quando o último dado complexo referenciado por ele é removido. Esta operação dispara operações de reorganização na B^+*-tree* para completar a remoção desta chave. Tais operações em l_1 (*unsplit*, distribuição de elementos entre nós irmãos etc) podem ser propagadas para a raiz da árvore.

4.3.2 cx-Sim Chained tree

As abordagens apresentadas até então para que o *cx-Sim tree* responda a consultas por similaridade com condições estendidas compostas envolveram alterações apenas no

Algoritmo 6: Operação de Remoção no método *cx-Sim Covering tree*

```
   Data: featureVector, traditionalVector, condition
   Result: True if the element was deleted and false otherwise
 1 begin
 2     if cxSimTable.search(featureVector, traditionalVector, condition) then
 3         level2 = getCoveringSlim(traditionalVector[0]);
 4         if level2.delete(featureVector) then
 5             if level2.isEmpty() then
 6                 level1.delete(traditionalValue[0]);
 7             end
 8             return true;
 9         end
10     end
11     return false;
12 end
```

nível l_2. Consequentemente, a validação do segundo atributo tradicional se dá em um terceiro momento e imediatamente após a verificação da similaridade.

As próximas duas abordagens têm por objetivo, assim como o *cx-Sim tree* original, validar toda a condição adicional num primeiro momento (l_1) e só então partir para a validação mais onerosa – a verificação da similaridade (l_2) – apenas para os dados realmente elegíveis ao conjunto resposta.

Sendo assim, na variação *cx-Sim Chained tree* representada na Figura 12, o nível l_2 é mantido como o original – *Slim-tree*– e a alteração ocorre em l_1.

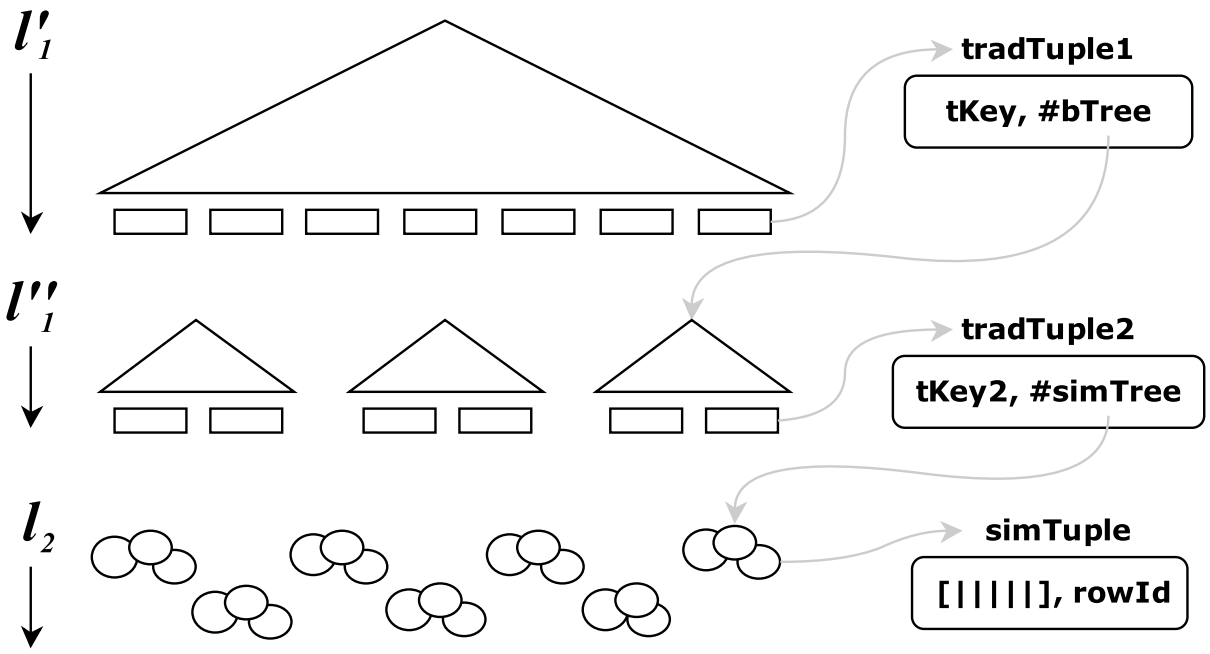


Figura 12 – Representação do método de acesso *cx-Sim Chained tree*, destacando o fluxo da busca indexada através dos arcos cinzas.

Arquiteturalmente, o *cx-Sim Chained tree* é tal que:

- Nível l_1: Este nível passa a ser composto por um encadeamento de B^+*-trees* (duas neste caso), de maneira que o primeiro atributo da condição adicional encontra-se em l'_1 e o segundo em l''_1. Tal encadeamento de pares de dados tradicionais se dá da mesma maneira que nas abordagens *cx-Sim tree* e *cx-Sim Covering tree* se encadeiam pares de dados formados por um dado tradicional e seu respectivos dado complexo. Ou seja, para cada valor distinto do primeiro atributo tradicional (l'_1) existirá uma referência a uma segunda B^+*-tree* em l''_1 onde será armazenado o segundo atributo tradicional. Por sua vez, para cada combinação de chaves (l'_1, l''_1) existirá uma árvore métrica em l_2 onde estará armazenado o vetor de características e, consequentemente, de onde será aferida a condição de similaridade.

- Nível l_2: responsável pelo armazenamento dos atributos complexos, mantém as mesmas características da abordagem *cx-Sim tree*, como exemplo a organização em uma floresta de árvores dinâmicas, disjuntas agora em relação a combinação de valores dos atributos tradicionais. Devido ao encadeamento explicado no nível anterior (l_1), não existe mais atributo tradicional armazenado neste nível. Consequentemente, a estrutura de dados escolhida para o armazenamento dos dados neste nível volta a ser a *Slim-tree* original.

Este método também responde a consultas por similaridade com condições adicionais simples ou compostas, desde que um índice de cobertura apropriado esteja disponível.

Quando comparado ao *cx-Sim tree*, o *cx-Sim Chained tree* é equivalentemente comparável em relação a: basear-se na existência da relação de ordem total no domínio dos dados tradicionais, evitar acessos a disco e cálculos de distância desnecessários ao validar a condição adicional antes da similaridade, ser dinâmico e não balanceado quanto a altura em sua totalidade, mas sim balanceado em relação a cada uma das suas árvores internas (B^+*-tree* ou florestas de árvores B^+ e de similaridade).

As principais vantagens do *cx-Sim Chained tree* tratam-se de:

- Proporcionar uma pré-validação de toda a condição tradicional antes de se avaliar desnecessariamente atributos complexos. Com isso, evita-se ainda mais que nas abordagens anteriores a execução de acessos a disco e cálculos de distância ao validar a condição de similaridade;

- Um índice menor em relação a espaço em disco, visto que não há repetição ao armazenar as chaves tradicionais (combinações de atributos tradicionais), enquanto que no *cx-Sim Covering tree* todos os atributos tradicionais são armazenados junto a cada vetor de características que representa o dado complexo.

A seguir estão descritas as principais operações do *cx-Sim Chained tree* para a execução de consultas por similaridade estendida com condições.

4.3.2.1 Inserção

A inserção de elementos no *cx-Sim Chained tree* é demonstrada no Algoritmo 7. Ao inserir a composição de atributos tradicionais em l_1, cada valor distinto ou range do primeiro atributo tradicional em l'_1 faz referência a região de l''_1 onde o segundo atributo tradicional deve ser adicionado. Por sua vez, cada valor distinto ou range do segundo atributo tradicional em l''_1 faz referência a região de l_2 onde o atributo complexo deve ser adicionado. Então, apenas o atributo complexo é inserido na respectiva *Slim-tree* de l_2, identificando através de um processo recursivo o nó folha cuja área de cobertura sofra o menor crescimento depois da inserção do novo elemento. Quanto menor o crescimento da área de cobertura do nó, menor a taxa de sobreposição e, consequentemente, melhor a performance da estrutura durante as operações de busca.

Algoritmo 7: Operação de Inserção no método *cx-Sim Chained tree*

```
   Data: featureVector, traditionalVector, rowId
   Result: True if the insertion is successful or false otherwise
 1 begin
 2     if !level1.search(traditionalVector) then
 3         level1.insert(traditionalVector);
 4         level2 = new SlimTree(traditionalVector);
 5     end
 6     level2 = getSlimTree(traditionalVector);
 7     if level2.insert(featureVector, rowId) then
 8         return true;
 9     end
10     return false;
11 end
```

Nesta abordagem o primeiro nível também armazena apenas combinações de valores distintos ($k_{l'_1}$, $k_{l''_1}$). Portanto, o compartilhamento de atributos tradicionais por atributos complexos distintos não implica em duplicação de dados em l_1, gerando um índice menor.

4.3.2.2 Busca

O Algoritmo 8 representa a execução de consultas através do *cx-Sim Chained tree* que, assim como o *cx-Sim tree*, beneficia-se de operações em tempo logarítmico ao efetuar as buscar por valores ou ranges que satisfaçam a condição tradicional em l_1 ($<$, $\leq$, $\geq$, $>$, $=$, $\neq$). Tal condição, neste caso, é tida como um encadeamento de condições tradicionais que devem atender a l'_1 e l''_1, respectivamente. Encontrando elementos que atendam às condições de busca, são obtidas as referências para as árvores dinâmicas onde os critérios de similaridade serão avaliados.

Algoritmo 8: Operação de Busca no método *cx-Sim Chained tree*

```
   Data: featureVector, traditionalVector, condition, k
   Result: The query result set
 1 begin
 2 |   level2Trees = level1.search(traditionalVector, condition);
 3 |   resultSet = null;
 4 |   foreach l2Tree in level2Trees do
 5 |   |   l2Tree.search(featureVector, k, resultSet);
 6 |   end
 7 |   return resultSet;
 8 end
```

Em relação a verificação estrita do critério de similaridade, o processo é análogo ao do *cx-Sim tree*. Caso a busca por similaridade seja uma *range query*, o algoritmo atravessa as devidas *Slim-trees* utilizando-se da estratégia *branch-and-bound*, realizando a poda de subárvores tais que suas áreas de cobertura não interceptem a área delimitada pelo raio da consulta. Caso a busca por similaridade seja uma *k-NN query*, o raio inicial é setado como infinito, sendo gradualmente reduzido durante a execução da operação. A estrutura é varrida utilizando-se da estratégia *best-first*, que ordena as subárvores para serem visitadas de acordo com a distância mínima entre suas áreas de cobertura e o elemento de busca.

Apesar do método em questão conter em l_1 ao menos dois níveis de atributos tradicionais, o mesmo está apto a responder a consultas cuja condição avalia apenas o primeiro atributo, pois, uma vez que este esteja validado, assume-se que toda a floresta de B^+*-trees* de l_1'' é elegível e segue para a validação da similaridade. Neste ponto da busca, todos os elementos que tivessem validadas as condições tradicional e de similaridade seriam adicionados ao conjunto resposta.

Caso a condição adicional da consulta por similaridade seja composta, todos os elementos que tivessem validadas as condições tradicional e de similaridade seriam adicionados ao conjunto resposta.

Uma consulta cuja condição definida sobre o atributo tradicional seja satisfeita por mais que uma entrada em l_1 requer a execução da operação de busca em mais de uma *Slim-tree* de l_2. Nestes casos, o resultado final da consulta é dado através do *merge* dos resultados de todas as buscas. Isso pode ser feito sequencialmente ou em paralelo. Considerando-se a execução sequencial, uma busca alimenta a próxima com o conjunto resposta parcial produzido. Considerando-se a execução em paralelo, as buscas devem compartilhar um conjunto resposta global. Esta abordagem em paralelo permite obter o conjunto resposta final (*merged*), bem como aprimorar a capacidade de poda ao executar *k-NN queries* reduzindo a área de cobertura mais rápido do que a abordagem sequencial.

4.3.2.3 Remoção

A remoção de elementos do método *cx-Sim Chained tree* também usa a operação de busca como base. Uma vez identificada a existência da chave desejada basta providenciar a remoção da mesma nos níveis l_1 e l_2, como descrito a seguir.

Encontrada a chave composta a ser removida de l_1, o algoritmo dispara um processo para remover o elemento na respectiva *Slim-tree* como mostrado no Algoritmo 9. A operação de busca na *Slim-tree* é a *Point Query*, pois deseja-se remover o elemento particular que resulta de uma busca pontual. No entanto, isso pode gerar reorganizações na *Slim-tree*, tais como distribuição de elementos entre nós irmãos, troca de representativos, remoção de nós e reinserção de elementos.

Algoritmo 9: Operação de Remoção no método *cx-Sim Chained tree*

```
   Data: featureVector, traditionalVector, condition
   Result: True if the element was deleted and false otherwise
 1 begin
 2     if cxSimTable.search(featureVector, traditionalVector, condition) then
 3         level2 = getSlimTree(traditionalVector);
 4         if level2.delete(featureVector) then
 5             if level2.isEmpty() then
 6                 level1.delete(traditionalValue);
 7             end
 8             return true;
 9         end
10     end
11     return false;
12 end
```

Se a remoção na *Slim-tree* deixar a árvore vazia (remoção do nó raiz), o algoritmo propaga esta mudança para o elemento em l_1'' que referencia esta árvore. O valor de um atributo tradicional que referencia uma árvore dinâmica de dados complexos é removido de l_1'' quando o último dado complexo referenciado por ele é removido. Esta operação dispara operações de reorganização na B^+*-tree* para completar a remoção desta chave. Tais operações em l_1'' (*unsplit*, distribuição de elementos entre nós irmãos etc) podem ser propagadas para a raiz da árvore. Se a remoção na B^+*-tree* de l_1'' deixar a árvore vazia (remoção do nó raiz), o algoritmo propaga esta mudança para o elemento em l_1' que referencia a respectiva árvore de l_1''. Ou seja, um processo análogo ao descrito entre l_2 e l_1'' ocorre entre l_1'' e l_1'.

4.3.3 cx-Sim Composite tree

Assim como o *cx-Sim Chained tree*, o *cx-Sim Composite tree* representado na Figura 13 também tem por objetivo validar a condição adicional por completo num primeiro momento (l_1) e, num segundo momento, validar a similaridade (l_2) apenas para dados com

potencial real de compor o conjunto resposta. Ou seja, o nível l_2 é mantido como o original – *Slim-tree*– e a alteração ocorre apenas em l_1.

A abordagem escolhida para este último algoritmo visa manter o mesmo nível de particionamento dos dados métricos (l_2) proporcionado pela *cx-Sim Chained tree*, mas eximindo a necessidade de uma floresta de B^+*-trees* para o armazenamento do segundo atributo tradicional (l''_1).

Arquiteturalmente, o *cx-Sim Composite tree* é tal que:

- Nível l_1: a chave armazenada neste nível da solução passa a ser uma composição de atributos tradicionais ($attrTrad_1$, ... , $attrTrad_N$) de modo que a estrutura, assim como na *cx-Sim tree*, seja composta por apenas uma B^+*-tree* em l_1. Da mesma maneira que no *cx-Sim Chained tree*, para cada combinação de chaves ($attrTrad_1$, $attrTrad_2$) – considerando-se como exemplo apenas dois atributos – existirá uma árvore dinâmica em l_2 onde estará armazenado o vetor de características e, consequentemente, de onde será aferida a condição de similaridade.
- Nível l_2: assim como na abordagem *cx-Sim Chained tree*, esta armazena os atributos complexos em uma floresta de árvores dinâmicas, disjuntas em relação a composição de valores dos atributos tradicionais. Devido a organização do nível l_1, também não existe mais atributo tradicional armazenado em l_2, pois a condição adicional é toda pré-validada antes de partir para a aferição da condição de similaridade. Consequentemente, a estrutura de dados escolhida para o armazenamento dos dados neste nível volta a ser a *Slim-tree* original.

Este método também responde a consultas por similaridade com condições adicionais simples ou compostas, desde que um índice de cobertura apropriado esteja disponível.

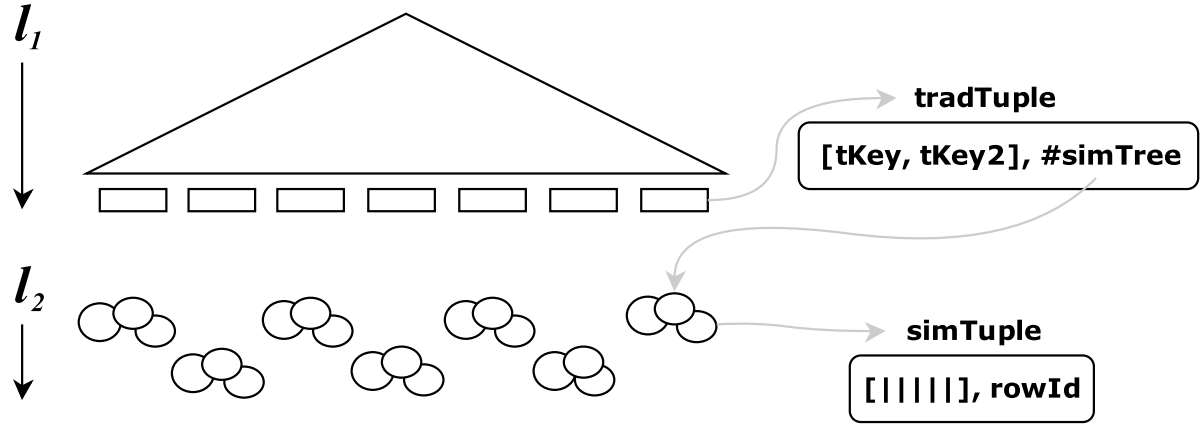


Figura 13 – Representação do método de acesso *cx-Sim Composite tree*, destacando o fluxo da busca indexada através dos arcos cinzas.

Mas, pensando nas vantagens que a sua abordagem traz em relação as anteriores, podemos destacar:

- A pré-validação de toda a condição tradicional antes de se avaliar desnecessariamente os atributos complexos. Com isso, assim como o *cx-Sim Chained tree*, evita-se por completo a execução de acessos a disco e cálculos de distância desnecessários ao validar a condição de similaridade;

- A redução no número de arquivos de índice necessários para o armazenamento dos dados tradicionais, pois diferentemente do *cx-Sim Chained tree*, este o faz com apenas uma B^+*-tree*. Além disso, assim como o *cx-Sim Chained tree*, esta abordagem não permite repetição no armazenamento das chaves (composições de atributos tradicionais) e, consequentemente, também ocupa um espaço reduzido em disco.

A seguir estão descritas as principais operações do *cx-Sim Composite tree* para a execução de consultas por similaridade estendida com condições.

4.3.3.1 Inserção

A inserção de elementos no *cx-Sim Composite tree* é demonstrada no Algoritmo 10. O armazenamento do dado tradicional, diferentemente do encadeamento de atributos descrito na Seção 4.3.2.1 para o método *cx-Sim Chained tree*, ocorre através de apenas uma chave, esta sim composta por N atributos. Por exemplo: ($attrTrad_1$, $attrTrad_2$) para dois atributos. Ao inserir a composição de atributos tradicionais em l_1, cada valor distinto ou range desta chave composta faz referência a região de l_2 onde o atributo complexo deve ser adicionado. Então, apenas o atributo complexo é inserido na respectiva *Slim-tree* de l_2, identificando através de um processo recursivo o nó folha cuja área de cobertura sofra o menor crescimento depois da inserção do novo elemento. Quanto menor o crescimento da área de cobertura do nó, menor a taxa de sobreposição e, consequentemente, melhor a performance da estrutura durante as operações de busca.

Nesta abordagem o primeiro nível também armazena apenas combinações de valores distintos ($attrTrad_1$, ... , $attrTrad_N$). Portanto, o compartilhamento de atributos tradicionais por atributos complexos distintos não implica em duplicação de dados em l_1, gerando um índice menor.

4.3.3.2 Busca

O Algoritmo 11 representa a execução de consultas com o uso do *cx-Sim Composite tree* que, assim como o *cx-Sim tree*, beneficia-se de operações em tempo logarítmico ao efetuar as buscar por valores ou ranges que satisfaçam a condição tradicional em l_1 (<,

Algoritmo 10: Operação de Inserção no método *cx-Sim Composite tree*

```
   Data: featureVector, compositeKey, rowId
   Result: True if the insertion is successful or false otherwise
 1 begin
 2     if !level1.search(compositeKey) then
 3         level1.insert(compositeKey);
 4         level2 = new SlimTree(compositeKey);
 5     end
 6     level2 = getSlimTree(compositeKey);
 7     if level2.insert(featureVector, rowId) then
 8         return true;
 9     end
10     return false;
11 end
```

$\leq$, $\geq$, $>$, $=$, $\neq$). Esta condição, é tida como uma composição de atributos tradicionais que compõem uma chave composta, respectivamente.

Algoritmo 11: Operação de Busca no método *cx-Sim Composite tree*

```
   Data: featureVector, compositeKey, condition, k
   Result: The query result set
 1 begin
 2     level2Trees = level1.search(compositeKey, condition);
 3     foreach tradIdx in (1 .. compositeKey.size - 1) do
 4         foreach tradTuple in level2Trees do
 5             if !operate(tradTuple.getAttr[tradIdx], compositeKey[tradIdx], condition)
               then
 6                 level2Trees.delete(tradTuple);
 7             end
 8         end
 9     end
10     resultSet = null;
11     foreach l2Tree in level2Trees do
12         l2Tree.search(featureVector, k, resultSet);
13     end
14     return resultSet;
15 end
```

É importante destacar que, ao trabalhar com chaves compostas que a relação de ordem é aplicável a chave como um todo. Por exemplo, considerando a chave composta (velocidade, altitude), em relação a ordem dos elementos, temos que:

$$[95, 0] > [80, 375] > [80, 100] > [65, 570]$$

Além disso, ao definir uma consulta cada atributo que compõe a chave compostas pode ser comparado através de um diferente operador, como apresentado abaixo:

Retorne as k posições de veiculos mais próximas ao ponto i_q
com velocidade entre 60 e 90km
e altitude maior que 300m

Sendo assim, para responder a consultas usando tal índice é necessário uma validação por atributo que compõe a chave. Ou seja, primeiramente valida-se apenas o primeiro atributo e, como resultado deste filtro, temos:

[80, 375], [80, 100] e [65, 570]

Este conjunto resposta provisório passa então pela validação do segundo atributo de onde a chave [80, 100] é eliminada, retornando:

[80, 375] e [65, 570]

Encontrados os elementos que atendam às condições adicionais da busca, são obtidas as referências para as árvores dinâmicas onde os critérios de similaridade serão avaliados.

Em relação a verificação estrita do critério de similaridade, o processo é análogo ao do *cx-Sim tree*. Caso a busca por similaridade seja uma *range query*, o algoritmo atravessa as devidas *Slim-trees* utilizando-se da estratégia *branch-and-bound*, realizando a poda de subárvores tais que suas áreas de cobertura não interceptem a área delimitada pelo raio da consulta. Caso a busca por similaridade seja uma *k-NN query*, o raio inicial é setado como infinito, sendo gradualmente reduzido durante a execução da operação. A estrutura é varrida utilizando-se da estratégia *best-first*, que ordena as subárvores para serem visitadas de acordo com a distância mínima entre suas áreas de cobertura e o elemento de busca.

Todos os elementos que tivessem validadas as condições tradicional e de similaridade seriam adicionados ao conjunto resposta.

Uma consulta cuja condição definida sobre o atributo tradicional seja satisfeita por mais que uma entrada em l_1 requer a execução da operação de busca em mais de uma *Slim-tree* de l_2. Nestes casos, o resultado final da consulta é dado através do *merge* dos resultados de todas as buscas. Isso pode ser feito sequencialmente ou em paralelo. Considerando-se a execução sequencial, uma busca alimenta a próxima com o conjunto resposta parcial produzido. Considerando-se a execução em paralelo, as buscas devem compartilhar um conjunto resposta global. Esta abordagem em paralelo permite obter o conjunto resposta final (*merged*), bem como aprimorar a capacidade de poda ao executar *k-NN queries* reduzindo a área de cobertura mais rápido do que a abordagem sequencial.

4.3.3.3 Remoção

Assim como nos outros métodos, o *cx-Sim Composite tree* também usa a operação de busca como base. Uma vez identificada a existência do dado desejado basta providenciar a remoção da mesma nos níveis l_1 e l_2, como descrito a seguir.

Encontrada a chave composta a ser removida de l_1, o algoritmo dispara um processo para remover o elemento na respectiva *Slim-tree* como mostrado no Algoritmo 12. A operação de busca na *Slim-tree* é a *Point Query*, pois deseja-se remover o elemento particular que resulta de uma busca pontual. No entanto, isso pode gerar reorganizações na *Slim-tree*, tais como distribuição de elementos entre nós irmãos, troca de representativos, remoção de nós e reinserção de elementos.

Algoritmo 12: Operação de Remoção no método *cx-Sim Composite tree*

```
   Data: featureVector, compositeKey, condition
   Result: True if the element was deleted and false otherwise
 1 begin
 2     if cxSimTable.search(featureVector, compositeKey, condition) then
 3         level2 = getSlimTree(compositeKey);
 4         if level2.delete(featureVector) then
 5             if level2.isEmpty() then
 6                 level1.delete(compositeKey);
 7             end
 8             return true;
 9         end
10     end
11     return false;
12 end
```

Se a remoção na *Slim-tree* deixar a árvore vazia (remoção do nó raiz), o algoritmo propaga esta mudança para o elemento em l_1 que referencia esta árvore. O valor de um atributo tradicional que referencia uma árvore dinâmica de dados complexos é removido de l_1 quando o último dado complexo referenciado por ele é removido. Esta operação dispara operações de reorganização na B^+*-tree* para completar a remoção desta chave. Tais operações em l_1 (*unsplit*, distribuição de elementos entre nós irmãos etc) podem ser propagadas para a raiz da árvore.

4.4 Considerações Finais

Este capítulo apresentou o método de acesso ***cx-Sim tree***, considerando quatro variações de implementação distintas, que visa acelerar a execução de consultas por similaridade com condições adicionais, mesmo para situações de alta seletividade da condição adicional de busca. A estrutura proposta foi concebida com o intuito de aprimorar a poda do espaço de busca, por considerar tanto a condição adicional quanto o critério de similaridade.

Cada uma das variações desenvolvidas traz as suas particularidades, mas independentemente disso, todas respondem a consultas por similaridade com condição adicional simples ou compostas. Em se tratando das particularidades de cada derivação, pode-se salientar o seguinte: a *cx-Sim tree* que indexa apenas um atributo tradicional é o caso geral do método proposto, pois a partir dele é possível indexar atributos tradicionais e

complexos e, ainda assim, pós-validar outros atributos tradicionais adicionais sem que um índice adequado tenha sido criado anteriormente. As variações da *cx-Sim tree* para indexar múltiplos atributos tradicionais são úteis quando a condição adicional da consulta por similaridade envolve mais de um atributo tradicional. A variação ***cx-Sim Covering tree*** é um índice de cobertura que encapsula os atributos tradicionais pós-validados na abordagem anterior. Consequentemente, os acessos a disco necessários para verificar o segundo atributo tradicional ao se usar uma *cx-Sim tree* com um único atributo tradicional são substituidos por acessos a índice na *cx-Sim Covering tree*, melhorando consideravelmente o tempo de execução. A variação ***cx-Sim Chained tree***, por sua vez, possibilita com que todas as combinações dos atributos tradicionais sejam armazenadas de maneira encadeada e sem repetição. Além dessa abordagem de indexação ocupar menos espaço em disco que as anteriores, ela possibilita a pré-validação da condição adicional. Com isso, são evitados acessos a disco e cálculos de distância desnecessários. Já a ***cx-Sim Composite tree*** proporciona os mesmos benefícios do método *cx-Sim Chained tree*, mas com um aprimoramento estrutural. Este armazena a composição de atributos tradicionais em uma chave composta, de modo que fisicamente as chaves distintas fiquem em apenas um arquivo de índice.

Este trabalho é inovador em relação aos algoritmos *Covering-Slim* e *Table-Slim*, pois ele apresenta uma nova estrutura de indexação com três derivações, onde todas combinam um atributo complexo e um ou mais atributos tradicionais. Além disso, apenas a variação *cx-Sim Covering tree* requer o encapsulamento parcial dos atributos complexos para cada entrada no índice (em l_2). As outras três derivações do método não requerem encapsulamento algum do(s) atributo(s) tradicional(is) para cada entrada no índice, o que proporciona uma redução no tamanho da estrutura de dados como um todo. Estas características fazem do método proposto uma alternativa adequada para consultas por similaridade estendidas com condições em grandes bases de dados complexos.

O capítulo 5 apresenta os experimentos executados fazendo-se uso das derivações do método *cx-Sim tree*.

5 EXPERIMENTOS E RESULTADOS

O objetivo deste capítulo de experimentos é apresentar o avanço em eficiência na execução de ${}_ck$-NN sobre dados complexos, comparando o método proposto com outros métodos que respondem a consultas por similaridade com condições gerais (não apenas baseadas em palavras-chave). A Seção 5.1 descreve os parâmetros dos experimentos e as bases de dados utilizadas: uma contendo dados georreferenciados do censo dos Estados Unidos da América, outra contendo imagens médicas e a terceira contendo dados georreferenciados de veículos.

A primeira parte dos experimentos avalia o desempenho da *cx-Sim tree* em comparação com outras abordagens na execução de consultas por similaridade com **condições adicionais simples**, isto é, que utilizam apenas um atributo tradicional. Partindo-se do pressuposto que o método *cx-Sim tree* foi desenvolvido pra responder a consultas ${}_ck$-NN com condições gerais (não baseadas em palavras-chave), somente os métodos *Covering-Slim* e *Table-Slim* (descritos na Seção 3.4) compõem uma *baseline* realmente comparável para estes experimentos. Isso porque, enquanto esses métodos aceitam todo o intervalo de valores dos tipos de dados tradicionais, as soluções existentes baseadas em palavras-chave limitam o conjunto de valores possíveis e são baseadas em árvores de baixa dimensionalidade. Desta forma, essa parte dos experimentos compara a *cx-Sim tree* com as abordagens *Table-Slim* e *Covering-Slim*. Os resultados apresentados na Seção 5.2 foram obtidos inicialmente através da aplicação dos métodos *cx-Sim tree*, *Covering-Slim* e *Table-Slim* nas três bases de dados citadas e publicados em Soares e Kaster[48]. Para possibilitar uma comparação justa, o método *cx-Sim tree* foi implementado utilizando *Slim-trees* no segundo nível da estrutura de dados. Esta escolha também é fortalecida pelo fato da *Slim-tree* ser uma estrutura de indexação balanceada e eficiente para dados métricos e de alta dimensionalidade.

Após finalizados os experimentos comparando os três métodos citados, surgiu um questionamento importante:

O particionamento proporcionado pela arquitetura em dois níveis do cx-Sim tree seria mais performático que um algoritmo que usasse uma Slim-tree e uma B^+-tree como estruturas independentes, mas que verificasse a condição adicional diretamente na B^+-tree?

O algoritmo desenvolvido, denominado *B-Slim*, requer que exista uma *Slim-tree* indexando o atributo complexo utilizado na consulta por similaridade estendida com condições (simples) e uma B^+-*tree* indexando o atributo utilizado na condição, mas estas são

estruturas de dados independentes. Isto é, a inserção de um novo dado na tabela de dados implica em inserir o atributo tradicional na *B*$^+$*-tree*, bem como o respectivo vetor de características na *Slim-tree*, podendo esses índices serem usados isoladamente. O algoritmo efetua buscas em dois passos: uma busca na *B*$^+$*-tree*, retornando o conjunto de *rowIds* das tuplas que satisfazem a condição adicional definida sobre o atributo tradicional, e outra na *Slim-tree*, onde a consulta por similaridade estendida com condições (Rq ou $_c k$-NNq) é executada. A consulta por similaridade na *Slim-tree* primeiramente verifica o critério de similaridade, realizando podas com base na propriedade de desigualdade triangular. Um elemento candidato a fazer parte da resposta da consulta com base no critério de similaridade é testado quanto à condição adicional verificando-se se o seu *rowId* pertence ao conjunto de *rowIds* obtido no primeiro passo. Este algoritmo não realiza acessos adicionais à tabela de dados, como é o caso da abordagem *Table-Slim*, e não exige que exista um índice composto que contenha os atributos usados na consulta, como é o caso da *Covering-Slim* e da *cx-Sim tree*. Consequentemente, enriquece o caráter comparativo apresentado na Seção 5.2.

Observe-se que o algoritmo *B-Slim* é diferente de um plano de execução formado por duas operações subsequentes: primeiro fazer uma seleção convencional, selecionando-se as tuplas que satisfazem a condição adicional, e, em seguida, executar a consulta por similaridade com base no resultado da operação anterior. Este plano de execução não executaria a consulta por similaridade utilizando a *Slim-tree*, e sim por meio de uma varredura sequencial. O *B-Slim* também não é equivalente a um plano de execução que realize duas buscas independentes, uma na *B*$^+$*-tree* e outra na *Slim-tree*, seguida pela interseção dos resultados, pois se a consulta for uma $_c k$-NNq o resultado não seria correto. Por exemplo, considere-se a consulta *Q1* (*Retorne as k imagens de ultrassonografia mais similares à imagem de consulta* i_q *dentre exames realizados em pacientes de 35 a 40 anos*). A interseção entre os pacientes de 35 a 40 anos e as imagens mais similares à imagem de consulta i_q pode ter menos do que k elementos (pode até ser vazia), o que não responde à consulta indicada.

A segunda parte dos experimentos avalia o desempenho da *cx-Sim tree* na execução de consultas por similaridade estendidas com **condições adicionais compostas**, comparando as diferentes variações desenvolvidas para indexar mais do que um dado tradicional. Os resultados apresentados na Seção 5.3 foram obtidos por meio de testes envolvendo a *cx-Sim tree* indexando apenas um atributo convencional, identificada como *cx-Sim Simple tree*, e as variações *cx-Sim Covering tree*, *cx-Sim Composite tree* e *cx-Sim Chained tree*, nas mesmas bases de dados reais utilizadas nos experimentos anteriores. As principais diferenças entre os métodos são analisadas neste capítulo considerando as métricas das operações mais custosas e relevantes de uma consulta por similaridade: acessos a disco, cálculos de distância, número de comparações e tempo total de execução.

5.1 Descrição das Bases de Dados e dos Experimentos

Três bases de dados reais foram utilizadas nos experimentos: *USCities*, *HCImages* e *BRPositions*. A base de dados *USCities* tem como atributo complexo as coordenadas geográficas das cidades dos Estados Unidos da América e mais aproximadamente 100 atributos numéricos representando dados demográficos do censo do ano 2000[1]. A base de dados *HCImages* é composta por histogramas de cores (256 níveis de cinza) e 13 atributos de tipos de dados tradicionais (numéricos e textuais), extraídos dos cabeçalhos de imagens médicas de exames radiológicos no formato DICOM, realizados pelo Hospital das Clínicas da Faculdade de Medicina de Ribeirão Preto da Universidade de São Paulo (HCFMRP-USP). A última base de dados, *BRPositions*, refere-se a um histórico de posições de veículos que fazem uso das soluções tecnológicas da empresa Veltec Soluções Tecnológicas S/A[2]. Por razões de confidencialidade, estas bases não permitem identificar particularidades dos habitantes dos Estados Unidos da América, dos pacientes ou dos veículos e/ou seus proprietários.

A facilidade que os dados georreferenciados proporcionam à definição de consultas e interpretação de resultados foi decisiva para a escolha do primeiro e terceiro conjuntos de dados, pois a similaridade nestes casos é mais intuitiva. Isso porque em aplicações geográficas a similaridade é comumente medida pela distância entre elementos no espaço 2D. Dessa maneira, a função adotada para os conjuntos de dados *USCities* e *BRPositions* foi a distância Euclidiana (L_2). Em relação à base de dados *HCImages*, a função adotada foi a distância Manhattan (L_1). É importante destacar que o método proposto é naturalmente aplicável a outros dados complexos (como vídeos, dados científicos ou séries temporais), bastando pra isso a definição do vetor de características que represente os dados complexos e da função de distância apropriada.

O método *cx-Sim tree* e suas variações e os concorrentes foram desenvolvidos em C++ utilizando a biblioteca Arboretum. Esta, por sua vez, trata-se de uma biblioteca *open-source* de métodos de indexação para dados complexos, desenvolvida em C++ pelo Grupo de Banco de Dados e Imagens do Instituto de Matemática e Ciência da Computação da Universidade de São Paulo (GBDI-ICMC-USP)[3]. Os experimentos foram executados em um computador equipado com um processador intel CORE i5, 4GB de RAM, 500GB de HD SATA2 e sistema operacional Ubuntu Linux (versão 12.04 / 64 bits).

Os experimentos da Seção 5.2 testaram as quatro estruturas de dados através de consultas por similaridade estendidas com condições simples, a saber:

i) Retorne as k cidades mais próximas a cidade X

[1] Em 2000, o U.S. Census Bureau realizou um censo nos Estados Unidos da América, cujos dados estão disponíveis em <http://www.census.gov/main/www/cen2000.html>

[2] Disponível em <http://www.veltec.com.br>

[3] Disponível em <http://gbdi.icmc.usp.br/arboretum>

e idade_media_da_populacao entre Y e Z anos
(USCities)

ii) Retorne os k exames mais similares ao exame E
e idade_paciente entre F e G anos
(HCImages)

iii) Retorne as k posições mais próximas ao ponto de referência P
e velocidade $\leq$ *Q km/h*
(BRPositions)

Os experimentos da Seção 5.3 testaram as quatro estruturas de dados através das consultas por similaridade estendidas com condições compostas a seguir:

iv) Retorne as k cidades mais próximas a cidade X
com idade_media_da_populacao entre Y e Z anos
e tamanho_medio_da_familia $>$ *W*
(USCities)

v) Retorne os k exames mais similares ao exame E
com idade_paciente entre F e G anos
e peso_paciente entre C e D
(HCImages)

vi) Retorne as k posições mais próximas ao ponto de referência P
com velocidade $\leq$ *Q km/h*
e altitude entre R e S
(BRPositions)

As consultas sofreram duas variações controladas e idênticas para todos os métodos avaliados:

- **Número de vizinhos mais próximos (*k*):** variação entre 1 e 500;
- **Seletividade da condição envolvendo os atributos tradicionais:** variação entre 50% e $99,99\%$ para consultas com condições adicionais simples e variação entre 75% e $99,99\%$ para consultas com condições compostas.

Como mostra a Tabela 1, os conjuntos de dados *USCities*, *HCImages* e *BRPositions* têm, respectivamente, 25.374, 501.100 e 2.715.873 elementos, o que permite avaliar o

comportamento do método proposto com volumes distintos de dados. A tabela também mostra que, para cada *setup* da consulta, ou seja, para cada conjunto de valores distintos destas variações, as consultas foram executadas com elementos aleatoriamente selecionados (as cidades X, os exames E e os pontos de referência P), e os resultados apresentados a seguir referem-se à média de todas as execuções para cada *setup* (média de 500 consultas para os conjuntos de dados *USCities* e *BRPositions* e de 100 consultas para o conjunto *HCImages*).

Tabela 1 – Tamanho das bases de dados e números de execuções realizadas em cada uma.

Dataset	**USCities**	**HCImages**	**BRPositions**
#Elements	25.374	501.100	2.715.873
#Query Objects	500	100	500

5.2 Abordagem para Consultas por Similaridade com Condições Adicionais Simples

Nesta seção são apresentados os experimentos que avaliam a *cx-Sim tree* com relação aos seus concorrentes na resolução de consultas aos vizinhos mais próximos estendidas com condições adicionais simples, variando-se o número de vizinhos (Subseção 5.2.1) e variando-se a seletividade da condição adicional (Subseção 5.2.2).

5.2.1 Experimentos com Variação no Número de Vizinhos Mais Próximos

Esta avaliação experimental é iniciada com a variação no número de vizinhos mais próximos (k), fixando-se a seletividade das condições de busca. As seletividades consideradas para os três conjuntos de dados são apresentadas na Tabela 2.

Tabela 2 – Seletividades das consultas com condições simples com variação no número de vizinhos mais próximos.

Dataset	**USCities**	**HCImages**	**BRPositions**
Seletividade	$90,7\%$	$88,6\%$	$87,7\%$

Considerando-se a Figura 14 onde são apresentados os números de acessos ao arquivo de dados, tal análise mostra que os métodos *B-Slim*, *Covering-Slim* e *cx-Sim* apresentam quantidades significantemente menores que o quarto método – *Table-Slim*. Os três primeiros necessitaram somente k acessos a disco por execução enquanto o quarto atingiu ordens de 18, 35 e 37 vezes mais acessos a disco para os conjuntos de dados *USCities*, *HCImages* e *BRPositions*, respectivamente. Tal diferença é aceitável analisando-se o esforço necessário para decidir se uma tupla será ou não adicionada ao conjunto resposta. Enquanto o *Table-Slim* necessita de um acesso a disco para obter o dado complexo e outro

para acessar o dado tradicional, os demais métodos necessitam de apenas um acesso pra analisar os dois.

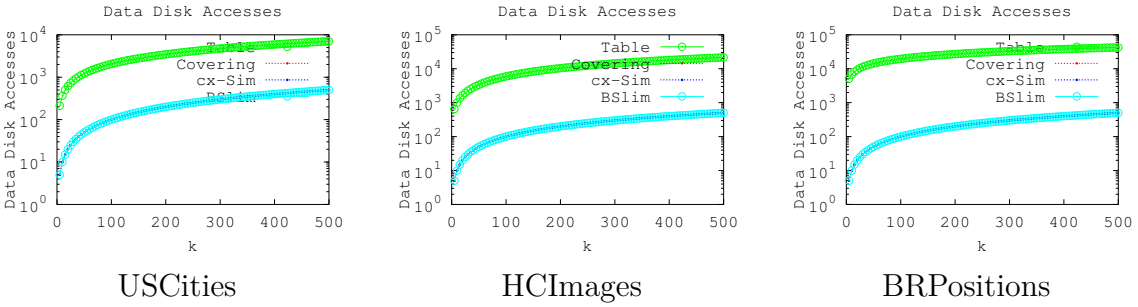


Figura 14 – Número de acessos ao arquivo de dados, variando o número de vizinhos (k).

Considerando-se acessos a índice, o método *cx-Sim* obteve o resultado solicitado com muito menos esforço que os demais. De acordo com a Figura 15, a quantidade de acessos era menor na ordem de 5, 8 e 6 vezes em relação aos métodos *Covering-Slim* e *Table-Slim*. Quando comparado ao seu plano de execução concorrente, o *B-Slim*, a diferença foi ainda maior, da ordem de 11, 10 e 13 vezes para os conjuntos de dados *USCities*, *HCImages* e *BRPositions*, respectivamente. Devido a esta vasta diferença relacionada a acessos a disco, o método *cx-Sim* também mostrou-se mais eficiente em relação à quantidade total de acessos a disco, como mostrado na Figura 16.

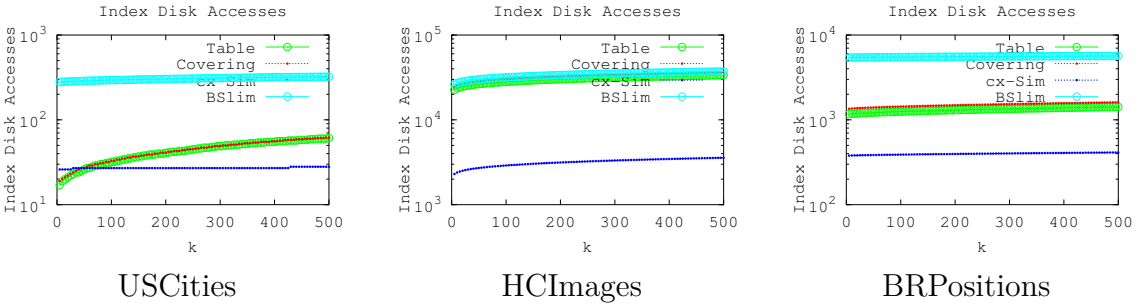


Figura 15 – Número de acessos a blocos de índice, variando o número de vizinhos (k).

Um quesito onde o êxito obtido pelo *cx-Sim* foi consideravelmente melhor e merece ser destacado pode ser notado na Figura 17, onde são apresentados os números de comparações entre os dados tradicionais. Devido a arquitetura em dois níveis, uma vez fixada a condição (dado tradicional), este MAM apresenta quantidade constante de comparações independentemente do k selecionado, pois o esforço para a busca na B^+-Tree é o mesmo. Tal resultado não é possível aos outros MAMs e estes obtiveram resultados da ordem de 390, 1876 e 120 (*Covering-Slim* e *Table-Slim*) e 121, 5700 e 2256 (*B-Slim*) vezes mais para as bases *USCities*, *HCImages* e *BRPositions*, respectivamente.

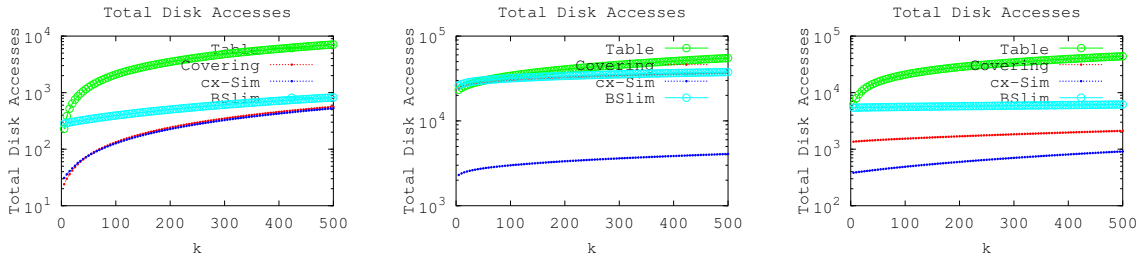


Figura 16 – Quantidade total de acessos a disco, variando o número de vizinhos (k).

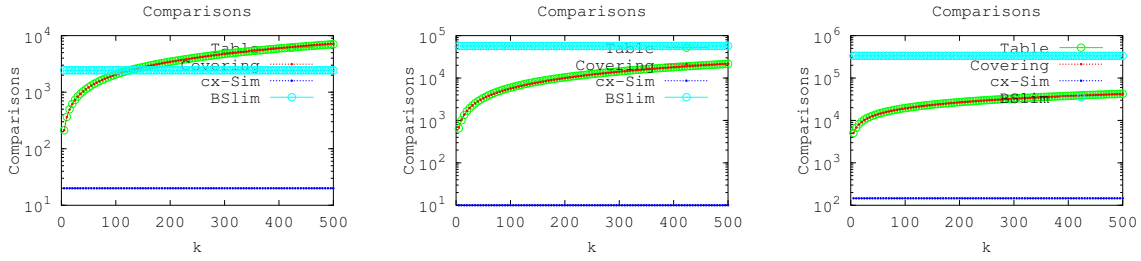


Figura 17 – Quantidade de comparações, variando o número de vizinhos (k).

Além do número de comparações, foi analisado também a quantidade de cálculos de distância entre pares de dados complexos. De acordo com informações apresentadas na Figura 18, os resultados obtidos com o método *cx-Sim* são menores que os obtidos pelos outros três, *B-Slim*, *Covering-Slim* e *Table-Slim*. Tal diferença era da ordem de 7,5, 8,5 e 6,6 vezes mais comparações para as bases *USCities*, *HCImages* e *BRPositions*, respectivamente. Vale a pena observar que o número obtido pelos demais métodos em comparação são idênticos, pois todos armazenam a mesma massa de dados complexos em apenas uma *Slim-tree*. O *cx-Sim*, por sua vez, o faz de maneira particionada na floresta de árvores métricas organizada no segundo nível do MAM.

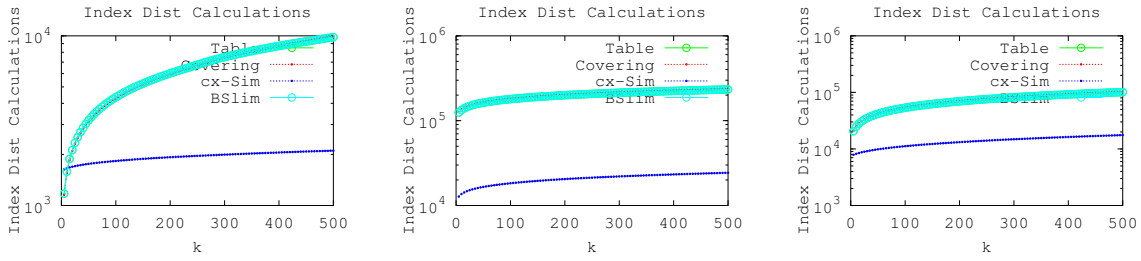
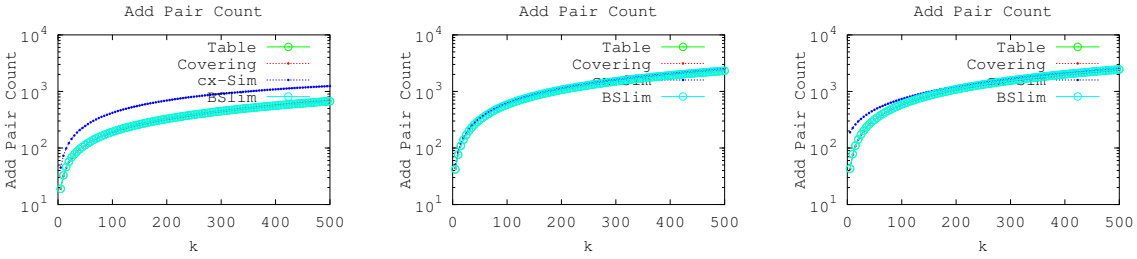


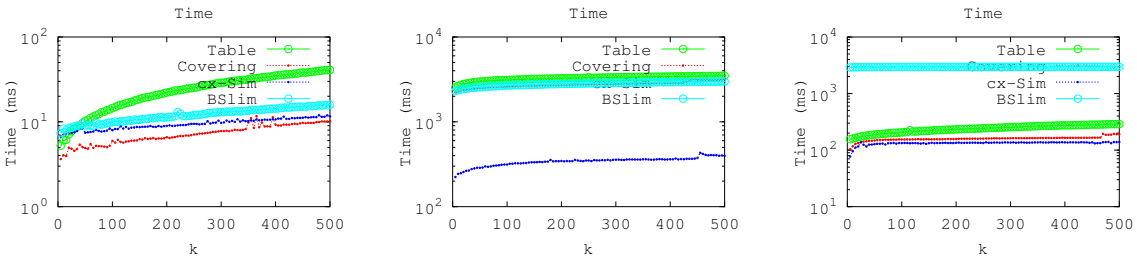
Figura 18 – Número de cálculos de distância no índice, variando o número de vizinhos (k).

Há um quesito onde o método *cx-Sim* apresentou valores maiores que o *B-Slim*, o *Covering-Slim* e o *Table-Slim*, particularmente nos experimentos alimentados com os conjuntos de dados georreferenciados. Este é a quantidade de dados adicionados ao conjunto resposta durante a busca e pode ser observado na Figura 19. Esta diferença foi a menor obtida dentre os quesitos avaliados – da ordem de até 27%, 3% e 11% maior que as bases *USCities*, *HCImages* e *BRPositions*, respectivamente. Tal resultado não foi significante relacionado a performance das buscas por similaridade estendida com condições, pois, de acordo com a Figura 20, o tempo médio de processamento das consultas executadas com o *cx-Sim* foi consideravelmente menor. Em relação à base de dados *USCities*, estes foram da ordem de até 27%, 36% e 84% menores que os métodos *B-Slim*, *Covering-Slim* e *Table-Slim*, respectivamente. Considerando-se a base de dados *HCImages*, as diferenças foram da ordem de até 87%, 84% e 88% menores que os métodos *B-Slim*, *Covering-Slim* e *Table-Slim*, respectivamente. Na base de dados *BRPositions*, por sua vez, as diferenças apresentadas foram da ordem de até 95%, 79% e 84% menores que os métodos *B-Slim*, *Covering-Slim* e *Table-Slim*, respectivamente.



USCities HCImages BRPositions

Figura 19 – Número de operações de fila no conjunto resposta (número de inserções na estrutura de dados auxiliar), variando o número de vizinhos (k).



USCities HCImages BRPositions

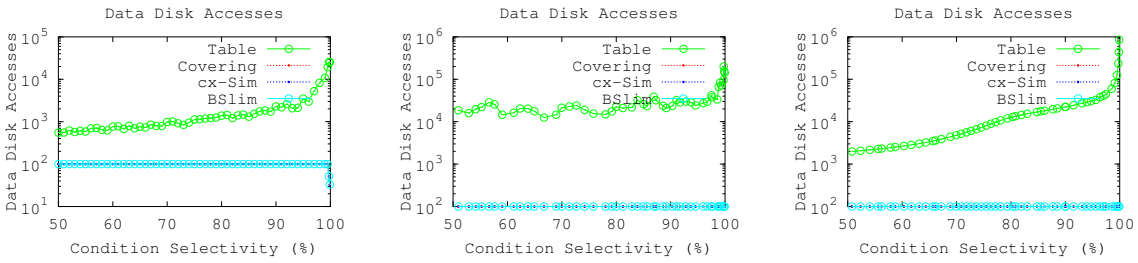
Figura 20 – Tempo de processamento das consultas, variando o número de vizinhos (k).

Encerradas as comparações variando a quantidade de vizinhos mais próximos, segue na próxima seção os resultados dos experimentos cuja variação ocorreu na seletividade da condição de busca.

5.2.2 Experimentos com Variação na Seletividade da Condição de Busca

A segunda parte dos experimentos considera a variação na seletividade da condição de busca. Nesta fase, o k – variável da fase anterior – foi fixado em 100, ou seja, deseja-se recuperar os 100 vizinhos mais próximos ao objeto de consulta.

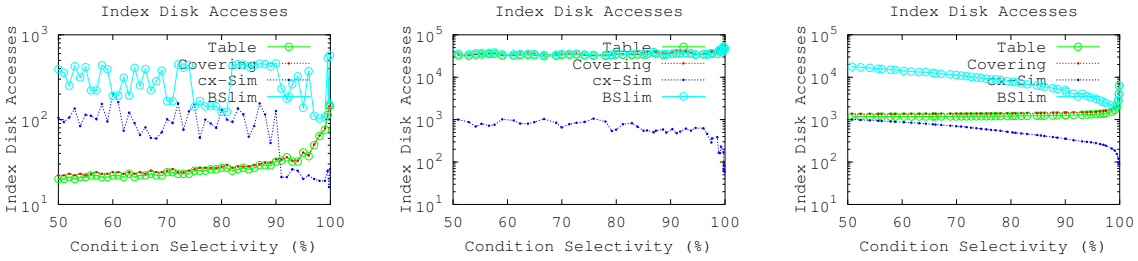
Iniciando-se pelo número de acessos ao arquivo de dados (Figura 21), este permaneceu inalterado para os métodos *B-Slim*, *cx-Sim* e *Covering-Slim*, ou seja, k acessos. Pelas mesmas razões mencionadas anteriormente, tal valor permaneceu em ascenção para o método *Table-Slim*, principalmente para os conjuntos de dados *USCities* e *BRPositions*.



USCities HCImages BRPositions

Figura 21 – Número de acessos ao arquivo de dados, variando a seletividade da condição de busca.

Em relação à quantidade de acessos a índice (Figura 22), o comportamento nesta fase foi diferente da anterior: a quantidade de acessos do MAM *cx-Sim* decresceu a medida que a seletividade aumentou. Tal comportamento é esperado, pois sua arquitetura assegura que serão analisadas somente as tuplas do índice complexo com a condição tradicional já validada, ou seja, as operações custosas de similaridade serão executadas apenas com dados que têm condições reais de compor o conjunto resposta. Os MAMs *Covering-Slim* e *Table-Slim*, por sua vez, não apresentam esta característica restritiva. Já o plano de execução concorrente, *B-Slim*, também apresentou decréscimo nas bases georreferenciadas, mas o número absoluto de acessos foi da ordem de 10 vezes maior.



USCities HCImages BRPositions

Figura 22 – Número de acessos aos blocos de índices, variando a seletividade da condição de busca.

O mesmo comportamento também pode ser observado em relação a:

- Quantidade total de acessos a disco (Figura 23);
- Quantidade de comparações entre dados tradicionais (Figura 24);

Em relação à quantidade de cálculos de distância entre pares de dados complexos, de acordo com informações apresentadas na Figura 25, o comportamento decrescente à medida que a seletividade aumenta se repete para o método *cx-Sim*. Mas agora este é em relação aos outros três métodos, *B-Slim*, *Covering-Slim* e *Table-Slim*. Assim como mencionado no experimento com variação do k, vale a pena observar que os números obtidos pelos demais métodos em comparação são idênticos, pois todos armazenam a mesma massa de dados complexos em apenas uma *Slim-tree*. O *cx-Sim*, por sua vez, o faz de maneira particionada na floresta de árvores métricas organizada no segundo nível do MAM.

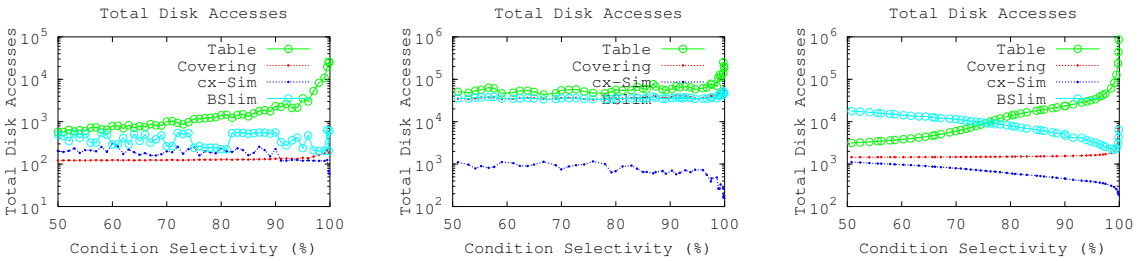


Figura 23 – Quantidade total de acessos a disco, variando a seletividade da condição de busca.

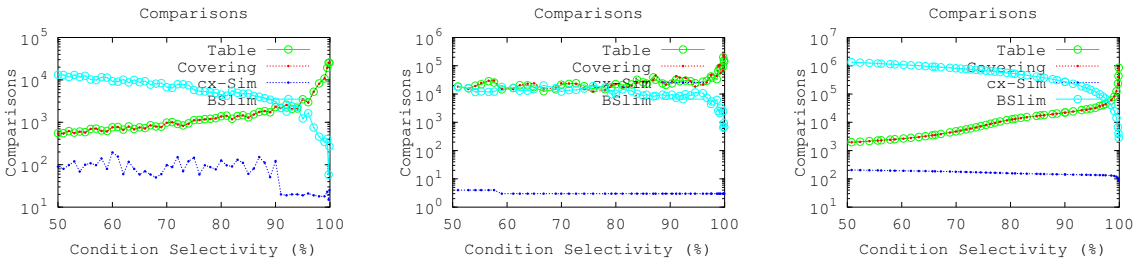


Figura 24 – Quantidade de comparações, variando a seletividade da condição de busca.

Como na primeira fase dos experimentos, o método *cx-Sim* também apresentou valores maiores que o *B-Slim*, o *Table-Slim* e o *Covering-Slim* em relação à quantidade de dados adicionados ao conjunto resposta durante as buscas, particularmente nos conjuntos de dados georreferenciados. Isto pode ser observado na Figura 26.

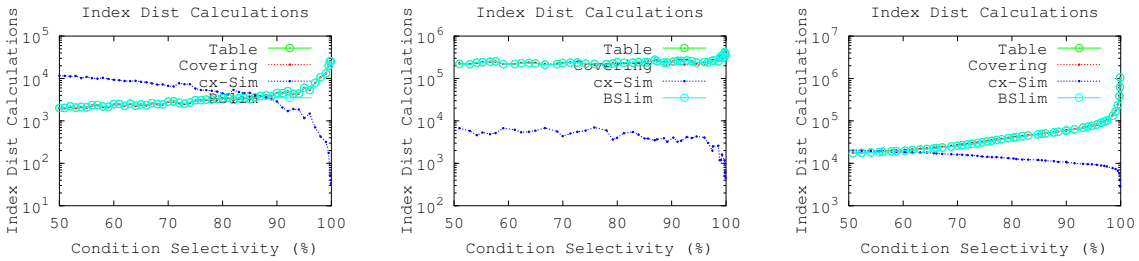


Figura 25 – Quantidade de cálculos de distância no índice, variando a seletividade da condição de busca.

Considerando-se o tempo de processamento e a base de dados *USCities*, a menor delas, os três métodos utilizados na comparação (*B-Slim*, *Table-Slim* e *Covering-Slim*) obtiveram resultados melhores que o *cx-Sim* até que a seletividade alcançasse a faixa de 90%. Estes foram melhores na ordem de, no máximo, 74%, 73% e 81%. No entanto, o método *cx-Sim* começou a reduzir o tempo de execução enquanto o *Covering-Slim* não, de acordo com a Figura 27. Em relação à base de dados *HCImages*, o tempo médio de busca do método *cx-Sim* foi de até 96 %, 95% e 98% menor que o *B-Slim*, o *Covering-Slim* e o *Table-Slim*, respectivamente. A base de dados *BRPositions*, por sua vez, apresentou diferenças da ordem de até 98%, 79% e 84% menores que o o *B-Slim*, o *Covering-Slim* e o *Table-Slim*, respectivamente. Vale observar que o *B-Slim* apresentou uma queda mais acentuada a partir de 90% de seletividade, mas mesmo assim o plano de execução alternativo não teve resultado próximo ao do *cx-Sim*.

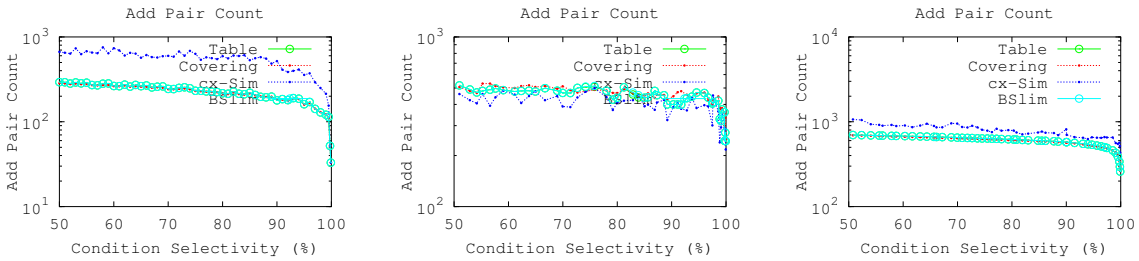


Figura 26 – Número de operações de fila no conjunto resposta (número de inserções na estrutura de dados auxiliar), variando a seletividade da condição de busca.

5.3 Abordagens para Consultas por Similaridade com Condições Adicionais Compostas

Nesta seção são apresentados os resultados dos experimentos que avaliaram o desempenho das variações da *cx-Sim tree* na resolução de consultas por similaridade es-

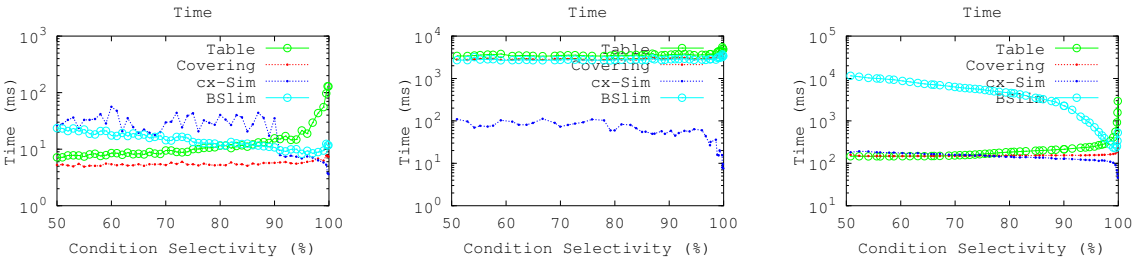


Figura 27 – Tempo de processamento das consultas, variando a seletividade da condição de busca.

tendidas com condições compostas. São comparadas a *cx-Sim tree* armazenando apenas um atributo tradicional, que verifica a outra parte da condição fazendo um acesso à tabela de dados, identificada como *cx-Sim Simple tree*, e as três derivações da *cx-Sim tree* desenvolvidas para indexação de múltiplos atributos tradicionais: *cx-Sim Covering tree*, *cx-Sim Composite tree* e *cx-Sim Chained tree*. De forma semelhante à seção anterior, a Subseção 5.3.1 mostra resultados de experimentos variando-se o número de vizinhos e a Subseção 5.3.2 experimentos variando-se a seletividade da condição adicional composta.

5.3.1 Experimentos com Variação no Número de Vizinhos Mais Próximos

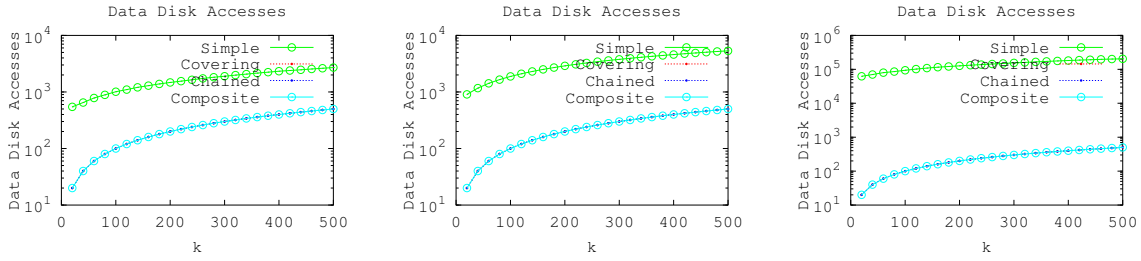
Assim como nos experimentos para condições simples, esta avaliação para condições compostas é iniciada com a variação no número de vizinhos mais próximos (k), cuja seletividade das condições de busca nos três conjuntos de dados é apresentada na Tabela 3. Por uma questão de simplificação, ao mencionar cada uma das derivações do método *cx-Sim tree* original, estes serão tratados apenas como *Simple*, *Covering*, *Chained* e *Composite*.

Tabela 3 – Seletividade das consultas submetidas no experimento com condições compostas e variação no número de vizinhos mais próximos.

Dataset	**USCities**	**HCImages**	**BRPositions**
Seletividade (condição 1º atributo)	74,5%	74,98%	74,92%
Seletividade (condição composta)	90,08%	88,29%	90,06%

Considerando-se a Figura 28 onde são apresentados os números de acessos ao arquivo de dados, tal análise mostra que os métodos *Covering*, *Chained* e *Composite* apresentam quantidades significantemente menores que o *Simple*. Os três primeiros necessitaram somente k acessos a disco por execução enquanto o quarto atingiu ordens de 5, 8 e 408 vezes mais acessos a disco para os conjuntos de dados *USCities*, *HCImages* e *BRPositions*, respectivamente. Tal diferença é aceitável analisando-se o esforço necessário para decidir se uma tupla será ou não adicionada ao conjunto resposta. Enquanto o

Simple necessita de um acesso a disco para obter o dado complexo e outro para acessar o segundo atributo da condição composta, os demais métodos necessitam de apenas um acesso pra analisar os dois.



USCities HCImages BRPositions

Figura 28 – Número de acessos ao arquivo de dados, variando o número de vizinhos (k).

Considerando-se acessos a índice, a análise precisa ser dividida pelo tipo de base de dados, como pode ser observado na Figura 29. Considerando-se as bases georreferenciadas (*USCities* e *BRPositions*), os métodos *Chained* e *Composite* cujo atributo tradicional (composto) está armazenado em *B-tree(s)* obtiveram acessos da ordem de 16 e 4 vezes mais que os métodos *Simple* e *Covering*. Já em relação à base de dados multimídia (*HCImages*), os métodos *Simple* e *Covering*– cuja validação do segundo atributo tradicional é pós-processada – tiveram resultados da ordem de 50% maiores que os métodos *Chained* e *Composite*. Outras informações importantes sobre os acessos a índice estão representadas nas Tabelas 4 e 5. Quanto à proporção no tamanho das tuplas indexadas, enquanto nas bases georreferenciadas o tamanho da tupla tradicional indexada é equivalente ao tamanho da tupla complexa indexada, na base multimídia a tupla com o dado complexo equivale a 128 tuplas com o dado tradicional. Já em relação ao fator de bloco (*Blocking factor* – Bfr), i.e. o número de registro que cabem em um bloco (ou página) de disco, enquanto que nas bases *USCities* e *BRPositions* existem quatro vezes mais dados complexos do que tradicionais por bloco, na base *HCImages* essa relação cai para $\frac{1}{32}$. Consequentemente, a validação da similaridade na base multimídia exige quantidade muito maior de acessos a índice do que nas georreferenciadas.

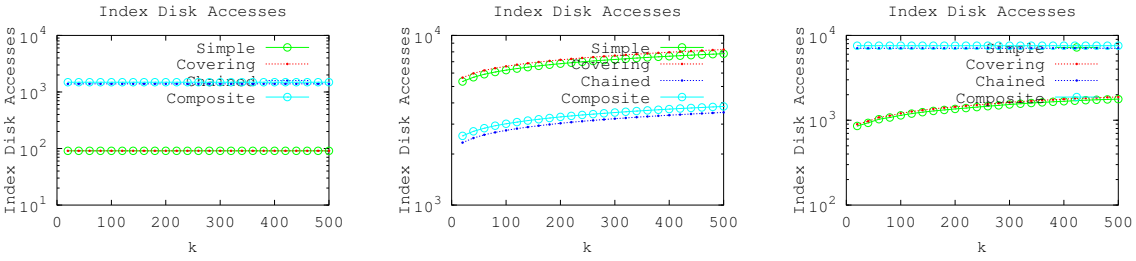
Tabela 4 – Proporção entre a tupla tradicional indexada no nível l_1 e a tupla complexa indexada no nível l_2, para as bases *USCities*, *HCImages* e *BRPositions*.

	USCities	**HCImages**	**BRPositions**
Dado Tradicional (l_1)	2x long	2x long	2x long
Dado Complexo (l_2)	2x long	256x long	2x long
Proporção (l_2/l_1)	1	128	1

Analisando o experimento em relação à quantidade total de acessos a disco, o *Simple* permanece como a derivação mais onerosa para as três bases (Figura 30). Isso

Tabela 5 – Comparativo entre B (block size), R (record size) e Bfr (Blocking factor) para as bases *USCities*, *HCImages* e *BRPositions*.

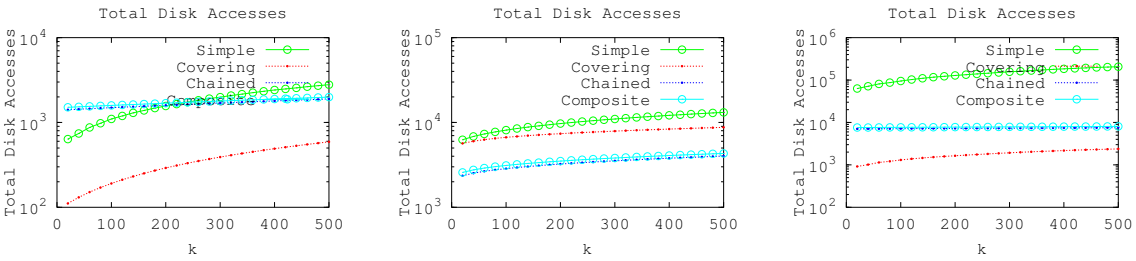
	USCities		**HCImages**		**BRPositions**	
	l_1	l_2	l_1	l_2	l_1	l_2
***B* (block size)**	4096k	16384k	4096k	16384k	4096k	16384k
***R* (record size)**	8bytes	8bytes	8bytes	1k	8bytes	8bytes
Bfr $(\lfloor B/R \rfloor)$	512	2048	512	16	512	2048



USCities HCImages BRPositions

Figura 29 – Número de acessos a blocos de índice, variando o número de vizinhos (k).

deve-se ao acesso adicional ao arquivo de dados, necessário para pós-validar a condição adicional de cada tupla já validada segundo com o critério de similaridade. Processo este desnecessário para as demais derivações.



USCities HCImages BRPositions

Figura 30 – Quantidade total de acessos a disco, variando o número de vizinhos (k).

Em relação à quantidade de comparações entre os dados tradicionais, pode ser notado na Figura 31 que, devido a arquitetura em dois níveis, uma vez fixada a condição adicional este MAM apresenta quantidade constante de comparações. Tal comportamento independe do k selecionado, pois o esforço para a busca na B^+-*tree* é sempre o mesmo. Quanto ao comparativo entre os métodos, é claramente notável que o *Composite* necessita de muito mais comparações do que os outros três métodos. Essa quantidade adicional é da ordem de 34, 8 e 893 vezes mais que os demais para as bases *USCities*, *HCImages* e *BRPositions*, respectivamente. Tal diferença deve-se justamente pelo fator que, em relação ao armazenamento das chaves, é tido como benéfico: armazenar os dados tradicionais em

apenas uma estrutura de dados e, consequentemente, um arquivo físico. Com isso, a altura da B^+-*tree* no *Composite* é consideravelmente maior que nos demais.

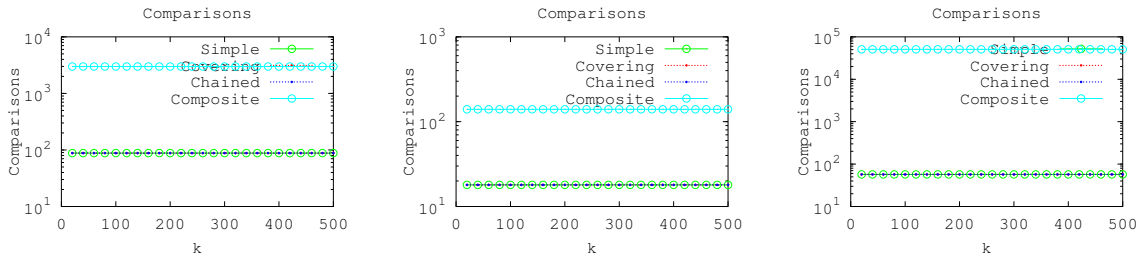


Figura 31 – Quantidade de comparações, variando o número de vizinhos (k).

Quanto à quantidade de cálculos de distância entre pares de dados complexos, de acordo com informações apresentadas na Figura 32, fica clara a eficiência dos métodos que pré-validam a condição adicional integralmente antes de verificar a similaridade sobre os que pós-validam parte dela. Isso deve-se ao fato de que os métodos *Simple* e *Covering* armazenarem a massa de dados complexos em apenas uma *Slim-tree*, enquanto que os métodos *Chained* e *Composite* o fazem através de uma floresta de árvores dinâmicas menores. Portanto, o número de cálculos de distância necessários para a poda da *Slim-tree* única é consideravelmente maior que na floresta de árvores menores. Tal diferença é de até 3, 2 e 6 vezes maior para as bases *USCities*, *HCImages* e *BRPositions*, respectivamente.

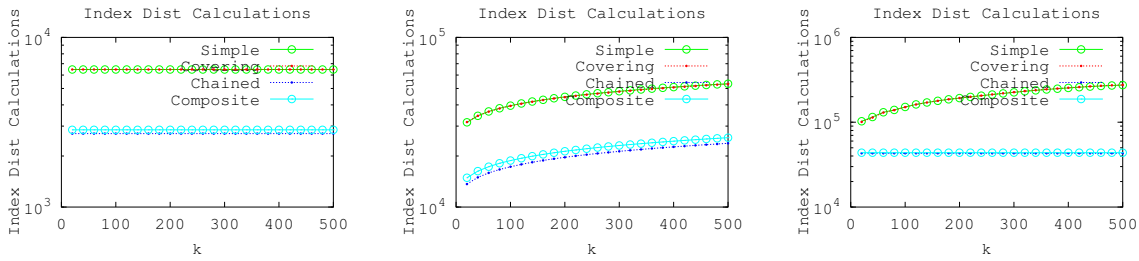


Figura 32 – Número de cálculos de distância no índice, variando o número de vizinhos (k).

Para finalizar a análise dos experimentos com condições compostas e variando o k, é apresentado o comparativo entre os tempos médios de busca na Figura 33. Assim como na análise da quantidade de acessos a disco, os resultados também foram diferentes dado o tipo de base de dados: georreferenciada ou multimídia. Em relação à base de dados *HCImages*, os métodos *Chained* e *Composite* apresentaram tempo médio de busca 50% menores que os métodos *Simple* e *Covering*. Em relação às bases *USCities* e *BRPositions*, os resultados se inverteram de modo que o método *Covering* é que apresentou melhores

resultados: da ordem de até 8 e 11 vezes menor, respectivamente. Tal fato é justificável para os casos onde a pré-validação da condição adicional composta resulte num particionamento da massa de dados complexos em uma floresta de árvores dinâmicas pequenas. Nesta condição, a verificação da similaridade não se beneficia das operações de poda das subárvores e necessita acessar individualmente uma quantidade muito maior de dados complexos (vide maior quantidade de acessos a índice apresentado na Figura 29). O método *Covering*, por sua vez, particiona os dados complexos apenas pelo primeiro atributo tradicional resultando em uma floresta de árvores dinâmicas mais volumosas. Essas por sua vez beneficiam-se das operações de poda e, mesmo com a pós-validação dos demais atributos tradicionais, proporciona um tempo médio de busca consideravelmente menor.

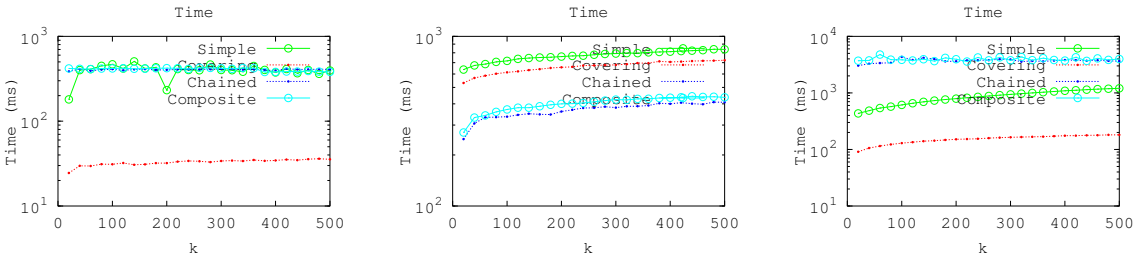


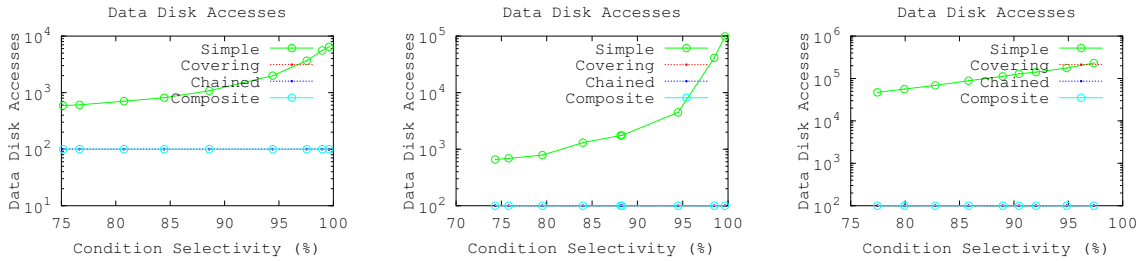
Figura 33 – Tempo de processamento das consultas, variando o número de vizinhos (k).

Encerradas as comparações variando a quantidade de vizinhos mais próximos, a seção a seguir apresenta os resultados dos experimentos cuja variação ocorreu na seletividade da condição de busca.

5.3.2 Experimentos com Variação na Seletividade da Condição de Busca

Esta segunda parte dos experimentos considera a variação na seletividade da condição de busca. Neste teste, o k – variável do teste anterior – também foi fixado em 100, ou seja, deseja-se recuperar os 100 vizinhos mais próximos ao objeto de consulta.

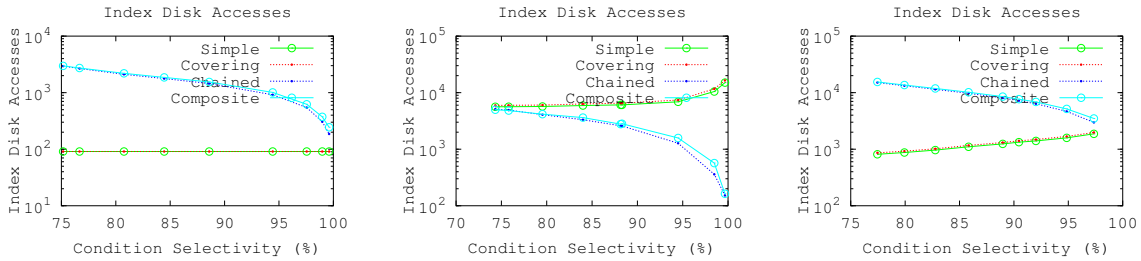
Iniciando-se pela quantidade de acessos ao arquivo de dados (Figura 34), tal análise mostra que os resultados são análogos aos do experimento variando k. Novamente os métodos *Covering*, *Chained* e *Composite* apresentam quantidades significantemente menores que o *Simple*. Os três primeiros necessitaram somente k acessos a disco por execução enquanto o quarto atingiu ordens de até 65, 986 e 2312 vezes mais acessos a disco para os conjuntos de dados *USCities*, *HCImages* e *BRPositions*, respectivamente. Tal diferença é aceitável analisando-se o esforço necessário para decidir se uma tupla será ou não adicionada ao conjunto resposta. Enquanto o *Simple* necessita de um acesso a disco para obter o dado complexo e outro para acessar o segundo atributo da condição composta, os demais métodos necessitam de apenas um acesso pra analisar os dois.



USCities HCImages BRPositions

Figura 34 – Número de acessos ao arquivo de dados, variando a seletividade da condição de busca.

Considerando-se acessos a índice, da mesma forma que na seção anterior com variação de k, a análise precisa ser dividida pelo tipo de base de dados, como pode ser observado na Figura 35. Considerando-se as bases georreferenciadas (*USCities* e *BRPositions*), os métodos *Chained* e *Composite*– cujo atributo tradicional (composto) está armazenado em B^+*-trees*– obtiveram decréscimo no número de acessos a índice com o avanço da seletividade mas sempre se mantiveram maiores que os métodos *Simple* e *Covering*. Já em relação à base de dados multimídia (*HCImages*), os métodos *Simple* e *Covering*– cuja validação do segundo atributo tradicional é pós-processada – também obtiveram decréscimo no número de acessos a índice com o avanço da seletividade mas sempre menores que os métodos *Chained* e *Composite*. As mesmas informações sobre a proporção no tamanho das tuplas indexadas e o *Bfr* apresentadas nas Tabelas 4 e 5, e explicadas na seção anterior, justificam tal comportamento. Consequentemente, a validação da similaridade na base multimídia exige quantidade muito maior de acessos a índice do que nas georreferenciadas.

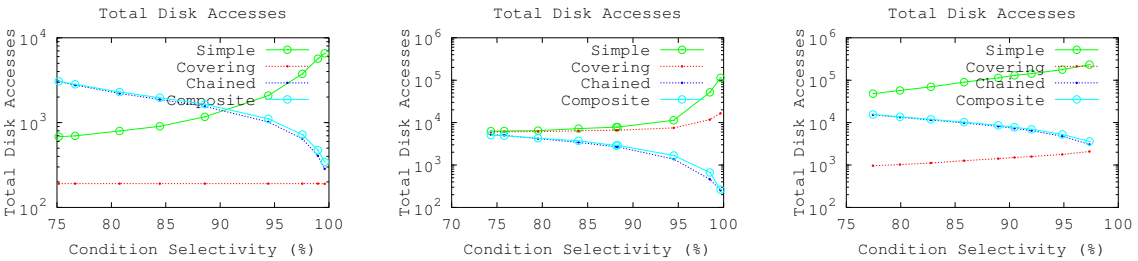


USCities HCImages BRPositions

Figura 35 – Número de acessos aos blocos de índices, variando a seletividade da condição de busca.

Analisando-se o experimento em relação à quantidade total de acessos a disco, o *Simple* permanece como a derivação mais onerosa, predominantemente para as bases *HCImages* e *BRPositions* (Figura 36). Isso novamente deve-se ao acesso adicional ao arquivo de dados, necessário para pós-validar a condição adicional de cada tupla imediatamente

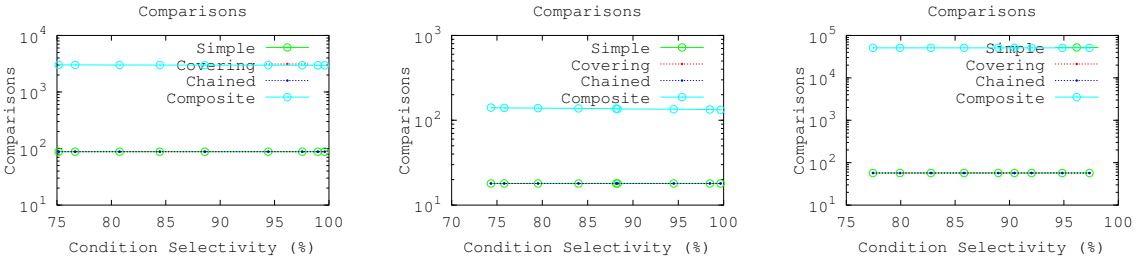
após verificar o critério de similaridade. Processo este desnecessário para as demais derivações.



USCities HCImages BRPositions

Figura 36 – Quantidade total de acessos a disco, variando a seletividade da condição de busca.

Em relação à quantidade de comparações entre os dados tradicionais, o resultado para os experimentos cuja variação ocorreu na seletividade da consulta é idêntico aos experimentos com variação no número de vizinhos mais próximos (k). Este pode ser verificado na Figura 37 e comparado à Figura 31. Quanto ao comparativo entre os métodos, é notável que o *Composite* executa muito mais comparações do que os outros três métodos e, essa quantidade adicional é da ordem de 34, 8 e 895 vezes mais que os demais para as bases *USCities*, *HCImages* e *BRPositions*, respectivamente. Tal diferença também deve-se justamente pelo fator que, em relação ao armazenamento das chaves, é tido como benéfico: armazenar os dados tradicionais em apenas uma estrutura de dados e, consequentemente, um arquivo físico. Com isso, a altura da B^+*-tree* no *Composite* é consideravelmente maior que nos demais.



USCities HCImages BRPositions

Figura 37 – Quantidade de comparações, variando a seletividade da condição de busca.

Quanto à quantidade de cálculos de distância entre pares de dados complexos, de acordo com informações apresentadas na Figura 38, fica clara também para experimentos com a variação na seletividade a eficiência dos métodos que pré-validam a condição adicional integralmente antes de verificar a similaridade sobre os que pós-validam parte dela. Isso deve-se ao fato de os métodos *Simple* e *Covering* armazenarem a massa de dados

complexos em apenas uma *Slim-tree*, enquanto os métodos *Chained* e *Composite* o fazem através de uma floresta de árvores dinâmicas menores. Portanto, o número de cálculos de distância necessários para a poda da *Slim-tree* única é maior que na floresta de árvores menores. Tal diferença é de até 41, 140 e 20 vezes maior para as bases *USCities*, *HCImages* e *BRPositions*, respectivamente.

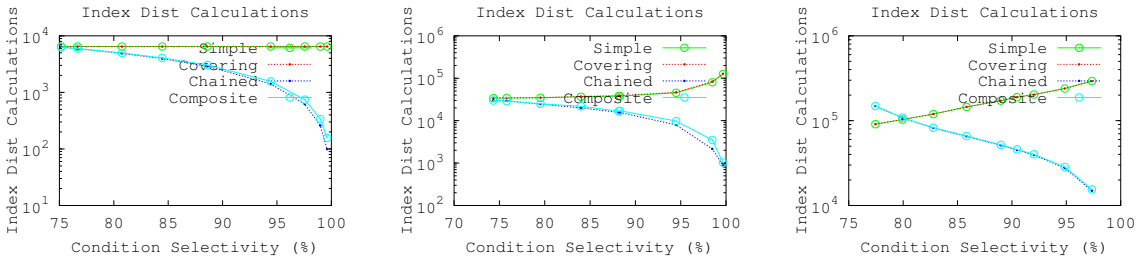


Figura 38 – Quantidade de cálculos de distância no índice, variando a seletividade da condição de busca.

Para finalizar a análise dos experimentos com condições compostas e variação na seletividade, é apresentado o comparativo entre os tempos médios de busca na Figura 39. Novamente, assim como nos experimentos relacionados a acessos a índice, os resultados foram diferentes dado o tipo de base de dados: georreferenciada ou multimídia. Em relação à base de dados *HCImages*, os métodos *Chained* e *Composite* apresentaram tempo médio de busca até 75 vezes menores que os métodos *Simple* e *Covering*. Já em relação às bases *USCities* e *BRPositions*, os resultados se inverteram de modo que os métodos *Covering* é que apresentou melhores resultados independentemente da variação na seletividade. Assim como no experimento cuja variação era em k, tal fato é justificavel para os casos onde a pré-validação da condição adicional composta resulte num particionamento da massa de dados complexos em uma floresta de árvores dinâmicas pequenas. Nesta condição, a verificação da similaridade não se beneficia das operações de poda das subárvores e necessita acessar individualmente uma quantidade muito maior de dados complexos (vide maior quantidade de acessos a índice apresentado na Figura 35). O método *cx-Sim Covering tree*, por sua vez, particiona os dados complexos apenas pelo primeiro atributo tradicional resultando em uma floresta de árvores dinâmicas mais volumosas. Essas por sua vez beneficiam-se das operações de poda e, mesmo com a pós-validação dos demais atributos tradicionais, proporcionam um tempo médio de busca consideravelmente menor.

5.4 Considerações Finais

Este capítulo apresentou em detalhes o modo como o método *cx-Sim tree* e suas variações foram validadas experimentalmente. Estes experimentos validaram as consultas

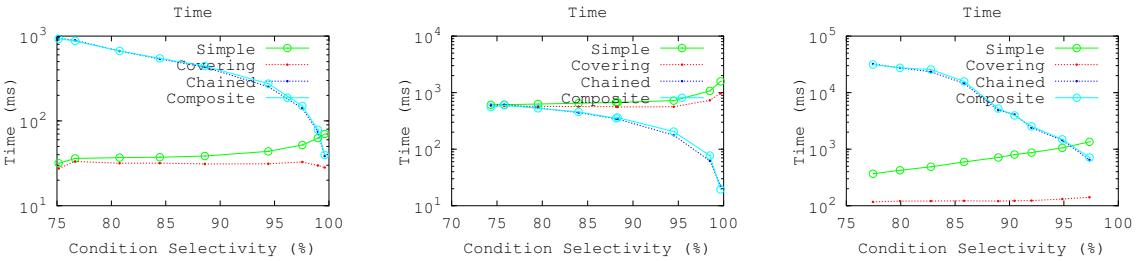


Figura 39 – Tempo de processamento das consultas, variando a seletividade da condição de busca.

por similaridade em duas abordagens: a primeira com condições simples e a segunda com condições compostas. Em ambas, os experimentos foram executados com três bases de dados reais e tiveram variações, tanto em relação ao número de vizinhos mais próximos consultados quanto à seletividade da condição de busca.

Considerando consultas por similaridade com condições simples, os resultados confirmaram que a *cx-Sim tree* tem desempenho superior aos métodos concorrentes, passando a figurar como estado da arte no que tange a execução deste tipo de consultas. Com relação às consultas com condições compostas, pôde-se verificar duas situações distintas considerando-se a ordem da validação do segundo atributo tradicional em relação a validação da similaridade, indicadas a seguir.

- **Pós-validação**: o segundo atributo tradicional é validado após a verificação da similaridade, portanto, em um terceiro estágio da busca. As estruturas *cx-Sim Simple tree* e *cx-Sim Covering tree* apresentam tal propriedade e são melhor empregadas em massas de dados cuja seletividade é alta já no primeiro atributo da chave. Dessa forma, já se obtém grande particionamento dos dados para a verificação da similaridade e, em último estágio, ocorre a validação pontual do segundo atributo.

- **Pré-validação**: os atributos tradicionais são integralmente validados antes da verificação de similaridade, visto que esta última é a operação mais onerosa do processo. As estruturas *cx-Sim Chained tree* e *cx-Sim Composite tree* apresentam tal propriedade e são melhor empregadas em massas de dados cuja seletividade é alta ao se considerar também o segundo atributo da chave e não somente o primeiro. Como o tempo de execução das buscas através do *cx-Sim Chained tree* é suavemente menor que o *cx-Sim Composite tree*, caso a granularidade de arquivos físicos e um tempo sensivelmente maior para a criação do MAM não sejam impeditivos, a recomendação é priorizar o primeiro.

No próximo capítulo é apresentada a conclusão desta dissertação.

6 CONCLUSÃO

Esta dissertação de mestrado propôs um novo novo método de acesso métrico capaz de responder à consultas por similaridade com condições adicionas. O método possui derivações com suas respectivas particularidades, mas desde o seu caso base beneficia-se diretamente da associação de dados tradicionais aos dados complexos, pois agrega a noção de ordem total a base de dados indexada, até então não ordenável. Com isso, torna-se possível a adição de atributos tradicionais ao predicado das consultas por similaridade, enriquecendo as possibilidades de filtro sobre os dados indexados.

Um outro benefício em relação ao MAM é estrutural e resulta de sua arquitetura em dois níveis. Os dados tradicionais são armazenados em estrutura dedicada e sem repetição (nível l_1), independentemente de quantos atributos complexos estão diretamente relacionados aos mesmos. Esta organização de atributos tradicionais distintos proporciona um particionamento da massa de dados complexos, de modo que possam existir agrupamentos de dados que compartilhem do mesmo atributo tradicional (nível l_2). Consequentemente, as onerosas operações necessárias para aferir a similaridade entre pares de dados complexos passam a ser executadas apenas para elementos que realmente tenham a possibilidade real de compor o conjunto resposta.

A Figura 40 ilustra a aplicabilidade dos MAMs propostos em uma consulta por similaridade e a seção 6.1 apresenta as principais contibuições desta dissertação.

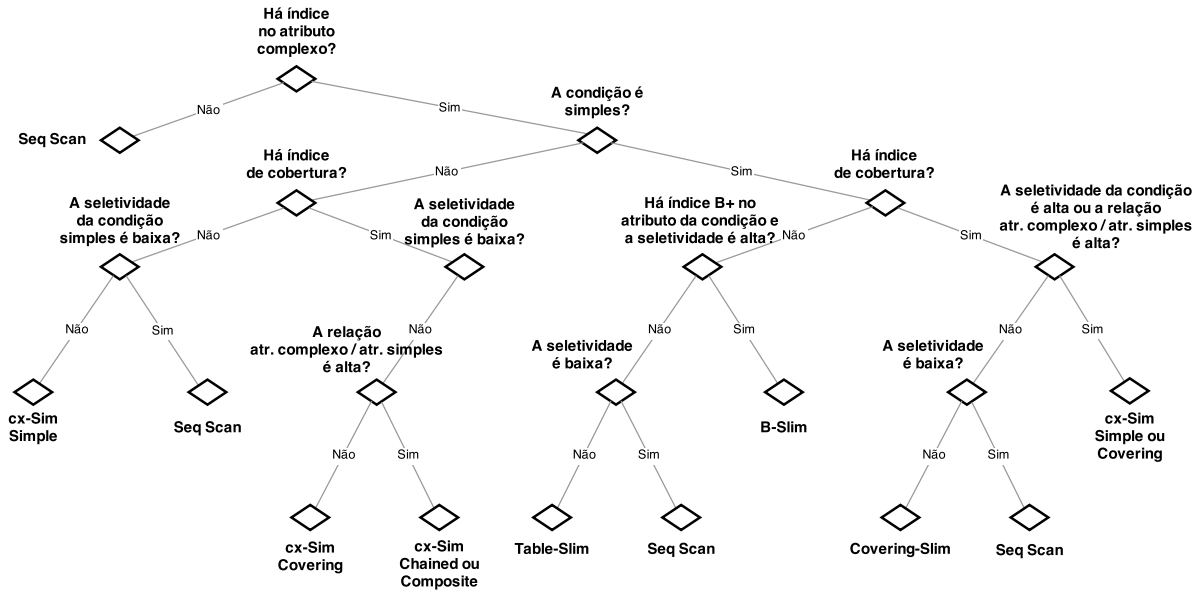


Figura 40 – Mapeamento da aplicabilidade dos MAMs propostos/referenciados em uma consulta por similaridade.

6.1 Principais Contribuições

a) Desenvolvimento de um novo MAM eficiente para responder a consultas por similaridade estendidas com condições.

O método apresentado, além de atender a consultas por similaridade com condições adicionais, tem a propriedade de pós-validar outros atributos tradicionais que não estejam no MAM. Dessa maneira, ele permite com que, após a verificação da similaridade e por meio de um acesso adicional ao arquivo de dados, sejam respondidas consultas por similaridade com condições adicionais compostas, ou seja, com mais de um atributo tradicional em seu predicado. Por se tratar de um acesso ao arquivo de dados, a pós-validação independe da criação antecipada de um índice adequado a consulta.

Ganhos em eficiência podem ser verificados nos resultados apresentados na Seção 5.2, onde o método *cx-Sim tree*, em seu caso geral (i.e. indexando apenas um atributo tradicional), foi aplicado a consultas por similaridade com condições adicionais. O método foi submetido a comparação com seus *baselines Table-Slim* e *Covering-Slim*, além do *B-Slim* que trata-se de um novo plano de execução implementado para enriquecer o caráter comparativo da pesquisa. Os resultados obtidos podem ser resumidos em:

(i) Redução na quantidade de acessos ao arquivo de dados;

(ii) Redução na quantidade total de acessos a disco;

(iii) Redução na quantidade de comparações entre dados tradicionais;

(iv) Redução na quantidade de cálculos de distância entre pares de dados complexos;

(v) Redução no tempo médio de processamento da consulta.

b) Desenvolvimento de uma variação que otimiza a execução de consultas por similaridade estendidas com condições compostas através de um pós-processamento eficiente.

Foi desenvolvida a variação *cx-Sim Covering tree*, que proporciona a pós-validação de outros atributos tradicionais eliminando acessos adicionais ao arquivo de dados. O feito é possível através da criação de um índice onde todos os atributos tradicionais passíveis de análise no predicado da consulta devem ser armazenados no MAM, junto ao vetor de características e ao *rowId*. Esta abordagem proporciona ganho expressivo de desempenho e se exime de acessos ao arquivo de dados em detrimento de ocupar um maior espaço em disco com o armazenamento dos novos atributos tradicionais.

Esta abordagem apresenta a possibilidade de pós-validar atributos tradicionais e é indicada para bases de dados cuja seletividade da condição simples é alta. Dessa maneira,

apenas o primeiro atributo tradicional é utilizado para particionar os dados complexos, resultando em árvores dinâmicas (em l_2) que se beneficiam das operações de poda por distância. A verificação dos demais atributos tradicionais, por sua vez, pode naturalmente ser pós-validada sem ônus à busca.

c) Desenvolvimento de variações que priorizam um pré-processamento eficiente para agilizar a execução de consultas por similaridade com condições compostas.

O segundo conjunto de derivações do método *cx-Sim tree* desenvolvido traz como benefício a pré-validação de toda a condição adicional, ou seja, todos os atributos tradicionais passíveis de filtro no predicado das consultas são validados antes de verificar a similaridade. Seu objetivo principal é, justamente, eximir por completo a possibilidade de executar acessos a disco e cálculos de distância desnecessários durante a verificação da similaridade, pois são as operações mais onerosas do processo.

A primeira derivação que pré-valida atributos tradicionais trata-se do *cx-Sim Chained tree*. A sua abordagem de armazenamento aninhado dos atributos tradicionais em l_1 proporciona um maior particionamento dos dados complexos, em detrimento de um aumento considerável no número de arquivos físicos que armazenam a parte tradicional do MAM. A segunda derivação que pré-valida atributos tradicionais trata-se do *cx-Sim Composite tree*. Este, por sua vez, possibilita o mesmo nível de particionamento dos dados complexos proporcionado pelo *cx-Sim Chained tree*, embora sua chave para a indexação dos atributos tradicionais seja composta. Dessa maneira, não existe aninhamento de atributos e nem granularidade de arquivos físicos que armazenam tal massa de dados.

Estas duas abordagens que pré-validam a condição tradicional por completo são indicadas para bases de dados cuja seletividade do primeiro atributo não é alta, mas a da composição de atributos o é. Dessa maneira, não é despendido esforço verificando a similaridade entre elementos complexos que seriam falsos positivos no que tange apenas a condição adicional.

6.2 Trabalhos Futuros

Avaliando as particularidades do método proposto e suas derivações, nota-se que é vantajoso pré-validar toda a condição tradicional, desde que o particionamento dos dados complexos resulte em árvores dinâmicas expressivas e/ou largas, ao ponto de beneficiarem-se das operações de poda por distância. Consequentemente, não é vantajoso pré-validar atributos complexos que resultem em árvores dinâmicas baixas. Por outro lado, é vantajoso manter uma árvore dinâmica expressiva e/ou larga (que se beneficie da operação de poda por distância), mesmo que pra isso seja necessário pós-validar a segunda parte da condição adicional. Ou seja, nota-se que uma abordagem nova, mais abrangente, dinâmica e que

busque: a) um ótimo tempo de execução para os casos pré-validados e b) um bom tempo de execução para os casos pós-validados, pode ser obtida com a utilização das melhores abordagens de cada conjunto: *cx-Sim Covering tree* e *cx-Sim Chained tree*.

Portanto, torna-se possível almejar um método *cx-Sim tree* mais amplo, onde:

- Até que se atinja uma quantidade mínima (*tradNodeOverflow*) de chaves complexas inseridas no MAM compartilhando da mesma composição de atributos tradicionais, a abordagem de indexação utilizada é a pós-validada. Naturalmente, a árvore dinâmica apresenta dados complexos que não compartilham do mesmo atributo tradicional, ou seja, indexados por um intervalo de atributos tradicionais.
- Atingido o *tradNodeOverflow*, a árvore dinâmica em questão é subdividida (*split*) em duas novas árvores dinâmicas, com quantidade de tuplas o mais próximo possível e chaves sequenciais, de maneira que surjam duas novas árvores indexadas por intervalos disjuntos e atendendo às condições do item acima.
- A sequência de operações de *split* se repete até que, ao subdividir a árvore, uma das metades seja composta apenas de dados complexos que compartilhem do mesmo atributo tradicional. Consequentemente, a abordadem empregada passa a ser a pré-validada e a árvore dinâmica indexada apenas por um atributo tradicional.

A partir deste momento, pode-se criar uma nova estrutura, adaptável ao ponto de se reorganizar e aplicar: i) uma abordagem especialista para casos particulares de onde podem ser respondidas consultas com ótimos tempos de busca e, ii) uma abordagem generalista para casos gerais de onde podem ser respondidas consultas com bons tempos de busca.

REFERÊNCIAS

[1] POLA, I. R. V. *Explorando Conceitos da Teoria de Espacos Métricos em Consultas por Similaridade Sobre Dados Complexos*. Tese (Doutorado) — University of São Paulo, Brazil, 2010.

[2] BARIONI, M. C. N. et al. Seamlessly integrating similarity queries in SQL. *Software Practice and Experience*, John Wiley & Sons, Inc., New York, NY, USA, v. 39, n. 4, p. 355–384, 2009. ISSN 0038-0644. Disponível em: <http://dx.doi.org/10.1002/spe.v39:4>.

[3] 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.

[4] CHÁVEZ, E. et al. Searching in metric spaces. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 33, n. 3, p. 273–321, 2001. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/502807.502808>.

[5] ESULI, A. Use of permutation prefixes for efficient and scalable approximate similarity search. *Inf. Process. Manage.*, v. 48, n. 5, p. 889–902, 2012.

[6] PATELLA, M.; CIACCIA, P. The Many Facets of Approximate Similarity Search. In: *SISAP*. [S.l.: s.n.], 2008. p. 10–21.

[7] PATELLA, M.; CIACCIA, P. Approximate similarity search: A multi-faceted problem. *J. Discrete Algorithms*, v. 7, n. 1, p. 36–48, 2009.

[8] FERREIRA, M. R. P. *Suporte a consultas por similaridade unárias em SQL*. Dissertação (Mestrado) — University of São Paulo, Brazil, 2008.

[9] KASTER, D. S. *Tratamento de condições especiais para busca por similaridade em bancos de dados complexos*. 177 p. Tese (Doutorado) — University of São Paulo, Brazil, 2012.

[10] YEUNG, A. K. W.; HALL, G. B. *Spatial Database Systems: Design, Implementation and Project Management*. 1. ed. [S.l.]: Springer, 2007. (The GeoJournal Library, v. 87).

[11] DATTA, R. et al. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, v. 40, n. 2, 2008.

[12] JING, F. et al. Learning in Region-Based Image Retrieval. In: *CIVR*. [S.l.: s.n.], 2003. p. 206–215.

[13] GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. 3. ed. [S.l.]: Addison-Wesley, 2007. ISBN 0-201-50803-6.

[14] SANTINI, S.; GUPTA, A. A Wavelet Data Model for Image Databases. In: *IEEE Intl. Conf. on Multimedia and Expo*. Tokyo, Japan: IEEE Computer Society, 2001.

[15] KHOTANZAD, A.; HONG, Y. H. Invariant Image Recognition by Zernike Moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, v. 12, n. 5, p. 489–497, May 1990 1990.

[16] GAUKER, C. A Critique of the Similarity Space Theory of Concepts. *Mind & Language*, v. 22, n. 4, p. 317–345, 2007.

[17] TORRES, R. d. S. et al. A genetic programming framework for content-based image retrieval. *Pattern Recognition*, v. 42, n. 2, p. 283–292, 2009.

[18] BUGATTI, P. H.; TRAINA, A. J. M.; Traina Jr., C. Assessing the best integration between distance-function and image-feature to answer similarity queries. In: *SAC*. [S.l.: s.n.], 2008. p. 1225–1230.

[19] HE, X. et al. Learning and inferring a semantic space from user's relevance feedback for image retrieval. In: *ACM Multimedia*. [S.l.: s.n.], 2002. p. 343–346.

[20] WILSON, D. R.; MARTINEZ, T. R. Improved Heterogeneous Distance Functions. *J. Artif. Intell. Res. (JAIR)*, v. 6, p. 1–34, 1997.

[21] LEVENSHTEIN, V. I. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, v. 10, n. 8, p. 707–710, feb 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[22] ELMASRI, R.; NAVATHE, S. *Fundamentals of Database Systems*. 6th. ed. USA: Addison-Wesley Publishing Company, 2010. ISBN 0136086209, 9780136086208.

[23] BAYER, R.; MCCREIGHT, E. Organization and Maintenance of Large Ordered Indices. In: *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*. New York, NY, USA: ACM, 1970. (SIGFIDET '70), p. 107–141. Disponível em: <http://doi.acm.org/10.1145/1734663.1734671>.

[24] GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. *Database Systems: The Complete Book*. 2. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2008. ISBN 9780131873254.

[25] KNUTH, D. E. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998. ISBN 0-201-89685-0.

[26] COMER, D. Ubiquitous B-Tree. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 11, n. 2, p. 121–137, jun. 1979. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/356770.356776>.

[27] GAEDE, V.; GÜNTHER, O. Multidimensional Access Methods. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 30, n. 2, p. 170–231, jun. 1998. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/280277.280279>.

[28] 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>.

[29] BENTLEY, J. L.; FRIEDMAN, J. H. Data Structures for Range Searching. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 11, n. 4, p. 397–409, dez. 1979. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/356789.356797>.

[30] GUTTMAN, A. R-trees: A Dynamic Index Structure for Spatial Searching. In: *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 1984. (SIGMOD '84), p. 47–57. ISBN 0-89791-128-8. Disponível em: <http://doi.acm.org/10.1145/602259.602266>.

[31] BOZKAYA, T.; OZSOYOGLU, M. Distance-based Indexing for High-dimensional Metric Spaces. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 26, n. 2, p. 357–368, jun. 1997. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/253262.253345>.

[32] BERCHTOLD, S.; KEIM, D. A.; KRIEGEL, H.-P. The X-tree: An Index Structure for High-Dimensional Data. In: *Proceedings of the 22th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996. (VLDB '96), p. 28–39. ISBN 1-55860-382-4. Disponível em: <http://dl.acm.org/citation.cfm?id=645922.673502>.

[33] KULIS, B.; JAIN, P.; GRAUMAN, K. Fast Similarity Search for Learned Metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE Computer Society, Washington, DC, USA, v. 31, n. 12, p. 2143–2157, 2009. ISSN 0162-8828. Disponível em: <http://dx.doi.org/10.1109/TPAMI.2009.151>.

[34] SLANEY, M.; CASEY, M. Locality-Sensitive Hashing for Finding Nearest Neighbors [Lecture Notes]. *IEEE Signal Processing Magazine*, v. 25, n. 2, p. 128–131, 2008. ISSN 1053-5888.

[35] SUNDARAM, N. et al. Streaming Similarity Search over One Billion Tweets Using Parallel Locality-sensitive Hashing. *Proc. VLDB Endow.*, VLDB Endowment, v. 6, n. 14, p. 1930–1941, set. 2013. ISSN 2150-8097. Disponível em: <http://dl.acm.org/citation.cfm?id=2556549.2556574>.

[36] TRAINA, A. J. M.; Traina Jr., C. Similarity Search in Multimedia Databases. In: *Handbook of Video Databases: Design and Applications*. 1. ed. [S.l.]: Boca Raton: CRC Press, 2003. p. 711–738.

[37] CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree: An efficient access method for similarity search in metric spaces. In: *Proceedings of the International Conference on Very Large Databases*. Athens, Greece: [s.n.], 1997. p. 426–435.

[38] VIEIRA, M. R. et al. Revisiting the DBM-Tree. *Journal of Information and Data Management*, v. 1, n. 1, p. 129–132, 2010.

[39] Traina Jr., C. et al. The Omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. *VLDB Journal*, v. 16, n. 4, p. 483–505, 2007.

[40] YOUSRI, N.; ISMAIL, M.; KAMEL, M. Adaptive similarity search in metric trees. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. [S.l.: s.n.], 2007. p. 419–424.

[41] MCFEE, B.; LANCKRIET, G. R. G. Large-scale music similarity search with spatial trees. In: *International Society for Music Information Retrieval Conference*. [s.n.], 2011. p. 55–60. Disponível em: <http://ismir2011.ismir.net/papers/PS1-3.pdf>.

[42] PARK, D.-J.; KIM, H.-J. An Enhanced Technique for k-Nearest Neighbor Queries with Non-Spatial Selection Predicates. *Multimedia Tools and Applications*, Kluwer Academic Publishers, Hingham, MA, USA, v. 19, n. 1, p. 79–19, 2003. ISSN 1380-7501. Disponível em: <http://dx.doi.org/10.1023/A:1021121030238>.

[43] PARK, D.-J.; LEE, D.-H. Spy-Tec+ : an Integrated Index Structure for k-Nearest Neighbor Queries with Semantic Predicates in Multimedia Database. *International Journal of Software Engineering and Knowledge Engineering*, v. 21, n. 7, p. 989–1011, 2011.

[44] De Felipe, I.; HRISTIDIS, V.; RISHE, N. Keyword Search on Spatial Databases. In: *Proceedings of the IEEE International Conference on Data Engineering*. [s.n.], 2008. p. 656–665. ISBN 978-1-4244-1836-7. Disponível em: <http://dx.doi.org/10.1109/ICDE.2008.4497474>.

[45] CONG, G.; JENSEN, C. S.; WU, D. Efficient retrieval of the top-k most relevant spatial web objects. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 2, n. 1, p. 337–348, 2009. ISSN 2150-8097. Disponível em: <http://dl.acm.org/citation.cfm?id=1687627.1687666>.

[46] ROCHA-JUNIOR, J. B. et al. Efficient processing of top-k spatial keyword queries. In: *Proceedings of the 12th international conference on Advances in spatial and temporal databases*. Berlin, Heidelberg: Springer-Verlag, 2011. (SSTD'11), p. 205–222. ISBN 978-3-642-22921-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2035253.2035270>.

[47] CHEN, L.; CONG, G.; CAO, X. An efficient query indexing mechanism for filtering geo-textual data. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. [s.n.], 2013. p. 749–760. Disponível em: <http://doi.acm.org/10.1145/2463676.2465328>.

[48] SOARES, L. C.; KASTER, D. S. cx-Sim: A Metric Access Method for Similarity Queries with Additional Conditions. *JIDM*, v. 4, n. 3, p. 437–452, 2013.

TRABALHOS PUBLICADOS PELO AUTOR

Trabalhos publicados pelo autor durante o programa.

1. SOARES, L. C.; KASTER, D. S. cx-Sim: A Metric Access Method for Similarity Queries with Additional Conditions. *Journal of Information and Data Management*, v. 4, n. 3, p. 437–452, 2013 (Qualis CC 2012, B3).