Explorando o uso de árvores B+ na Indexação de Dados por Similaridade

Jéssica Naiara Batista de Farias

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2019

Jéssica Naiara Batista de Farias

Explorando o uso de árvores B+ na Indexação de Dados por Similaridade

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Profª Drª Maria Camila Nardini Barioni
Coorientador: Prof. Dr. Humberto Luiz Razente

Uberlândia
2019

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

F224e
2019

Farias, Jéssica Naiara Batista de, 1992-
Explorando o uso de árvores B+ na indexação de dados por similaridade [recurso eletrônico] / Jéssica Naiara Batista de Farias. -- 2019.

Orientadora: Maria Camila Nardini Barioni.
Coorientador: Humberto Luiz Razente.
Dissertação (Mestrado) -- Universidade Federal de Uberlândia, Programa de Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2026.11048
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Barioni, Maria Camila Nardini, 1978-, (Orient.). II. Razente, Humberto Luiz, 1977-, (Coorient.). III. Universidade Federal de Uberlândia. Programa de Pós-graduação em Ciência da Computação. IV. Título.

André Carlos Francisco
Bibliotecário(a)-Documentalista - CRB-6/3408

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Naves de Ávila, nº 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

DECLARAÇÃO

Processo nº 23117.016496/2020-83

Interessado: Coordenação do Programa de Pós-Graduação em Ciência da Computação

Declaramos, para os devidos fins, que **Jéssica Naiara Batista de Farias** foi aluna do curso de mestrado, na linha de pesquisa de Ciência de Dados, integralizou todos os créditos e foi aprovada na defesa da dissertação de mestrado realizada no dia 29/11/2019. O competente diploma será emitido após a entrega da versão final do trabalho.

ERISVALDO ARAUJO FIALHO
Secretário PPGCO
Portaria R Nº 423/09

Documento assinado eletronicamente por **Erisvaldo Araújo Fialho**, **Assistente em Administração**, em 05/03/2020, às 10:41, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1912706** e o código CRC **60E5D940**.

Referência: Processo nº 23117.016496/2020-83 SEI nº 1912706

Dedico esse trabalho a minha mãe, Sônia, por toda dedicação e esforço durante todos esses anos. Você me inspira.

Agradecimentos

Agradeço à minha mãe, Sônia, pela dedicação e apoio incondicional que tornaram essa trajetória possível.

Aos meus familiares, Washington, Jandira, Stefane e Erick, obrigada por fazerem parte dessa história.

A Sassá, agradeço por trazer alegria aos nossos dias; tudo é bem melhor desde que você chegou.

Em especial, à minha avó Iraci, sou grata pelo cuidado e afeto. Ao meu avô Matias (in memoriam), agradeço por me ensinar as contas básicas e por, mesmo sem compreender a minha trajetória acadêmica, sempre me incentivar a continuar estudando.

Agradeço à professora Maria Camila e sou imensamente grata por não ter desistido de mim. Sua dedicação e profissionalismo me inspiram; sempre serei grata pelos ensinamentos e pelo incentivo que tornaram o desenvolvimento deste trabalho possível.

Ao Prof. Humberto Razente, agradeço pela dedicação e pelos ensinamentos.

Agradeço aos meus amigos que estiveram comigo nos bons e maus momentos, apoio que fez e faz toda a diferença. Em especial, ao Matheus e ao Alexandre, pelo acolhimento e parceria. Vocês me fazem ter fé.

Agradeço à CAPES pela bolsa, apoio financeiro essencial para a realização desta pesquisa.

"Elas me olhavam com orgulho, aprovando os livros no meu colo... Elas apostaram sua raiva, lapidada brilhantemente em uma suave armadura de sobrevivência. O espírito daquelas mulheres sentava comigo em todas as aulas a que assistia."

(Golden 1983, 21)

Resumo

As consultas por similaridade são de grande utilidade para recuperação de dados complexos (como dados de natureza multimídia: imagens, vídeos e áudios), para os quais muitas vezes as relações de ordem não são significativas. Em diversos domínios de aplicações, ordenar os resultados com base em cálculos de distância entre elementos de dados torna mais intuitivo o processo de recuperação de dados. A otimização de operações que envolvem cálculos de similaridade é geralmente dada pela indexação dos dados em métodos de acesso especializados, chamados métodos de acesso métricos (*Metric Access Methods* - MAM). Apesar do desenvolvimento ocorrido nas últimas duas décadas, a presença de um grande número de dimensões nos dados degrada o desempenho dos métodos existentes. Nesse contexto, essa dissertação apresenta um novo método de acesso métrico para realização de consultas por similaridade, sendo elas, consultas por abrangência e consultas aos vizinhos mais próximos, por meio da adição de pivôs de referência em estruturas denominadas árvores B+. Os resultados dos experimentos realizados com diferentes conjuntos de dados reais demonstram a eficácia da abordagem adotada no desenvolvimento do método apresentado, denominado *GroupSim+*.

Palavras-chave: Árvores B+, Métodos de Acesso Métrico, Consultas por Similaridade.

Abstract

Similarity queries are beneficial for retrieving complex data (such as multimedia data: images, videos, and audios), for which order relationships are not significant. In many application domains, ordering results based on distance calculations makes the data recovery process more intuitive. The optimization of operations involving similarity calculations are usually given by indexing data on methods, called Metric Access Methods (MAM). Despite the development that occurred over the last two decades, the presence of a large number of dimensions in the data degrades the performance of existing methods. In this context, this dissertation presents a new metric access method for similarity queries, such as range queries and nearest neighbor queries, through adding reference pivots to structures called B + trees. Experimental results performed with different real datasets demonstrate the effectiveness of the presented MAM, called *GroupSim+*.

Keywords: B+-Tree, Metric Access Methods, Similarity Queries.

Lista de ilustrações

Lista de tabelas

Sumário

Capítulo 1

Introdução

Os Sistemas Gerenciadores de Bancos de Dados (SGBDs) foram desenvolvidos para manipular dados numéricos e textuais de forma eficiente. Entretanto, devido a popularização do acesso a tecnologia tornou-se comum a utilização de dados complexos, tais como, dados multimídias (imagens, áudios, vídeos, textos longos), séries temporais, informações geográficas, dados biométricos, sequências de proteínas, dentre outros (POLA, 2010). Esses dados são gerados por meio de inúmeras aplicações e dispositivos e são considerados complexos pois não são formados apenas por um tipo simples de dados, como inteiro, ponto flutuante, cadeia de caracteres, etc (BARIONI, 2006). A complexidade da manipulação e recuperação desses dados tem aumentado devido a quantidade e diversidade dos dados gerados. Nesse contexto, torna-se necessário o desenvolvimento de estruturas especializadas que garantam a eficiência dessas operações.

Os operadores relacionais comumente utilizados nos SGBDs totalizam 6 operadores principais ($=, <>, >, <, >=, <=$) que permitem a comparação de dois objetos, definindo igualdade ou a precedência entre eles. Porém, consultas utilizando operadores relacionais não são utilizadas para dados complexos, pois a maioria dos domínios de dados complexos não possuem uma relação de ordem. Também, não é comum a ocorrência de dois dados exatamente iguais. Assim, para manipular dados considerados complexos é utilizado o conceito de similaridade para comparar objetos de dados, possibilitando assim, operações como, ordenação, indexação e recuperação (POLA, 2010).

As consultas por similaridade retornam objetos pertencentes ao conjunto de dados que, de acordo com um limiar estabelecido, são considerados mais similares ao objeto de consulta. Os principais métodos de consulta por similaridade são: Consulta por Abrangência (*Range Query*) e Consulta aos K-vizinhos mais próximos (*K-Nearest-Neighbor ou K-NN*) (CIACCIA et al., 1997). A otimização dessas consultas é normalmente realizada por meio da utilização de estruturas de indexação conhecidas como Métodos de Acesso Métricos (MAM). Nesses métodos, as comparações por similaridade requerem que os domínios de dados sejam representados em um espaço métrico. Um espaço métrico é definido por um par $\langle \mathbb{S}, d() \rangle$, onde $\mathbb{S}$ é o domínio de dados e $d()$ é uma função de distância que obedece as

propriedades de simetria, não-negatividade e desigualdade triangular. Essa última propriedade é especialmente importante para os MAM uma vez que permite podar regiões do espaço que certamente não contêm objetos que atendem as condições de uma consulta.

Mapeamentos unidimensionais de dados têm sido empregados no desenvolvimento de MAM interessantes. A ideia básica desses métodos consiste em ordenar os objetos de dados considerando as distâncias em relação a um objeto de referência (pivô). Isso permite a utilização de estruturas dinâmicas baseadas na relação de ordem total, como a árvore $B+$, para indexação dos dados. Exemplos de MAM baseados nessa estratégia são: *OmniB-Forest* (TRAINA et al., 2007), *iDistance* (JAGADISH et al., 2005) e *GroupSim* (RAZENTE et al., 2017a). A otimização das consultas por similaridade de modo exato nesses métodos é baseada na propriedade da desigualdade triangular considerando cada pivô, o objeto de consulta e o raio de abrangência dado.

A principal característica do método *OmniB-Forest* é a utilização de uma árvore $B+$ para cada mapeamento unidimensional. O conjunto candidato de uma busca por abrangência nesse método é a intersecção dos anéis formados com base no objeto de consulta, o raio e o conjunto de pivôs. Já o método *iDistance* particiona o conjunto de dados considerando o conjunto de pivôs, armazenando-os em uma mesma árvore $B+$, por meio de uma constante c para separar as partições. O método *GroupSim* emprega uma árvore $B+$ para indexação de um mapeamento unidimensional, entretanto, agrega as distâncias para os demais mapeamentos unidimensionais em cada entrada da estrutura, permitindo a computação de regiões mínimas de poda com um custo inferior aos outros métodos.

1.1 Abordagem do Problema e Contribuições

O desenvolvimento de um método de acesso métrico para indexação e recuperação de objetos por similaridade não é uma tarefa simples pois, o método deve apresentar um bom desempenho para conjuntos com diferentes dimensionalidades, cardinalidades, e distribuições. Um dos principais desafios está relacionado a definição de uma estratégia que melhore a definição da região de busca, diminuindo o custo computacional gasto em comparações desnecessárias.

O trabalho apresentado nessa dissertação visou contribuir para a otimização de consultas por similaridade por meio da exploração de uma nova abordagem baseada na utilização de uma árvore $B+$ na construção de índices dinâmicos. A abordagem adotada na proposta do novo MAM, denominado *GroupSim+*, buscou combinar as vantagens de dois métodos presentes na literatura, sendo elas: o particionamento do espaço e o armazenamento de um vetor contendo a distância do objeto quando comparado a todos os pivôs. A abordagem adotada possibilitou um maior estreitamento das regiões mínimas de poda em relação aos métodos correlatos, que resultou na diminuição do tempo de consulta. Também foram propostos algoritmos eficientes de consulta por abrangência e aos vizinhos mais próximos.

Os experimentos realizados ajudaram a corroborar as hipóteses levantadas e a direcionar a realização de trabalhos futuros.

1.2 Organização da Dissertação

A dissertação está organizada da seguinte forma:

- O Capítulo 2 apresenta a fundamentação teórica necessária para o entendimento do trabalho, tendo como tópicos: a representação dos dados, o espaço métrico, funções de distância, consultas por similaridade e métodos de acesso métrico.
- O Capítulo 3 apresenta os principais métodos de acessos métricos correlatos, *Slim-Tree*, *OmniB-Forest*, *iDistance* e *GroupSim*.
- O Capítulo 4 detalha o método *GroupSim+*. O MAM é descrito e representado por imagens e pseudo-códigos que auxiliam no seu entendimento.
- No Capítulo 5 são apresentados os experimentos e métricas usadas para avaliar os resultados, bem como, as configurações, conjuntos de dados utilizados e uma análise a respeito dos resultados obtidos.
- O Capítulo 6 apresenta as conclusões, descrevendo as principais contribuições do trabalho descrito aqui e as propostas para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Para uma melhor compreensão do trabalho de pesquisa apresentado aqui, os principais conceitos utilizados no processo de indexação de dados complexos por similaridade são apresentados neste capítulo. Os símbolos e definições descritos na Tabela 1 são usados em toda a dissertação.

Tabela 1 – Símbolos e suas definições

Símbolo	**Descrição**
$\mathbb{X}$	Conjunto de todos os objetos do domínio de dados.
$\boldsymbol{d}(x_1,x_2)$	Função de distância ou função de dissimilaridade.
x_q	Objeto de consulta (objeto de referência). $x_q \in \mathbb{X}$
r_q	Raio de referência para consulta.
L_p	Função de distância Minkowski. L_1 é a função de distância Manhattan, L_2 é a função Euclidiana e L_∞ a Chebychev.
$R_q(x_q, r_q)$	Consulta por abrangência utilizando x_q como objeto de referência. e r_q como raio.
$kNN(x_q, k)$	Consulta aos k-vizinhos mais próximos utilizando x_q como objeto. de referência e $\boldsymbol{k}$ a quantidade de objetos a serem retornados.

2.1 Representação dos Dados

Existem diferentes domínios de dados complexos que requerem a realização de consultas por similaridade. Para realizar a comparação por similaridade nesses domínios de dados, muitas vezes é necessário, extrair informações que representem características intrínsecas a esses dados. Por exemplo, no domínio de imagens, podem ser observadas características como, cor, forma e textura. Para o domínio de dados de áudio é comum analisar a frequência e a altura do comprimento da onda. Desse modo, para cada dado, independente do domínio, é criado um vetor denominado **vetor de características** contendo valores numéricos que representam as características extraídas. Nessa abordagem de representação dos dados cada característica corresponde a uma dimensão no espaço

dos dados. Este procedimento é importante, visto que a similaridade entre os dados será calculada por meio desses valores e não utilizando os objetos em si. Essa estratégia de busca é denominada **Recuperação por conteúdo** (SMEULDERS et al., 2000).

Existem dois principais modelos de espaço que podem ser utilizados para representação de objetos de diferentes domínios de dados complexos, são eles: multidimensional e métrico. Uma descrição a respeito de cada um desses modelos é apresentada a seguir:

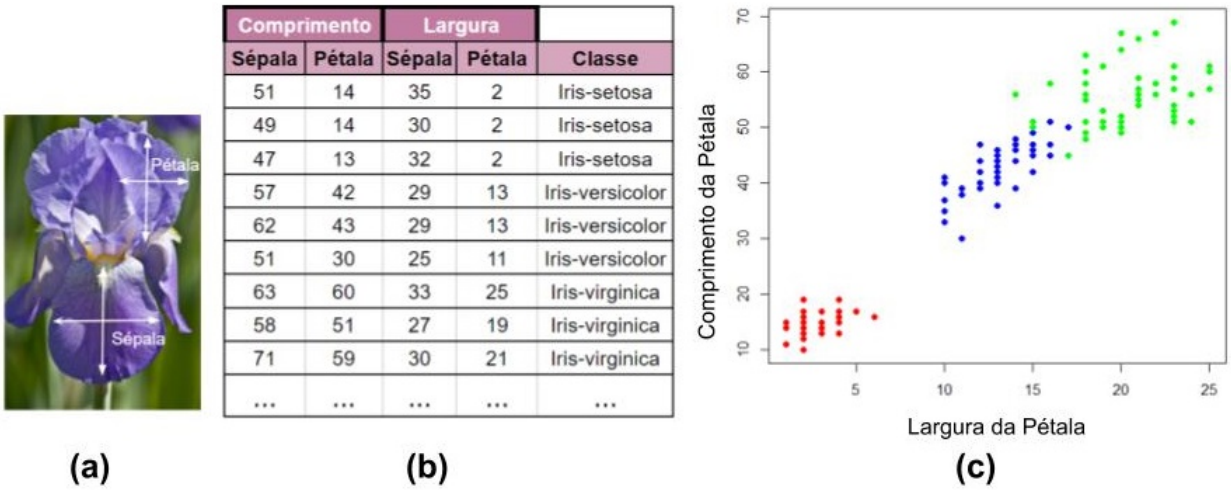
- Modelo multidimensional: Segundo Samet (2006) o modelo multidimensional possui objetos representados por vetores de características de mesmo tamanho que contenham apenas valores numéricos. Cada objeto pode ser representado como um ponto no espaço onde cada dimensão se refere a uma das características presentes no vetor de características. Esse vetor pode ter inúmeras dimensões. Nesse contexto, a escolha de um extrator de características que represente bem esses dados, com o menor número de características, é um desafio. A comparação entre os objetos é feita utilizando os vetores de características para calcular a distância (proximidade espacial) entre eles. Dessa forma é possível mensurar o quão similar os objetos são. A Figura 1 apresenta uma ilustração do conjunto de dados *Iris* (ASUNCION; NEWMAN, 2007). Esse conjunto de dados contém dados numéricos de medições realizadas a respeito de duas características, comprimento e altura, de duas partes da flor, as pétalas e as sépalas. A Figura 1 (c) apresenta uma representação gráfica com duas dimensões, comprimento e altura da pétala, desse mesmo conjunto de dados.

- Modelo métrico: Segundo Chávez et al. (2001) o modelo métrico é mais amplo e engloba o modelo multidimensional. Uma das características deste modelo é a possibilidade dos objetos terem um número variável de dimensões, ou seja, não há um tamanho fixo para os vetores de características. Um dos pontos principais ao utilizar o modelo métrico, é que as comparações entre os objetos são feitas por meio de medidas de dissimilaridade. A Figura 2 ilustra três objetos de um conjunto de imagens, com diferentes quantidades de características extraídas. Nesse exemplo, por meio do cálculo da dissimilaridade, um objeto pode ser considerado mais similar ou menos similar aos demais objetos. Nesse modelo, quanto mais similar os objetos são, mais próximos seus pontos estão no espaço.

2.2 Espaço Métrico

As consultas por similaridade geralmente requerem que os objetos sejam representados em espaços métricos, que englobam os espaços multidimensionais (ZEZULA et al., 2006). Assim sendo, um espaço métrico M pode ser representado por um par $\mathbb{X}$, $\boldsymbol{d}$, sendo $\mathbb{X}$ o domínio dos dados e $\boldsymbol{d}$ uma função de distância que satisfaz as seguintes propriedades para quaisquer x_1,x_2,$x_3 \in \mathbb{X}$:

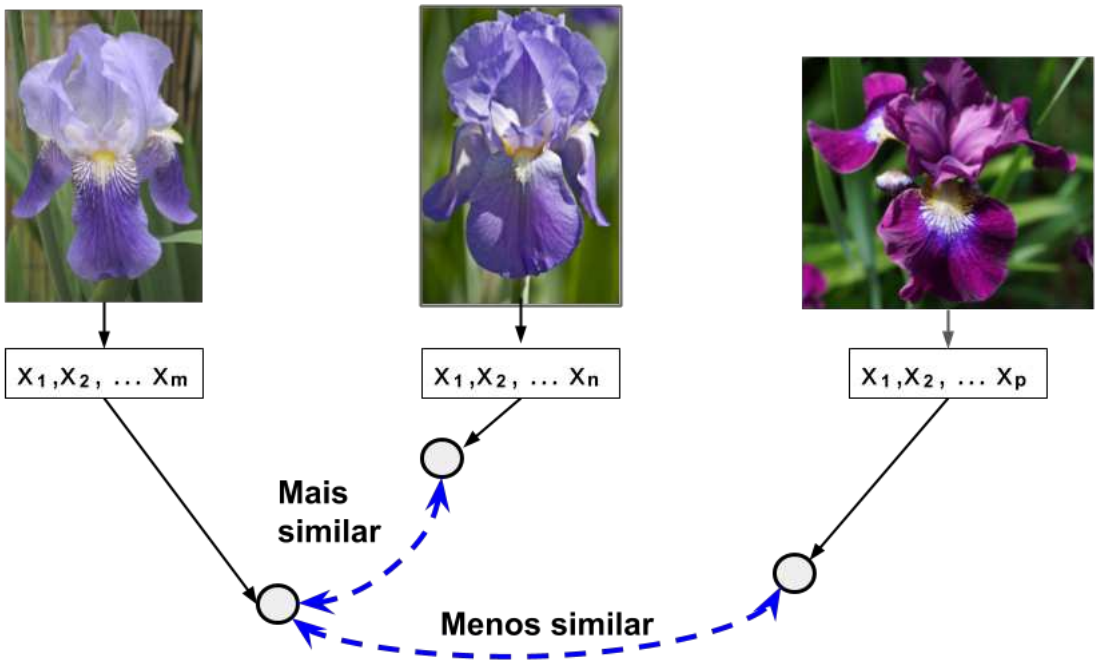
Figura 1 – Conjunto de dados Iris (ASUNCION; NEWMAN, 2007), representado graficamente, considerando duas dimensões.



Comprimento		Largura		
Sépala	Pétala	Sépala	Pétala	Classe
51	14	35	2	Iris-setosa
49	14	30	2	Iris-setosa
47	13	32	2	Iris-setosa
57	42	29	13	Iris-versicolor
62	43	29	13	Iris-versicolor
51	30	25	11	Iris-versicolor
63	60	33	25	Iris-virginica
58	51	27	19	Iris-virginica
71	59	30	21	Iris-virginica
...	...	...	...	...

Fonte: adaptado de (BARIONI et al., 2014).

Figura 2 – Similaridade entre imagens de flores íris, representadas espacialmente de acordo com seus respectivos vetores de características.

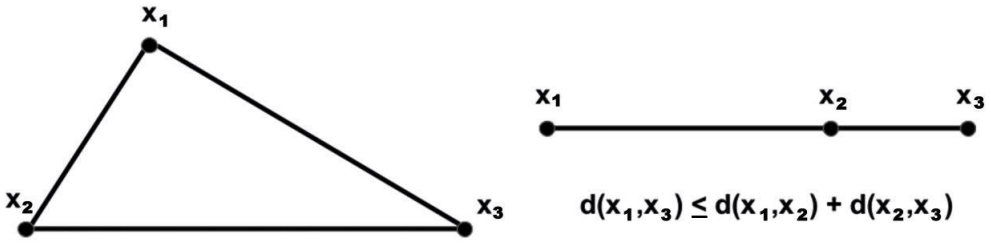


Fonte: Criação da Figura a partir de imagens disponíveis via licença *Creative Commons CC0* (PIXABAY, 2018).

- Identidade: $\boldsymbol{d}(x_1,x_1) = 0$;
- Simetria: $\boldsymbol{d}(x_1,x_2) = \boldsymbol{d}(x_2,x_1)$;
- Não-negatividade: Se $x_1 \neq x_2$, então $\boldsymbol{d}(x_1,x_2) > 0$;
- Desigualdade triangular: $\boldsymbol{d}(x_1,x_3) \leq \boldsymbol{d}(x_1,x_2) + \boldsymbol{d}(x_2,x_3)$

A Figura 3 ilustra essa propriedade, originada na geometria euclidiana, que afirma que, em um triângulo qualquer, o comprimento de um dos lados é sempre inferior à soma dos outros dois lados (SANTANA; QUEIRÓ, 2010). Para os métodos de acessos métricos essa é uma propriedade fundamental, pois assim como nas demais estruturas de indexação é necessário garantir que a estrutura não seja totalmente percorrida na realização de consultas. A propriedade da desigualdade triangular será utilizada para realizar **podas**, ou seja, eliminar sub-árvores nas quais certamente o dado não está. Desse modo, se torna essencial utilizar funções de distância que também atendam a essa propriedade.

Figura 3 – Representação da Desigualdade Triangular



Fonte: adaptado de (POLA, 2010)

2.3 Funções de Distância

Considerando a definição de espaço métrico apresentada na Seção 2.2, é necessária a utilização de uma função de distância para calcular a similaridade entre dois objetos. Dentre as funções de distância disponíveis na literatura científica da área que atendem a propriedade da desigualdade triangular, uma das mais amplamente utilizadas é a Função Minkowski ou L_p. A definição da Função Minkowski é apresentada na Equação 1 (WILSON; MARTINEZ, 1997).

$$d(x_1, x_2) = \sqrt[p]{\sum_{i=1}^{n} |x_1^i - x_2^i|^p} \tag{1}$$

Na Equação 1, $\boldsymbol{p}$ é um parâmetro que recebe um valor inteiro maior ou igual a 1 e $\boldsymbol{n}$ corresponde ao número de dimensões (características). Alguns dos valores possíveis para $\boldsymbol{p}$ são 1, 2 ou ∞, que correspondem as funções L_1 ou Manhattan (veja a Equação 2), L_2 ou Euclidiana (veja a Equação 3) e L_∞ ou Chebychev (veja a Equação 4). Uma visão geral sobre outras funções de distância pode ser obtida em (CHA, 2007) e (WILSON; MARTINEZ, 1997)

$$d(x_1, x_2) = \sum_{i=1}^{n} |x_1^i - x_2^i| \tag{2}$$

$$d(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_1^i - x_2^i)^2} \tag{3}$$

$$d(x_1, x_2) = max_{i=1}^{n} |x_1^i - x_2^i| \tag{4}$$

Considerando que o retorno da função de distância $\boldsymbol{d(}x_1, x_2\boldsymbol{)}$ indica a distância entre x_1 e x_2, quanto menor o valor retornado mais similares esses objetos são. Em alguns cenários, caso os vetores de características estejam em escalas diferentes é necessário realizar a normalização dos dados antes de calcular a distância, para garantir que a variação desses valores ocorram em uma mesma faixa (KASTER, 2012).

Para as aplicações que necessitam realizar processos de análise de dados em espaços com muitas dimensões um desafio é lidar com a maldição da dimensionalidade (SAMET, 2006). Neste contexto, a alta dimensionalidade pode impactar negativamente o desempenho de operações de cálculos de distância entre objetos. Nesse caso pode ser necessária a utilização de técnicas de redução de dimensionalidade, por exemplo a seleção de características (FACELI et al., 2011).

2.4 Consultas por Similaridade

Conforme descrito na Seção 2.1 as consultas por similaridade são comumente utilizadas para recuperação de dados complexos por conteúdo. Utilizando vetores de características e uma função de distância (veja a Seção 2.3) a distância entre dois objetos pode ser calculada. É importante observar que, diferente das consultas utilizando operadores relacionais, as consultas por similaridade retornam um conjunto contendo os objetos mais similares ao objeto de referência informado na consulta (FILHO et al., 2001).

Na literatura científica da área é possível encontrar a definição de diferentes consultas por similaridade (AGRAWAL et al., 1993) e (SANTOS, 2012). Dentre essas, as mais utilizadas são a consulta por Abrangência (*Range Query*) e a consulta aos k-Vizinhos

mais Próximos (*k-Nearest Neighbor Query*). Essas consultas são descritas nas Seções 2.4.1 e 2.4.2, respectivamente.

2.4.1 Consulta por Abrangência

Uma consulta por abrangência $R_q(x_q, r_q)$ retorna os objetos do conjunto de dados que sejam dissimilares do objeto de referência x_q até no máximo um limiar definido pelo raio r_q. Formalmente, dado um objeto de consulta $x_q \in \mathbb{X}$, um conjunto de objetos $X \subseteq \mathbb{X}$, uma função de distância, *d()*, e uma distância máxima de busca r_q, o subconjunto reposta de uma consulta por abrangência pode ser definido como apresentado na Equação 5 (CHÁVEZ et al., 2001).

$$R_q(x_q, r_q) = \{x_i | x_i \in X : d(x_i, x_q) \leq r_q\} \tag{5}$$

A Figura 4(a) ilustra uma consulta por abrangência em um conjunto de objetos, representados em duas dimensões, utilizando a função de distância Euclidiana. O conjunto de objetos retornado da consulta por abrangência pode conter inúmeros objetos ou nenhum. Como um exemplo prático, considerando o domínio de dados ilustrado na Figura 1, uma busca por abrangência necessita de um vetor de características que represente uma imagem de uma flor do conjunto Íris como objeto de referência e a definição de um raio para encontrar as flores mais semelhantes ao objeto de consulta, que estejam dentro do raio definido.

2.4.2 Consulta aos k-Vizinhos mais Próximos

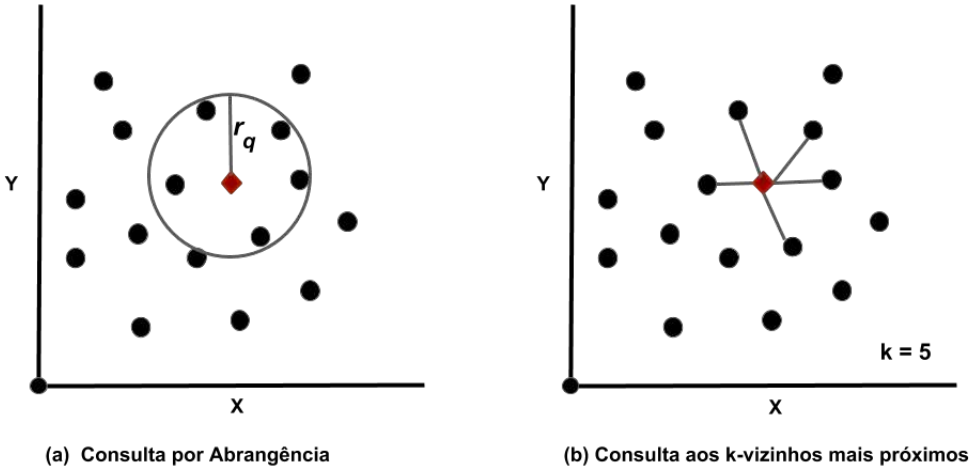
Uma consulta aos k-vizinhos mais próximos $kNN(x_q, k)$ retorna os $\boldsymbol{k}$ objetos do conjunto de dados mais similares ao objeto de referência x_q. Formalmente, dado um objeto de consulta $x_q \in \mathbb{X}$, um conjunto de objetos $X \subseteq \mathbb{X}$, uma função de distância *d()*, e um número inteiro $\boldsymbol{k}$ de objetos a serem retornados, o subconjunto resposta $X' \subseteq X$ retornado por uma consulta aos k-vizinhos mais próximos pode ser definido como apresentado na Equação 6 (CHÁVEZ et al., 2001).

$$\{x_j | x_j \in X', \forall x_i \in X - X' : d(x_j, x_q) \leq d(x_i, x_q), |X'| = k\} \tag{6}$$

A Figura 4(b) representa uma consulta aos k-vizinhos mais próximos em um conjunto de objetos, representados em duas dimensões, utilizando a função de distância Euclidiana. O conjunto de objetos retornados da consulta kNN contém no máximo k objetos. Considerando o exemplo ilustrado na Figura 2, é possível usar a consulta kNN para obter as k flores mais próximas a imagem de uma flor usada como referência para a busca, utilizando vetores de características que representam características intrínsecas das imagens

para calcular a distância entre eles. É importante observar, que essa consulta retorna os objetos ordenados de acordo com a proximidade do objeto de referência. O kNN é um dos algoritmos de busca mais utilizados, devido a adaptações para aprender *rankings*, e é comumente aplicado a problemas de meta-aprendizagem (SOUZA et al., 2001) (BRAZDIL et al., 2008).

Figura 4 – Exemplos de consultas por similaridade: (a) consulta por abrangência considerando um raio r_q; (b) consulta aos k-vizinhos mais próximos tendo $k = 5$.



2.5 Seleção de pivôs

A escolha do método de seleção de pivôs é fundamental. Na literatura, podem ser encontrados diversos algoritmos para essa finalidade, como, *Hull of Foci* (HF), *Maximum of Minimum Distances* (MMD), *Maximum of Sum of Distances* (MSD) e *Sparce Spatial Selection* (SSS), dentre outros. O algoritmo MSD foi escolhido devido às considerações apresentadas por (RAZENTE et al., 2017b) e (SOCORRO et al., 2011).

O método utilizado nas experimentações é descrito na seção seguinte. Essa técnica adota uma abordagem incremental para a seleção de pivôs, priorizando objetos que estejam o mais distante possível dos já selecionados. O processo inicia-se com a escolha aleatória de um objeto, seguida da adição incremental de novos pivôs com base em uma agregação de distâncias.

Cada novo pivô é selecionado de forma a maximizar a distância em relação ao pivô anterior. O primeiro pivô é escolhido aleatoriamente, e os demais são selecionados sucessivamente, garantindo que cada novo pivô esteja localizado na maior distância possível em relação ao último escolhido.

Uma das principais vantagens desse algoritmo é sua capacidade de preservar os pivôs previamente selecionados ao aumentar a quantidade de pivôs para experimentação, acres-

centando novos de acordo com o método estabelecido. Isso permite uma comparação mais precisa do desempenho nas diferentes configurações testadas.

2.6 Considerações Finais

Este capítulo apresentou os principais conceitos relacionados a representação e consultas de dados complexos em um espaço métrico. Também foram apresentados os principais métodos de acesso métricos da literatura que contribuem para a estruturação do *Group-Sim+*, método proposto que é descrito no próximo capítulo.

Capítulo 3

Trabalhos Correlatos

3.1 Métodos de Acesso Métrico

Os Métodos de Acesso Métricos (*Metric Access Methods* - MAM) são estruturas de indexação que utilizam apenas funções de distância para organizar os dados. Considerando que as consultas por similaridade utilizadas para recuperar os dados complexos são realizadas em um espaço métrico é adequado utilizar MAM para o processo de indexação desses dados (BARIONI, 2006).

Considerando um conjunto de dados, a estratégia proposta pelos MAM consiste em particionar o espaço de dados em várias regiões. Para cada uma dessas regiões é definido um (ou mais) objeto(s) representante(s) que agrupa(m) outros objetos de acordo com critérios de similaridade. Alguns exemplos de MAM são M-Tree (CIACCIA et al., 1997), *Slim-tree* (TRAINA et al., 2000), MM-tree (POLA et al., 2007), *OmniB-Forest* (TRAINA et al., 2007), *iDistance* (JAGADISH et al., 2005), *GroupSim* (RAZENTE et al., 2017b), dentre outros.

Dentre os MAM listados, *OmniB-Forest*, *iDistance* e *GroupSim* utilizam mapeamentos unidimensionais e árvores $B+$ na estratégia de indexação, semelhante a estrutura proposta no trabalho descrito nesta dissertação. Mapeamentos unidimensionais de dados têm sido empregados no desenvolvimento de MAM interessantes. A ideia básica desses métodos consiste em ordenar os objetos de dados considerando as distâncias em relação a um objeto de referência (pivô). Nas Seções a seguir serão apresentados os MAM mais significativos para o desenvolvimento deste trabalho, sendo eles, *Slim-tree*, *OmniB-Forest*, *iDistance* e *GroupSim*.

3.1.1 *Slim-tree*

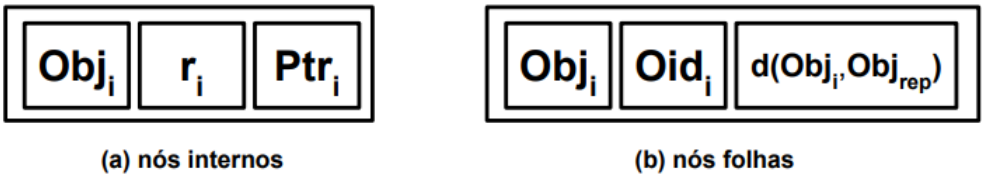
A *Slim-Tree* (TRAINA et al., 2000) é um método de acesso métrico eficiente que pode ser usado para o armazenamento e a recuperação dados por conteúdo. A estrutura *Slim-Tree* é dinâmica, balanceada e tem estrutura *bottom-up*, ou seja, cresce a partir das

folhas em direção a raiz. Ela possui dois tipos de nós: nós folhas (ou nós externos), onde se encontram todos os objetos; e nós índices, que são utilizados para definir regiões do espaço de dados. Devido ao tamanho das páginas ser fixo, cada tipo de nó possui uma quantidade máxima de objetos pré-definida.

Semelhante a outras estruturas métricas, a propriedade da desigualdade triangular é utilizada para realizar podas, ou seja, eliminar a busca do objeto em uma sub-árvore onde certamente ele não está. Assim, é possível limitar a região do espaço onde serão realizadas as buscas, diminuindo a quantidade de comparações durante a realização de uma consulta por similaridade.

A *Slim-Tree* é construída a partir da seleção de alguns objetos, conhecidos como **objetos representantes**, ou **pivôs**, que definem o centro de uma região do espaço. A partir de um objeto representante é estabelecido um raio definindo uma região de cobertura(também chamado de bola), sendo que apenas os objetos pertencentes a essa região podem ser armazenados nesse ramo (ou sub-árvore). Sendo assim, os nós internos da árvore possuem tuplas que contém as seguintes informações: (1) objeto representante; (2) raio de cobertura que engloba as suas sub-árvores; e (3) nó raiz das sub-árvores. Os nós folhas possuem: (1) objeto que o nó armazena; (2) identificador do nó; e (3) a distância desse nó em relação ao objeto representante. A Figura 5 representa a estrutura dos dois tipos de nós, internos e externos. A parte identificada por (a) representa a estrutura dos nós internos a árvore, sendo Obj_i o objeto representante, r_i o raio de cobertura que engloba as suas sub-árvores e Ptr_i um ponteiro que indica o nó raiz de sua sub-árvore. A parte (b) representa a estrutura dos nós folhas, sendo Obj_i o objeto representante, Oid_i um identificador do nó e $d(Obj_i, Obj_{rep})$ a distância desse objeto em relação ao objeto de referência.

Figura 5 – Estrutura dos dois tipos de nós, sendo (a) os nós internos e (b) os externos.



Devido a estrutura dinâmica da árvore, novos objetos podem ser inseridos a qualquer momento, não sendo necessário que todos os objetos estejam disponíveis no momento da sua construção. Para inserir um objeto, a árvore é percorrida em busca do nó folha onde o raio de cobertura abrange o objeto a ser inserido. Se mais de um nó for apto a receber o novo objeto o algoritmo *ChooseSubtree* será executado para definir qual a sub-árvore mais apropriada para a inserção. A *Slim-Tree* possui três opções para o algoritmo *ChooseSubtree*: (1) aleatório (*random*) que seleciona aleatoriamente um dos nós; (2) distância mínima (*MinDist*) que escolhe o nó que possui a menor distância entre seu representante

e novo objeto; e (3) ocupação mínima (*MinOccup*) que seleciona o nó que tenha o menor número de objetos armazenados. O *MinOccup* é a opção padrão da Slim-Tree.

A escolha do algoritmo *ChooseSubtree* tem influência na estrutura da árvore resultante. Por exemplo, a opção *MinOccup* resulta em árvores onde os nós tenham uma maior quantidade de objetos armazenados, diminuindo a quantidade de acessos a disco. Porém possui um maior grau de **sobreposição**, termo que define quando mais de um nó abrange uma mesma região do espaço. A opção *MinDist* tende a gerar árvores com menor taxa de ocupação dos nós, consequentemente mais altas e com menor grau de sobreposição.

Quando ocorre *overflow*, ou seja, ao inserir um objeto em um nó que atingiu sua capacidade máxima, é necessário redistribuir os objetos entre dois nós, o nó atual e um novo nó que será criado. Essa redistribuição é denominada *split*. Quando ocorre um *split* um objeto de cada nó é selecionado como objeto representante e é promovido para o nível superior (nó pai). A *Slim-tree* possui três algoritmos de *split*, sendo eles:

1. **Aleatório (*random*):** seleciona de forma aleatória os objetos representantes para os nós, e os demais objetos são distribuídos entre eles considerando a menor distância entre o novo objeto representante;

2. **Mínimo dos maiores raios (*MinMax*):** todas as combinações de pares de possíveis representantes são consideradas na escolha. Para cada par, os demais objetos são distribuídos entre os nós de acordo com a menor distância entre o novo representante. O par de representantes que tiver o menor raio de cobertura será escolhido;

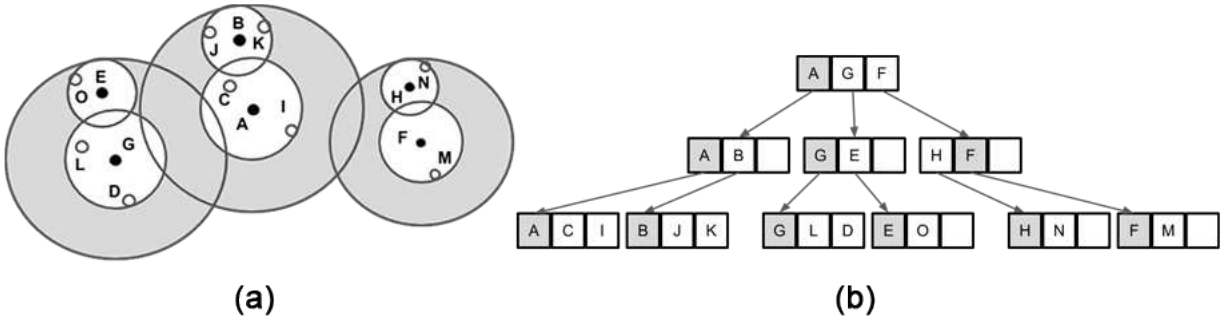
3. ***Minimal Spanning Tree* (MST):** essa estratégia baseia-se na construção de uma MST, uma árvore geradora de custo mínimo (KRUSKAL, 1956). Após a construção dessa estrutura que abrange grafos não-dirigidos considerando custos nas arestas, a aresta mais longa é removida e o objeto mais central de cada agrupamento é eleito como representante. Esse algoritmo constrói árvores praticamente equivalentes às construídas utilizando o algoritmo *MinMax*, porém com um menor custo de processamento e em menor tempo. Portanto, essa é a opção padrão para o *split* na *Slim-tree*.

É importante observar que as estratégias aleatórias (*random*) tanto para a escolha de sub-árvore (*ChooseSubTree*) quanto para a distribuição (*split*) foram propostas apenas para servirem de base de comparação na avaliação das outras estratégias propostas *MinMax* e MST (*ChooseSubTree*) e *MinDist* e *MinOccup* (*split*). Essa avaliação levou em consideração o ganho de desempenho das estruturas geradas em relação ao processamento gasto para execução dessas estratégias.

A Figura 6 representa uma *Slim-Tree* com 15 objetos, sendo (a) sua estrutura lógica e (b) a organização dos objetos na árvore. Na estrutura lógica o espaço de cor cinza representa a região mapeada pelo objetos representantes, na árvore os objetos representantes

são destacados com a mesma cor. As regiões formadas por círculos brancos representam a região de cobertura dos nós folhas. É possível observar que há sobreposição, ou seja, existem regiões que pertencem a área de cobertura de mais de um nó.

Figura 6 – A *Slim-Tree* é representada, considerando 15 objetos, sendo (a) sua estrutura lógica e (b) a organização dos objetos na árvore



Fonte: adaptado de (BARIONI, 2006)

A *Slim-Tree* foi desenvolvida tendo como objetivo a redução da sobreposição entre duas regiões, visto que, a sobreposição aumenta a quantidade de cálculos pois mais sub-árvores serão visitadas para responder uma consulta por similaridade. Além disso, a *Slim-Tree* conta com mecanismos de verificação da taxa de sobreposição, por meio de um fator de sobreposição chamado *Fat-Factor*. Esse fator mensura o grau de sobreposição entre os nós e é utilizado pelo algoritmo de reorganização da árvore *Slim-Down*. Experimentos apresentados em (TRAINA et al., 2000) corroboram a afirmação de que a *Slim-Tree* é um método de acesso métrico eficiente, que possibilita a realização de consultas por similaridade diminuindo tanto a quantidade de cálculos de distâncias quanto a quantidade de acessos a disco.

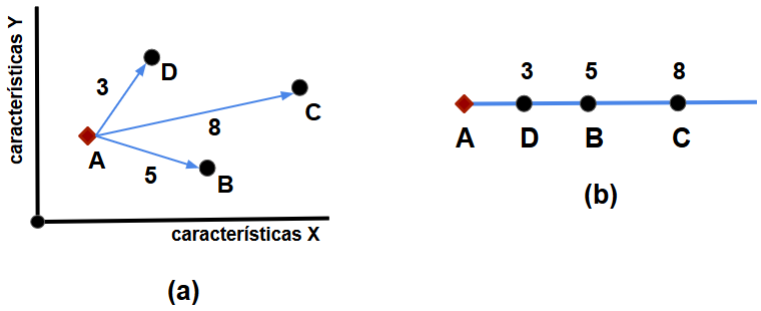
3.1.2 *OmniB-Forest*

A estratégia adotada pela *OmniB-Forest* (TRAINA et al., 2007) foi definida tendo como objetivo aumentar a capacidade de poda de nós de uma árvore métrica. Sua estrutura utiliza uma árvore *B+* (COMER, 1979), que possui construção *bottom-up*, e mantém todos os objetos indexados nos nós folhas, permitindo que eles possam ser percorridos de forma sequencial.

A *OmniB-Forest* faz parte da *Omni-Family*, uma família de estruturas de indexação criada com o objetivo de reduzir o número de comparações durante o processo de indexação dos dados. A ideia principal dessa família de estruturas é definir focos globais denominados *Foci* ou pivôs e por meio do cálculo da distância desses pivôs em relação aos demais objetos tornar possível estabelecer estruturas unidimensionais. Para uma melhor compreensão a Figura 7 ilustra objetos em um espaço métrico de duas dimensões, onde um dos objetos

é eleito pivô, ou seja, se torna um ponto de referência para o mapeamento dos demais objetos pertencentes a base de dados em uma estrutura unidimensional.

Figura 7 – Mapeamento dos dados métricos de duas dimensões em uma estrutura unidimensional.



Fonte: adaptado de (RAZENTE et al., 2017b)

O mapeamento de dados complexos em uma estrutura unidimensional torna viável estabelecer uma relação de ordem entre os objetos, possibilitando a indexação e recuperação desses dados. Para que a Omni-Forest realize esse mapeamento, as distâncias dos objetos em relação aos pivôs são armazenadas, bem como um identificador do objeto. A indexação de um objeto é realizada por meio da inclusão do objeto em um arquivo de acesso aleatório (*Object Random Access File* - ORAF) e a inclusão da distância para cada pivô <distância, $offset$> na árvore B+ correspondente. Para delimitar a região de busca utilizada no processo de indexação e recuperação é utilizada a propriedade da desigualdade triangular, um objeto de referência e um raio definidos no momento da busca.

Os objetos que satisfazem essas propriedades em relação a todos os pivôs fazem parte do conjunto dos mais similares. Para determinar quais objetos satisfazem essas propriedades em relação a todos os pivôs é utilizado um vetor de mesmo tamanho do conjunto de objetos, onde cada posição representa um dos objetos e terá o seu valor incrementado sempre que este objeto fizer parte do conjunto dos mais similares de algum dos pivôs. Em alguns casos, nos quais a base de dados contém uma grande quantidade de objetos, pode-se utilizar uma árvore binária balanceada, onde é utilizado o identificador do objeto como chave e o contador que será incrementado. Nos dois casos, os objetos que farão parte do conjunto de resposta, ou seja, os mais similares, são os que possuem contagem igual ao número de pivôs (focos globais).

Para realizar uma busca por abrangência, ocorre uma busca por profundidade para encontrar o objeto mais similar ao objeto buscado. Após encontrar esse objeto, pertencente a um nó folha, a estrutura poderá ser percorrida de forma sequencial até encontrar um objeto que não faça mais parte do conjunto de resposta, ou seja, ultrapassa o raio informado.

3.1.3 *iDistance*

O método *iDistance* (JAGADISH et al., 2005) define uma estratégia para mapear objetos multidimensionais em uma estrutura unidimensional, possibilitando o uso de uma árvore B+ para realizar a indexação e recuperação dos objetos. O objetivo principal dessa estrutura é otimizar as consultas kNN, e contém 3 etapas listadas abaixo:

- Definição das partições: Nesta etapa o conjunto de dados é dividido em n partições.
- Escolha dos pivôs: Para cada partição é definido um pivô, ou seja, um objeto que será utilizado como referência para a partição.
- Mapeamento de objetos: Para calcular a chave de indexação essencial para o processo de mapeamento e indexação do objeto, a Equação 7 é utilizada. Nessa equação uma constante c suficientemente grande, obtida por meio de um cálculo estimado da maior distância entre pares de objetos do conjunto, é utilizada para que não haja sobreposição entre os intervalos de diferentes partições. O inicio de cada partição pode ser calculado utilizando a fórmula $c_i = i$ x c, sendo i o índice do pivô que a partição irá representar.

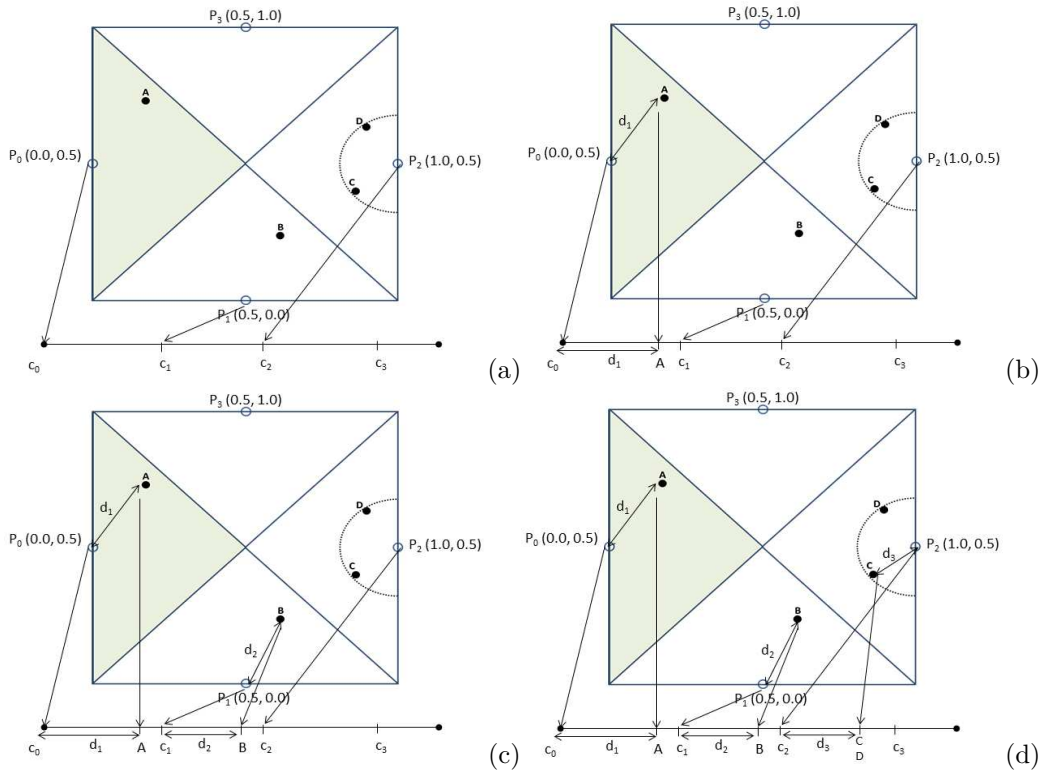
$$y = i\ x\ c + d(P_i, O) \tag{7}$$

Considerando um cenário de duas dimensões, após a definição das partições os objetos A, B, C e D são inseridos, a Figura 8 ilustra o procedimento de mapeamento unidimensional. Na Figura 8 (a), os pivôs P_0, P_1, P_2 têm a chave de indexação calculada conforme a terceira etapa descrita acima. Na Figura 8 (b) o objeto A é mapeado em relação ao pivô mais próximo, de acordo com a Equação 7, do mesmo modo, os objetos B, C e D são mapeados como mostra a Figura 8 (c) e (d).

Para realização da consulta kNN é utilizada uma estratégia semelhante a *OmniB*. Na busca é considerado um raio, iniciando com o valor 1 que será incrementado quando um objeto pertencente ao conjunto de respostas é encontrado. A busca é finalizada quando o valor do raio for igual ao k informado no momento da busca. Em essência, a estratégia é começar pesquisando uma pequena esfera, e de forma incremental, ampliar o espaço de busca até que todos os k-vizinhos mais próximos sejam encontrados.

Os Algoritmos 1, 2 e 3, apresentam a consulta aos k-vizinhos mais próximos no *iDistance*. Como entrada, no Algoritmo 1 temos o objeto de consulta q, o valor de incremento para o raio Δr e o valor máximo possível para o r representado por max_r. Na linha 2 o raio é iniciado com o valor zero, e na linha 3 a variável $stopflag$ é iniciada com $false$, essa variável será utilizada para finalizar a busca. Na linha 4 são inicializados os objetos *lp[], rp[], flag[]* que serão usados na função de consulta *SearchO* detalhada futuramente

Figura 8 – Mapeamento de um conjunto de objetos de duas dimensões utilizando o método *iDistance*



Fonte: adaptado de (RAZENTE et al., 2017b)

no momento da sua execução. Na linha 5 um laço de repetição é inicializado e só irá finalizar quando a variável *stopflag* estiver com o valor *true*. Na linha 6 o r é incrimentado, recebendo na sua primeira execução o valor do Δr. Na linha 7 a função *SearchO* é inicializada com os valores de q e r passados como parâmetro.

Algoritmo 1: Consulta aos vizinhos mais próximos no *iDistance*

Entrada: Objeto de consulta q, um valor de incremento para o raio Δr e o valor máximo possível para o r representado por max_r
Saída: Conjunto resposta δ

```
1 início
2     r = 0;
3     stopflag = false;
4     initialize lp[], rp[], oflag[];
5     enquanto stopflag == false faça
6         r = r + Δr;
7         SearchO(q, r);
8 retorna iDistanceKNN
```

O Algoritmo 2 descreve a função *SearchO* que realiza uma busca em um espaço particionado utilizando o método iDistance, combinando as buscas *SearchInward* e *SearchOutward* considerando o ponto de consulta q e dentro de um raio r, que são os parâmetros de entrada da função. Na linha 2, o ponto $Pfurthest$, que é o mais distante de q dentro do conjunto de respostas S. Na linha 3 verifica-se a distância de $Pfurthest$ e q é

menor que r e se S já contém K vizinhos mais próximos. Se ambas as condições forem verdadeiras, significa que a consulta chega ao fim, atualizando a variável $stopflag$ para $true$ finalizando o laço de repetição.

A linha 5 do Algoritmo 2 itera sobre todas as $mpartições$ do espaço, onde o O_i representa cada partição do espaço. Na linha 6, a distância de q em relação a O_i é calculada. Na linha 7, se O_i ainda não foi pesquisado, ou seja $oflag[i] == false$, então irá ocorrer a verificação. Se a esfera centrada em O_i com raio $dist_maxi$ contém q, significa que q está dentro da partição. A linha 9 marca a partição O_i como pesquisada alterando a $oflag[i]$ para $true$. Na linha 10 encontra-se o nó folha correspondente ao valor de busca usando a estrutura da árvore B+. Na linha 11 uma consulta é realizada dentro do nó folha localizado, representado pela função $SearchInward$ detalhada futuramente no Algoritmo 3. Na linha seguinte, a 12, a busca é realizada fora do nó folha localizado com a função *SearchOutward*. O bloco de instruções da linha 13 até a 16, é executado se a esfera centrada em O_i intersecta a esfera de pesquisa de q com raio r, diferente da condição da linha 8 que verificava se a esfera de pesquisa estava inserida dentro da partição. Na linha 14 a partição atual é marcada como pesquisada. A linha 15 localiza o nó folha baseado no raio máximo da partição. E na linha 16 a função *SearchInward* é executada conferindo os registros internos ao nó folha localizado.

O bloco de código da linha 17 até a linha 21 executa caso a condição da linha 7 seja falsa, ou seja, se a partição O_i já foi pesquisada. Se O_i não contém q nem intersecciona a esfera de pesquisa, executa a busca expandindo os nós irmãos. Na linha 18, se lp[i] não for nulo, significa que há um nó à esquerda para explorar, por isso, $lp[i]$ recebe o resultado da consulta *SearchInward* conforme representado na linha 19. A linha 20 valida se $rp[i]$ não é nulo, nessa condição significa que há um nó à direita para explorar. Desse modo, na linha 21, $rp[i]$ recebe a consulta *SearchOutward*, realizando uma nova consulta para fora a partir do nó à direita.

Os algoritmos *SearchInward* e *SearchOutward* são semelhantes, portanto apenas o primeiro será descrito detalhadamente no Algoritmo 3. Dado um nó folha, o *SearchInward* examina as entradas do nó à esquerda para determinar se o objeto faz parte do conjunto de resposta. A linha 1 do Algoritmo 3 inicia uma iteração sobre cada entrada e presente no nó folha (*node*). Na linha 2, verifica-se se o conjunto de respostas atuais S já atingiu a capacidade máxima K de objetos. Caso essa condição seja verdadeira, a linha 3 identifica o objeto mais distante de q dentro do conjunto S, denotado como $Pfurthest$. A linha 4 realiza uma verificação para constatar se a distância entre a entrada atual e e a consulta q é menor que a distância de $Pfurthest$ até q; em caso afirmativo, a linha 5 remove $Pfurthest$ de S e a linha 6 insere a nova entrada e no conjunto de respostas. O bloco correspondente ao senão, compreendido entre as linhas 7 e 8, é executado caso o conjunto S ainda não esteja completo, realizando diretamente a união da entrada e ao conjunto de resultados. Após o encerramento do laço, a linha 9 verifica se a chave da primeira entrada

Algoritmo 2: *SearchO*

```
   Entrada: Objeto de consulta q, e o raio r
   Saída: Conjunto resposta δ
 1 início
 2     Pfurthest = furthest(S, q);
 3     se dist(Pfurthest, q) < r e |S| == K então
 4         stopflag = true;
 5     para i = 0 até m − 1 faça
 6         dis = dist(O_i, q);
 7         se ! oflag[i] então
 8             se sphere(O_i, dist_max_i) contém q então
 9                 oflag[i] = true;
10                 lnode = LocatedLeaf(arvoreB, i ∗ c + dis);
11                 lp[i] = SearchInward(lnode, i ∗ c + dis − r);
12                 rp[i] = SearchOutward(lnode, i ∗ c + dis + r);
13             senão se sphere(O_i, dist_max_i) intersecção sphere(q, r) então
14                 oflag[i] = true;
15                 lnode = LocatedLeaf(arvoreB, dist_max_i);
16                 lp[i] = SearchInward(lnode, i ∗ c + dis − r);
17         senão
18             se lp not nil então
19                 lp[i] = SearchInward(lp[i] ← leftnode , i ∗ c + dis − r);
20             se rp not nil então
21                 rp[i] = SearchOutward(rp[i] ← rightnode , i ∗ c + dis + r);
22 retorna Conjunto resposta δ
```

do nó ($e_1.key$) é maior que o valor de barreira calculado em $ivalue$. Se for o caso, a linha 10 realiza uma chamada recursiva da função *SearchInward*, deslocando a busca para o nó irmão à esquerda ($node \leftarrow leftnode$) e atualizando o parâmetro de distância limite. Na linha 11, testa-se se o fim da partição corrente foi alcançado e, caso positivo, a linha 12 define o valor de *node* como nulo (nil). Por fim, a linha 13 encerra o algoritmo retornando o nó modificado ou finalizado.

Algoritmo 3: *SearchInward*

Entrada: Nó folha (*node*), distância $d(O_i, q)$ somada de $i * c$ menos o raio atual ($ivalue$)
Saída: Nó folha *node*

```
1  início
2      para cada entrada e em node(e = e_j, j = 1, 2, ..., Número de entradas) faça
3          se |S| == K então
4              Pfurthest = furthest(S, q);
5              se dist(e, q) < dist(Pfurthest, q) então
6                  S = S − Pfurthest;
7                  S ∪ e;
8          senão
9              S = S ∪ e;
10     se e_1.key > ivalue então
11         node = SearchInward(node ← leftnode, i * c + dis − r);
12     se O final da partição é encontrado então
13         node = nil;
14     retorna node
```

3.1.4 *GroupSim*

O método de acesso métrico *GroupSim* (RAZENTE et al., 2017b) apresenta uma abordagem baseada no mapeamento, indexação e recuperação de objetos de alta dimensionalidade utilizando uma única árvore B+. Inicialmente, o *GroupSim* realiza o mapeamento unidimensional, para isso, a quantidade de pivôs escolhidos é definida e posteriormente é executado um algoritmo de seleção de pivôs. Após a realização do mapeamento são gerados vetores unidimensionais os quais são indexados em uma árvore B+.

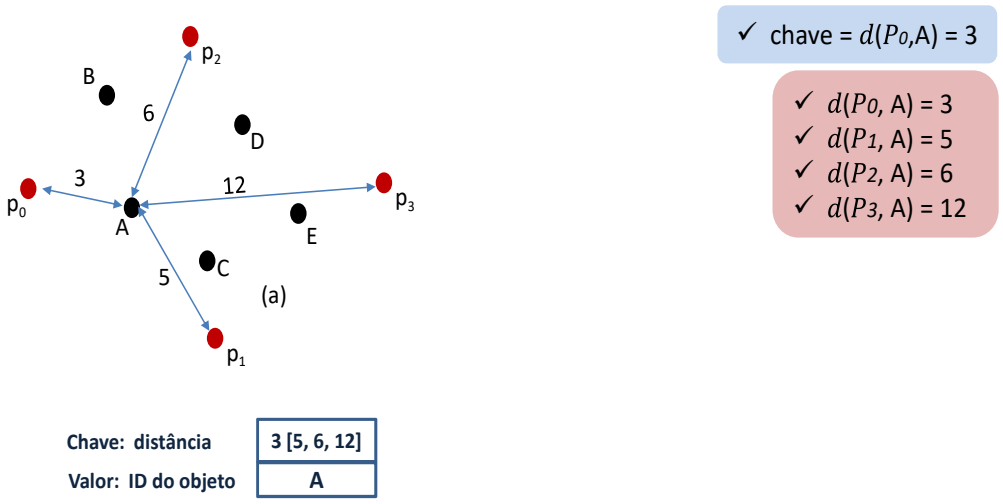
Na estratégia de indexação, após a escolha dos pivôs, para cada objeto pertencente a base de dados será calculada a distância do objeto em relação aos pivôs. Portanto, cada objeto terá uma página correspondente a um nó na árvore B+, contendo as seguintes informações: (1) a distância do objeto em relação ao primeiro pivô que será utilizada como chave para indexação deste objeto na árvore; (2) um ID que referencia o objeto real; e (3) um vetor contendo as distâncias desse objeto em relação aos demais pivôs.

A Figura 11 ilustra a indexação do objeto A, em relação ao pivô mais próximo, o p_0 e posteriormente calcula e armazena a distância em relação aos outros pivôs.

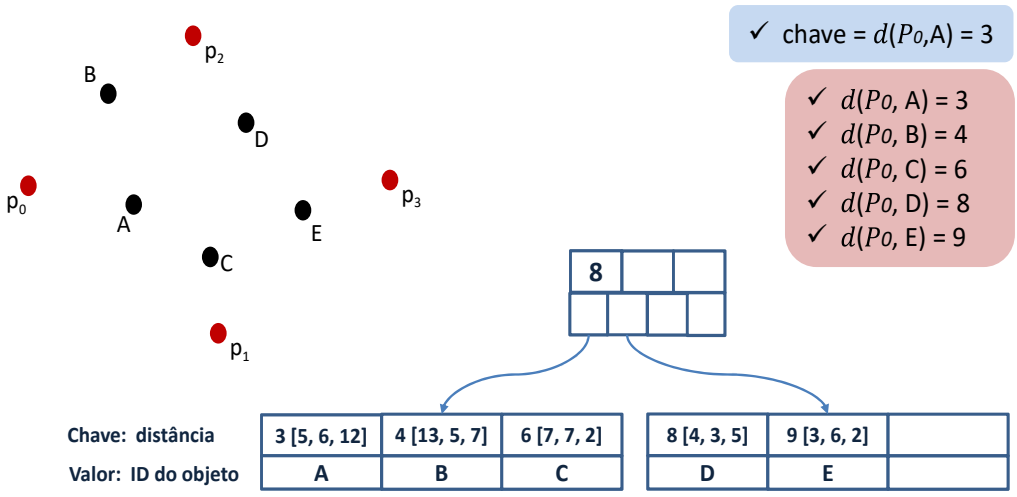
Uma das principais vantagens do *GroupSim* em relação ao *iDistance* e *OmniB-Forest* é o fato de cada página da árvore conter as distâncias do objeto que ela representa em relação a todos os pivôs, sendo necessário realizar esse cálculo apenas um vez. Essa estratégia permite diminuir a quantidade de acessos ao disco, ou seja, é possível determinar se o objeto está na intersecção entre as regiões definidas pelo raio da consulta e pela desigualdade triangular em relação a cada pivô, sem ter que esperar a realização das buscas em outras sub-árvores e sem ter que manter um grande vetor ou uma árvore binária em memória.

Juntamente ao método de acesso métrico foram propostos dois algoritmos para in-

Figura 9 – Exemplo da construção da árvore B+ após mapeamento de objetos bidimensionais *GroupSim+*



(a) Indexação do objeto A. p_0.



(b) Árvore construída após indexação dos objetos A, B, C, D e E.

dexação e recuperação de dados complexos baseados na estrutura do *GroupSim*. Estes algoritmos foram inspirados nas consultas por similaridade apresentadas na Seção 2.4, a consulta por abrangência e aos k-vizinhos mais próximos.

O Algoritmo 4 de *Indexação GroupSim* detalha o procedimento para a inserção de novos elementos na estrutura de índice. Dado um conjunto de objetos S e um conjunto de pivôs P, o algoritmo itera sobre os elementos processando suas respectivas distâncias. A linha 1 do algoritmo inicia uma avaliação para cada elemento do conjunto de objetos, em que o índice i varia de 0 até o tamanho total da coleção ($S.Tamanho$). Na linha 2, a distância entre o objeto atual S_i e o primeiro pivô P_0 é calculada e atribuída à variável *dist*. Na linha 3, essa mesma métrica de distância entre S_i e P_0 é alocada na última posição do vetor de distâncias, representada pelo índice $quantidade\_pivos - 1$ na variável

vet_dist. A linha 4 inicia um laço de repetição interno, no qual o índice j é incrementado de 1 até o valor equivalente à quantidade total de pivôs. Na linha 5, a posição j do vetor *vet_dist* é preenchida com o cálculo da distância (neste pseudocódigo, referenciado como a distância entre S_i e P_0). O bloco das linhas 6, 7 e 8 é responsável por associar os dados processados à estrutura *GroupSim*. Na linha 6, o valor de *dist* é incorporado à estrutura; na linha 7, o identificador exclusivo do objeto (OID_O_i) é alocado; e na linha 8, o vetor completo de distâncias *vet_dist* é adicionado. Por fim, após o encerramento do bloco principal, a última linha do algoritmo interrompe a execução e retorna o conjunto resposta representado por δ.

Algoritmo 4: *Indexação GroupSim*

Entrada: Conjunto de objetos S, conjunto de pivôs P.
Saída: *GroupSim* com os novos objetos adicionados.

```
1  início
2      se Cada i = 0, onde i ≤ S.Tamanho então
3          dist ← distância entre S_i e P_0;
4          vet_dist[quantidade_pivos − 1] ← distância entre S_i e P_0;
5          para Cada j = 1, onde j ≤ a quantidade de pivô faça
6              vet_dist[j] ← distância entre S_i e P_0;
7          GroupSim ← dist;
8          GroupSim ← OID_O_i;
9          GroupSim ← vet_dist;
10 retorna Conjunto resposta δ
```

O Algoritmo 5 referente a *Consulta por abrangência em uma GroupSim* detalha o procedimento para recuperar objetos que estejam dentro de um raio de busca especificado. A partir de um arquivo de busca sequencial S, da estrutura *GroupSim* GB, de um conjunto de pivôs P, do objeto de consulta q e do raio de abrangência r, o algoritmo constrói o conjunto de resposta. A linha 1 inicializa a busca abrindo um cursor na estrutura GB por meio do método *abrir_cursor_proximo*, com a posição de início dada pela diferença $d(q, P_0) - r$, atribuindo o resultado à variável *lista*. Na linha 2, a primeira chave é recuperada a partir do cursor e armazenada em *cursor_dist*. A linha 3 define a variável de controle $flag$ inicialmente como verdadeira (*true*). A linha 4 estabelece um laço de repetição condicional (*enquanto*) que continuará iterando desde que a distância registrada em *cursor_dist* seja menor ou igual ao limite superior da busca, definido por $d(q, P_0) + r$. Dentro do laço principal, a linha 5 inicia uma estrutura de repetição que itera o índice i de 1 até o tamanho do conjunto de pivôs. Na linha 6, realiza-se uma avaliação baseada na desigualdade triangular, testando se *cursor_dist* é menor ou igual a $d(P_i, q) - r$ e, simultaneamente, se é maior ou igual a $d(P_i, q) + r$. Caso a condição seja atendida, a linha 7 mantém a variável $flag$ como verdadeira; nas linhas 8 e 9 (bloco senão), caso a condição falhe, o valor da $flag$ é alterado para falso. A linha 12 verifica o estado final da $flag$ após a iteração pelos pivôs. Se ela for verdadeira, a linha 13 recupera o identificador do objeto (OID) a partir da estrutura *lista*. Na linha 14, o objeto completo é resgatado da

base sequencial S utilizando o OID. A linha 15 calcula a distância real entre a consulta q e o *objeto* recuperado, armazenando o valor em *dist*. A linha 16 atesta se essa distância exata *dist* é menor ou igual ao raio da consulta r; em caso afirmativo, a linha 17 anexa o *objeto* ao conjunto de resposta δ. A linha 19 avança a busca, atualizando o valor de *cursor_dist* com a próxima chave da *lista*. Por fim, ao término de todas as iterações do laço principal, a última linha do escopo interrompe a execução e retorna o conjunto resposta acumulado em δ.

Algoritmo 5: Consulta por abrangência em uma *GroupSim*

Entrada: Arquivo de busca sequencial S, estrutura *GroupSim* GB, conjunto de pivôs P, objeto de consulta q e o raio r.
Saída: Conjunto de resposta.

```
1  início
2      lista ← GB.abrir_cursor_proximo(d(q, P_0) − r);
3      cursor_dist ← lista.proxima_chave();
4      flag ← true;
5      enquanto cursor_dist ≤ d(q, P_0) + r faça
6          para i ← 1 até tamanho de pivôs faça
7              se cursor_dist ≤ d(P_i, q) − r e cursor_dist ≥ d(P_i, q) + r então
8                  flag ← true;
9              senão
10                 flag ← false;
11         se flag então
12             OID ← lista.getOID();
13             objeto ← S.getObjeto(OID);
14             dist ← d(q, objeto);
15             se dist ≤ r então
16                 δ ← δ + objeto;
17         cursor_dist ← lista.proxima_chave();
18 retorna Conjunto resposta δ
```

O Algoritmo 6 detalha o procedimento para recuperar os k objetos mais similares a um dado objeto de consulta. A partir de um arquivo de busca sequencial S, da estrutura *GroupSim* GB, de um conjunto de pivôs P, do objeto de consulta q e da quantidade desejada de recuperações k, o algoritmo constrói e refina progressivamente o conjunto de resposta. A linha 1 inicia a rotina abrindo um cursor na estrutura GB por meio da função *abrir_cursor_proximo*, com a posição de início dada pela diferença $d(q, P_0) - r$, e atribui esse cursor à variável *lista*. A linha 2 inicializa duas variáveis de controle de fluxo de leitura dos extremos do arquivo, definindo *continue_direita_eof* e *continue_esquerda_bof* como verdadeiras (*true*). A linha 3 define a variável *raio* com o valor infinito (∞), característica padrão no início de buscas k-NN. A linha 4 estabelece as distâncias de continuação à direita (*dist_continue_direita*) e à esquerda (*dist_continue_esquerda*), ambas inicializadas também com infinito. A linha 5 inicializa as flags booleanas de navegação principal, *continue_direita* e *continue_esquerda*, como verdadeiras. A linha 6 define a variável *interseção* igualmente como verdadeira. A

linha 7 estabelece um laço de repetição condicional (*enquanto*) que continuará a execução caso *continue_direita* ou (//) *continue_esquerda* sejam verdadeiras. Dentro do laço, a linha 8 realiza uma atribuição genérica de um ALGORITMO à variável de conjunto resposta δ. A linha 9 verifica se a flag *continue_esquerda* é verdadeira; em caso afirmativo, a linha 10 atualiza $continue\_esquerda\_bof$ com a negação lógica do início do arquivo da estrutura B ($!B.bof$). De maneira análoga, a linha 12 verifica se *continue_direita* é verdadeira, e a linha 13 atualiza $continue\_direita\_eof$ com a negação do fim do arquivo ($!B.eof$). Na sequência, a linha 15 atualiza o valor de *dist_continue_direita* resgatando a distância calculada pela função $B.dist\_direita()$, enquanto a linha 16 atualiza *dist_continue_esquerda* utilizando $B.dist\_esquerda()$. O bloco iniciado na linha 17 avalia se a distância pela esquerda é menor que o limite $d(q, P_0) - r$ ou se o início do arquivo ainda não foi alcançado ($continue\_esquerda\_bof$); em caso positivo, a linha 18 reafirma *continue_esquerda* como verdadeira. A linha 20 faz o teste correspondente para o outro lado, verificando se a distância pela direita é menor que $d(q, P_0) - r$ ou se o final do arquivo não foi atingido; caso a condição seja atendida, a linha 21 mantém *continue_direita* como verdadeira. Por fim, ao término do laço principal, a última linha do escopo retorna o conjunto de resposta contido em δ.

Algoritmo 6: Consulta aos k-vizinhos mais próximos em uma *GroupSim*

Entrada: Arquivo de busca sequencial S, estrutura *GroupSim* GB, conjunto de pivôs P, objeto de consulta q e o número k de objetos que deseja recuperar.
Saída: Conjunto de resposta.

```
1  início
2      lista ← GB.abrir_cursor_proximo(d(q, P0) − r);
3      continue_direita_eof ← true, continue_esquerda_bof ← true;
4      raio ← ∞;
5      dist_continue_direita ← ∞, dist_continue_esquerda ← ∞;
6      continue_direita ← true, continue_esquerda ← true;
7      interseção ← true;
8      enquanto continue_direita || continue_esquerda faça
9          δ ← ALGORITMO;
10         se continue_esquerda então
11             continue_esquerda_bof ←!B.bof;
12         se continue_direita então
13             continue_direita_eof ←!B.eof;
14         dist_continue_direita ← B.dist_direita();
15         dist_continue_esquerda ← B.dist_esquerda();
16         se dist_continue_esquerda < d(q, P0) − r || continue_esquerda_bof então
17             continue_esquerda ← true;
18         se dist_continue_direita < d(q, P0) − r || continue_direita_eof então
19             continue_direita ← true;
20 retorna Conjunto resposta δ
```

O Algoritmo 7 tem como objetivo processar os candidatos lidos pela estrutura *GroupSim* e atualizá-los na fila de resultados da busca pelos k-vizinhos mais próximos. A lógica inicial e o processamento de validação do bloco à direita (linhas 1 a 23) executam a fil-

tragem preliminar já estabelecida no contexto do método, avaliando a interseção espacial por meio da desigualdade triangular em relação ao conjunto de pivôs e promovendo a atualização padrão dos candidatos no conjunto resposta δ sempre que a distância máxima exigida é respeitada. Na reta final da execução, consolidando a avaliação para a estrutura à esquerda, o bloco iniciado na linha 24 executa a validação do candidato filtrado. A linha 25 recupera o registro completo na base de busca sequencial S valendo-se do OID mapeado. Na linha 26, calcula-se a distância exata *dist* entre o referido objeto e a consulta q. A linha 27 encarrega-se de inserir o objeto na estrutura do conjunto δ. Imediatamente após a inserção, o bloco das linhas 28 a 30 impõe a restrição de cardinalidade da busca: caso o volume de elementos em δ ultrapasse k, o objeto com a maior distância é descartado e o raio dinâmico de busca r é reajustado para o valor dessa nova distância limite. O processo é finalizado retornando o conjunto otimizado δ.

Algoritmo 7: Verifica se um objeto pertence a consulta

Entrada: Arquivo de busca sequencial S, estrutura *GroupSim* GB, conjunto de pivôs P, objeto de consulta q e o número k de objetos que deseja recuperar.
Saída: Conjunto de resposta.

```
1  início
2      se continue_direita for verdadeiro então
3          OID ← GB.elemento_direita_chave;
4          interseção ← true;
5          para i ← 1 até o número de pivôs faça
6              se GB.dist(P_i) < d(q, P_i) − r || GB.dist(P_i) > d(q, P_i) + r então
7                  interseção ← false;
8      se interseção então
9          objeto ← S.getObjeto(OID);
10         dist ← d(objeto, q);
11         δ.add(Objeto, q);
12         se δ.tamanho > k então
13             remove objeto com maior distância em δ;
14             r ← ultimadistânciadeδ(maiordistância);
15     se continue_esquerda então
16         OID ← GB.elemento_direita_chave;
17         interseção ← true;
18         para i ← 1 até número de pivôs faça
19             se GB.dist(P_i) < d(q, P_i) − r || GB.dist(P_i) < d(q, P_i) + r então
20                 interseção ← false;
21     se interseção então
22         objeto ← S.getObjeto(OID);
23         dist ← d(objeto, q);
24         δ.add(Objeto, d);
25         se δ.tamanho > k então
26             remove objetos com maior distância em δ;
27             r ← última distância de δ(maior distância);
28 retorna Conjunto resposta δ
```

Os experimentos realizados com diferentes quantidades de pivôs mostraram que o

método *GroupSim* em comparação com os métodos *iDistance* e *OmniB-Forest* resulta em menos acessos ao disco e utiliza menos tempo para calcular as interseções entre os objetos. Os experimentos também mostraram que este método tem melhor escalabilidade para grandes bases de dados do que os outros MAM citados.

3.2 Considerações Finais

Este capítulo apresentou os principais conceitos relacionados a representação e consultas de dados complexos em um espaço métrico. Também foram apresentados os principais métodos de acesso métricos da literatura que contribuem para estruturação do *GroupSim+*, método proposto que é descrito no próximo capítulo.

Capítulo 4

GroupSim+

Considerando as contribuição presentes na literatura no contexto de indexação de dados complexos, um novo método de acesso métrico foi elaborado. Este capítulo descreve o método denominado *GroupSim+* e apresenta figuras e pseudo-códigos para um melhor entendimento do seu funcionamento. A Seção 4.1 descreve o método desenvolvido. A realização de uma consulta por abrangência nessa estrutura é apresentada na Seção 4.2. E por fim, a Seção 4.3 descreve a consulta aos k-vizinhos mais próximos.

4.1 O método *GroupSim+*

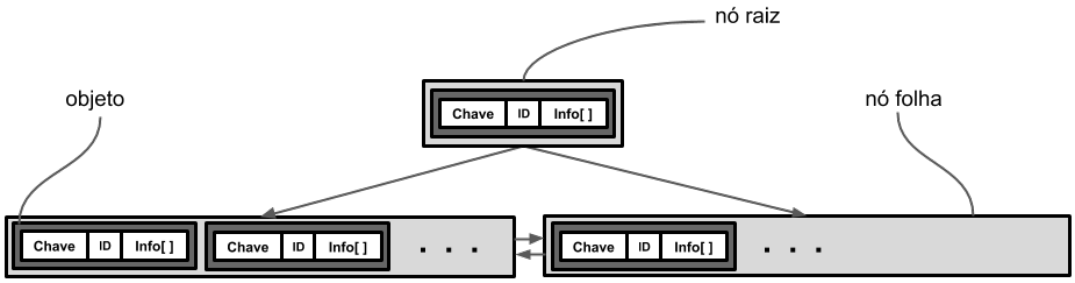
O método de acesso métrico *GroupSim+* foi elaborado incorporando a estratégia de mapeamento unidimensional ilustrada anteriormente na Figura 7. Essa estratégia explora o uso de pivôs para estabelecer uma ordem entre os objetos a serem indexados e recuperados. O método *GroupSim+* desenvolvido no trabalho descrito nesta dissertação explorou vulnerabilidades identificadas em MAM correlatos por meio da utilização de estratégias, como a técnica de particionamento do *iDistance* e o armazenamento de um vetor de distâncias dos objetos em relação a todos os pivôs, semelhante a abordagem do *GroupSim*.

No contexto de consultas por similaridade é essencial estabelecer técnicas para a realização de podas, ou seja, definir estratégias que limitem a região de busca por um determinado objeto. Essas estratégias permitem diminuir a quantidade de distâncias calculadas para a realização de consultas e consequentemente permitem reduzir o tempo de resposta. Como estratégia para realização de podas, em uma consulta por abrangência, por exemplo, o MAM *OmniB-Forest* coleta os identificadores dos objetos até atingir o raio da consulta em todas as árvores $B+$ e então computa a interseção dos identificadores nessas regiões de abrangência. No método *iDistance* a poda é limitada apenas pelo anel do pivô de cada partição, porém para cada objeto a ser verificado. Semelhante aos demais métodos, é utilizado o identificador do objeto para recuperá-lo e calcular a distância em relação ao objeto real.

Como estratégia para diminuir a região de busca, o método *GroupSim* emprega uma árvore $B+$ para indexação das distâncias dos objetos em relação a um pivô e armazena um vetor de distâncias para os demais pivôs permitindo o descarte de identificadores pela região mínima de poda com custo menor. Entretanto, ao indexar os objetos de dados em relação a apenas um pivô, os anéis formados por um raio em uma consulta podem abranger um volume grande de identificadores. A hipótese para o desenvolvimento do método *GroupSim+* é que a combinação do particionamento do espaço, característica incorporada do *iDistance*, e o armazenamento do vetor de distâncias para os demais pivôs do método *GroupSim* resultará no estreitamento das regiões mínimas de poda.

O *GroupSim+* é implementado por meio da extensão da árvore B+ e apenas uma árvore é utilizada no processo de indexação. Os objetos armazenados nos nós folha armazenam as seguintes entradas: $< chave, id, info[\,] >$, onde a ordenação é dada pela *chave*, representada pela distância de um objeto ao pivô mais próximo, o *id* é um identificador para o objeto e $info[\,]$ é um vetor de distâncias do objeto em relação aos demais pivôs. Um exemplo da estrutura *GroupSim+* é ilustrado na Figura 10.

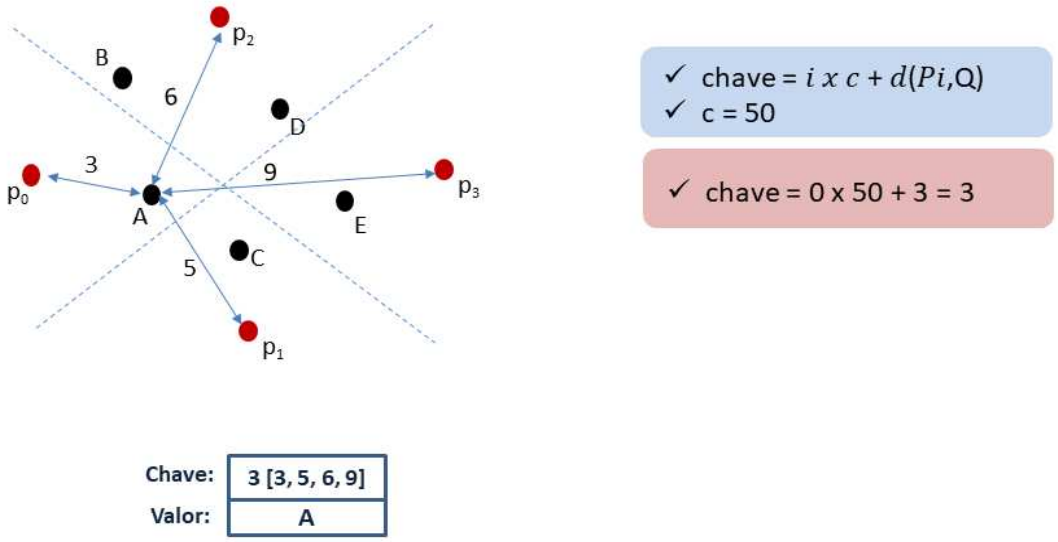
Figura 10 – Exemplo da estrutura *GroupSim+* contendo três nós: um nó raiz e dois nós folhas. Devido a estrutura da árvore B+ os nós folhas podem ser percorridos sequencialmente. Há três objetos indexados contendo as seguintes informações: a chave é a distância do objeto para o pivô mais próximo, o ID é o identificador do objeto e $info[\,]$ é um vetor de distâncias do objeto em relação aos demais pivôs.



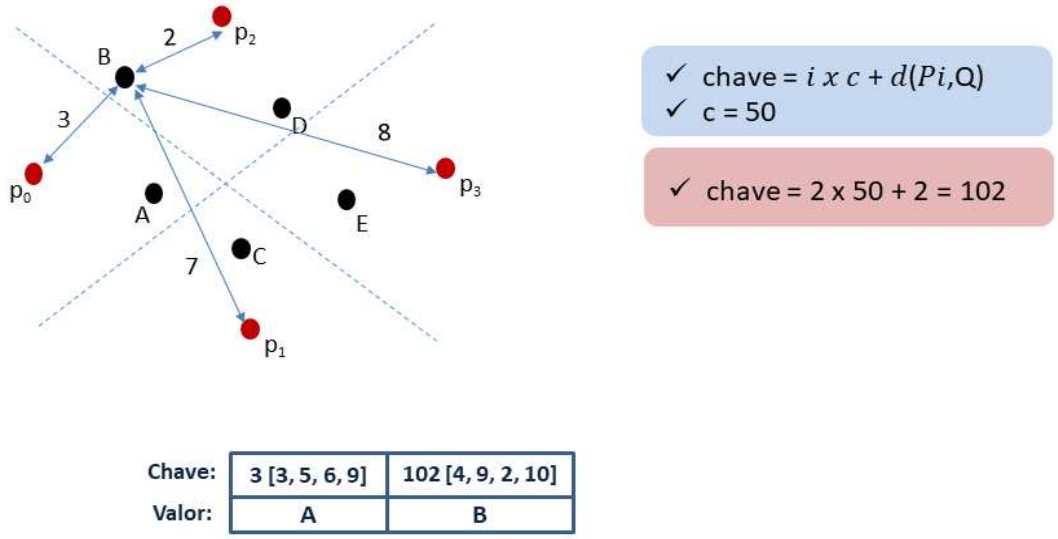
A Figura 11 ilustra o processo de indexação de objetos bidimensionais. A Figura 11 (a) o objeto A é indexado, tendo como chave de indexação o valor resultante da Equação 7, apresentada no iDistance. Também é ilustrado a construção da árvore, contendo na representação do objeto a chave de indexação, o vetor de distâncias. O campo valor simula a referencia ao objeto real. Na Figura 11(b) ilustra a indexação do objeto B, semelhante ao anterior. Na Figura 11(c) ilustra a árvore construída após a indexação de todos os objetos do conjunto, A, B, C, D e E.

Para a indexação de um conjunto dinâmico de objetos, seleciona-se inicialmente um conjunto estático de pivôs. A indexação de um objeto é feita da seguinte maneira. O objeto é inserido no final do arquivo de acesso aleatório (ORAF) e a distância do novo objeto para cada objeto do conjunto de pivôs é calculada. O particionamento dos dados na árvore $B+$ é dado pela adição de uma constante c multiplicada pelo número da partição à

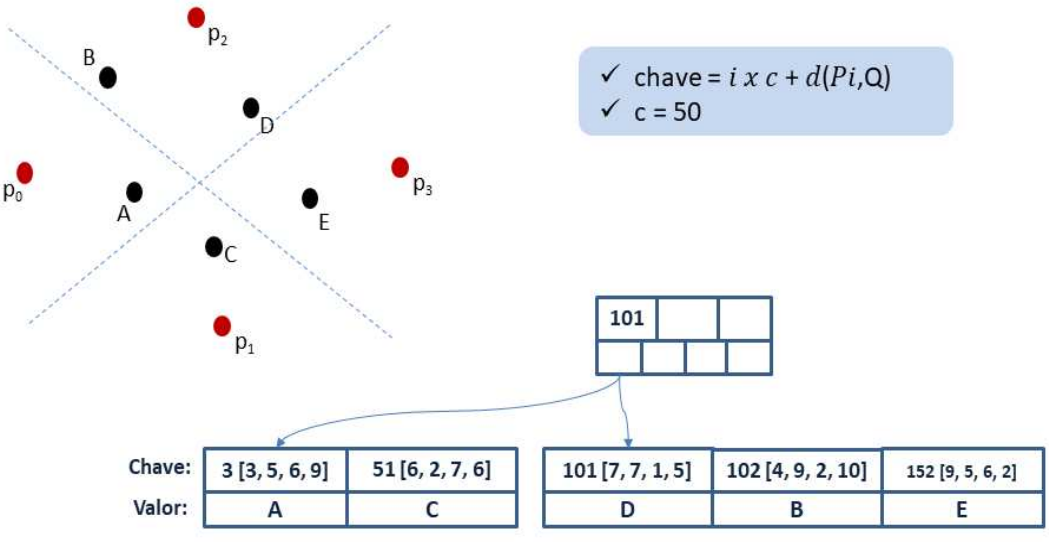
Figura 11 – Exemplo da construção da árvore B+ após mapeamento de objetos bidimensionais *GroupSim+*



(a) Indexação do objeto A. A chave de indexação é calculada em relação ao seu pivô mais próximo p_0.



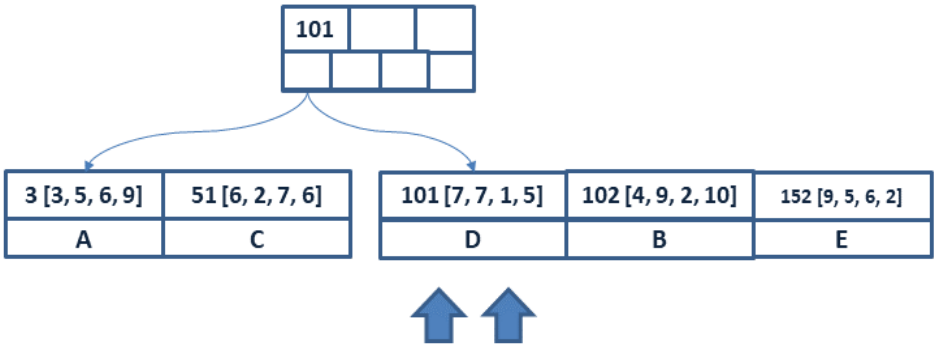
(b) Indexação do objeto B. A chave de indexação é calculada em relação ao seu pivô mais próximo p_2.



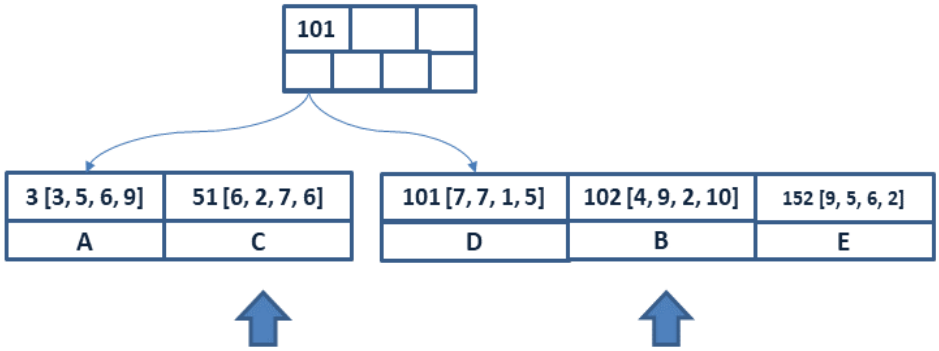
(c) Árvore construída após indexação dos objetos A, B, C, D e E.

Figura 12 – Exemplo *GroupSim+*

(a) A chave do índice é calculada de acordo com a Equação 7.



(b) A chave do índice é calculada de acordo com a Equação X.



distância do objeto em relação ao pivô mais próximo, resultando na *chave* de indexação. O *id* é a posição do objeto no ORAF e as distâncias para os demais pivôs são armazenadas em $info[\,]$. A constante c deve ser suficientemente grande para que não haja sobreposição entre os intervalos de diferentes partições. O pseudo-código presente no Algoritmo 8 apresenta uma descrição detalhada desse processo.

O Algoritmo 8 recebe como entrada o objeto a ser inserido O e um conjunto de pivôs P previamente definidos. Como saída é retornada a estrutura contendo o novo objeto indexado. Na linha 2 o objeto é inserido no ORAF e a sua posição é retornada para variável $offset$. Das linhas 3 à 10 é realizado o mapeamento dos objetos, para isso, um vetor contendo a distância do objeto em relação a todos os pivôs pertencentes a P é preenchido. Nesse trecho também é definido o pivô que possui a menor distância em relação ao objeto que será indexado. O pivô mais próximo será utilizado para definir a partição que o objeto irá pertencer, estrutura semelhante a ilustração da Figura 8 do método *iDistance* apresentada na Seção 3.1.3. Na linha 10 é chamado o método de adição de um objeto próprio da árvore B+, tendo como parâmetros a chave utilizada para mapeamento e indexação do objeto, o $offset$ e o vetor de distâncias $vet_dist[]$. A chave é definida pela Equação 8, onde *pivo_proximo* é o índice do pivô mais próximo do objeto que será indexado. A constante c é calculada com base na maior distância entre pares de objetos do conjunto e tem como objetivo separar os objetos de cada partição no

mapeamento unidimensional.

$$y = pivo\_proximo * c + d(O, P_{pivo\_proximo}) \tag{8}$$

Algoritmo 8: Indexação com *GroupSim+*

Entrada: Objeto a ser inserido O, conjunto de pivôs P.
Saída: Estrutura com o novo objeto adicionado.

```
1  início
2      offset ← oraf.adiciona(O);
3      menor_dist ← MAX;
4      pivo_proximo ← −1;
5      para cada i = 0, onde i ≤ |P| faça
6          vet_dist[i] ← d(O,P_i);
7          se vet_dist[i] ≤ menor_dist então
8              pivo_proximo ← i;
9              menor_dist ← d(O, P_i);
10     ArvoreB+.adicionar(pivo_proximo * c + menor_dist, vet_dist, offset);
11 retorna GroupSim+
```

A seguir são apresentados os algoritmos de consulta por abrangência e aos k-vizinhos mais próximos.

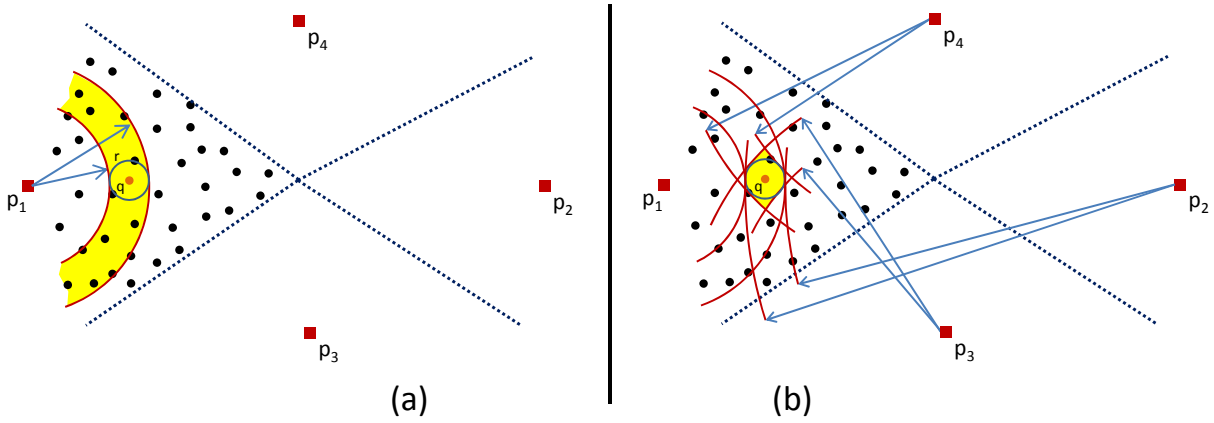
4.2 Consulta por abrangência

Para realizar uma consulta por abrangência é necessário informar como entrada o objeto de consulta q e um raio r de abrangência. Como saída é retornada uma lista de objetos que estão a até r de distância de q. No método *GroupSim+*, inicia-se procurando na árvore $B+$ pela chave mais próxima de $d(q, P_i) - r$ em cada partição i representada pelo pivô P_i. Todas as entradas com chaves no intervalo $d(q, P_i) - r$ a $d(q, P_i) + r$ devem ser verificadas (região em amarelo apresentada na Figura 13a), sendo que as distâncias armazenadas em $info[\,]$ permitem o descarte das entradas que certamente não fazem parte do conjunto resposta pela propriedade da desigualdade triangular (região em amarelo da Figura 13b). Essa abordagem permite evitar a recuperação dos objetos do ORAF e a computação das distâncias em relação a q. As partições não cobertas pelo raio de consulta não precisam ser verificadas.

O pseudo-código presente no Algoritmo 9 exibe os passos de uma consulta por abrangência no método *GroupSim+*. Como entrada, o algoritmo recebe o arquivo de busca sequencial S (ORAF), o conjunto de pivôs P, o objeto de consulta q e o raio r. Como saída é retornado o conjunto de resposta contendo todos os objetos que estão até o raio r de distância de q. A linha 2 indica o preenchimento de um vetor com as distâncias do

objeto q em relação a cada elemento do conjunto P. Na linha 3 é declarado o conjunto de resposta. A linha 4 inicia uma análise das partições a serem pesquisadas. Nessa análise um laço de repetição é iniciado para percorrer todas as partições indicadas pelo índice do pivô. O objeto mais distante do pivô pertencente a partição é armazenado na variável $dist\_max_i$ na linha 5, e posteriormente, na linha 6 esse valor é usado para definir se será necessário realizar a busca interna e externamente a essa partição. A Figura 14 ilustra esse passo. Se a condição da linha 6 for verdadeira a busca acontecerá dentro e fora da partição atual, conforme apresentado no pseudo-código no Algoritmo 10. Caso contrário, a condição da linha 10 será validada e a consulta ocorrerá dentro da partição atual, conforme apresentado no Algoritmo 11.

Figura 13 – Região mínima de poda no *GroupSim+* para a partição representada pelo pivô P_1 com base no objeto de consulta q e raio r. (a) Anel restringe entradas da árvore $B+$ que precisam ser verificadas. (b) Distâncias armazenadas em $info[\ ]$ permitem evitar a recuperação dos objetos do ORAF que não estiverem na região mínima de poda.

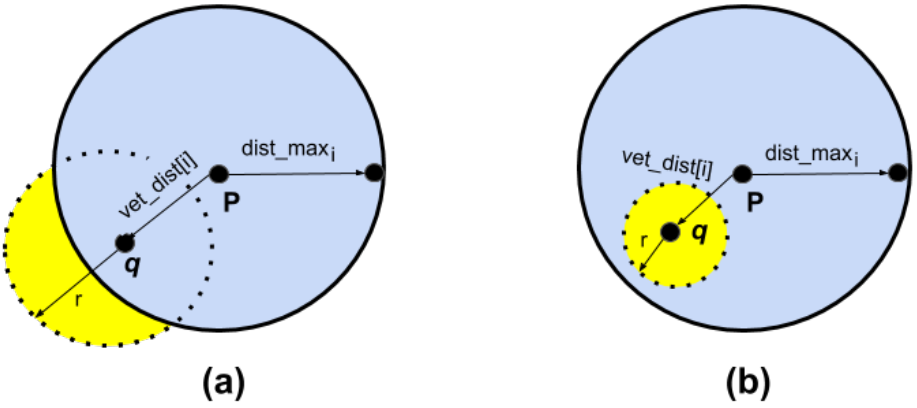


Algoritmo 9: Consulta por abrangência em uma *GroupSim+*

Entrada: Arquivo de busca sequencial S, conjunto de pivôs P, objeto de consulta q e raio r.
Saída: Conjunto de resposta.

```
1  início
2      vet_dist[|P|] ← distância de q em relação a cada pivô de P;
3      δ ← NULL;
4      para i ← 0, até i < |P| faça
5          dist_max_i ← a distância do objeto mais distante do pivô dessa partição ;
6          se (vet_dist[i] + r) ≤ dist_max_i então
7              //abrangência da consulta dentro da partição, anda para ambos lados
8              Algoritmo 10 (S, q, r, i, vet_dist[|P|, δ)
9          senão
10             se (vet_dist[i] − r) ≤ dist_max_i então
11                 //abrangência da interseção da região, anda apenas a partir da direita a partir da
                     borda
12                 Algoritmo 11 (S, q, r, i, vet_dist[|P|, δ)
13 retorna Conjunto resposta δ
```

Figura 14 – O raio da partição é definido pela $dist\_max_i$, a distância entre o pivô P_i representante da partição e o objeto mais distante dessa partição, $vet\_dist[i]$ contém a distância do objeto de consulta q em relação a P_i. Ao comparar se $dist\_max_i$ é maior ou igual a $vet\_dist[i]$ somado ao raio r, é possível definir se a busca será realizada interna e externamente. (a) A região amarela representa uma parte da região de busca que não pertence a essa partição. (b) A região de busca está incorporada na partição atual.



O Algoritmo 10 detalha a realização da consulta dentro e fora da partição atual. Como entrada o Algoritmo 10 recebe o arquivo de busca sequencial S, o objeto de consulta q, o raio r, o índice do pivô da partição atual i, o vetor de distâncias calculadas vet_dist e o conjunto de resposta δ. Como saída o Algoritmo 10 atualiza o conjunto de resposta. Na linha 2 o cursor é aberto na partição atual somada a distância do objeto em relação ao pivô dessa partição. A partir disso, são definidos nas linhas 3 e 4 dois cursores que a partir dessa posição irão percorrer o mapeamento unidimensional, um caminhando para o lado esquerdo e outro para o lado direito, até o limite de distância r. Nessas mesmas linhas também são criadas as variáveis que irão validar se ainda há objetos a serem conferidos, ou seja se o cursores devem continuar a percorrer a estrutura. A linha 5 valida se tem algum objeto a esquerda da posição inicial, ou seja, se essa não é a primeira posição do mapeamento. Se houver, na linha 6 o $cursor\_esquerda\_bof$ é atribuído como falso indicando que deste lado o aquivo ainda não chegou ao fim, e por isso deve ser percorrido. As mesmas condições são verificadas para o cursor direito nas linhas 7 e 8. Um laço de repetição é iniciado na linha 9, onde as instruções presentes nesse bloco de código serão executadas enquanto houver dados a serem percorridos, conforme condição da linha 28. A linha 12 torna verdadeiro caso o objeto faça parte da interseção, ou seja, da região mínima de busca. Se a condição for verdadeira, na linha 13, o objeto é recuperado, a distância real do objeto é calculada e é validado se o objeto faz parte do conjunto de resposta, se sim, ele é adicionado ao δ. A função que descreve a recuperação do objeto está descrita no Algoritmo 12. Nas linhas 14 até 17 ocorre o mesmo procedimento para o cursor direito. Após conferir se ainda há objetos a serem percorridos a esquerda, na linha 19 e linha 20, são atualizados as variáveis $cursor\_esquerda\_bof$ e $dist\_cursor\_esquerda$. A primeira

incrementa a posição do cursor a esquerda, recebendo verdadeiro caso houver mais objetos a serem lidos e falso caso não haja. A variável *dist_cursor_esquerda* recebe distância do objeto apontado pelo cursor esquerdo em relação a partição atual. O mesmo procedimento ocorre para o cursor direito nas linhas 21, 22 e 23. Por fim, na linha 26 é conferido se chegou ao limite da busca, seja pelo final do arquivo, ou por chegar ao final do anel que define a região de busca.

A consulta interna a partição é descrita no Algoritmo 11. Como entrada o Algoritmo Algoritmo 11 recebe o arquivo de busca sequencial S, o índice do pivô da partição atual i, o objeto de consulta q e o raio r. Como saída o Algoritmo Algoritmo 11 atualiza o conjunto de resposta. A linha 2 verifica se a distância do objeto em relação ao pivô é menor do que o raio, se a condição for verdadeira o cursor é aberto no início da partição, caso contrário o cursor é aberto no início do anel de busca definido pela posição do particionamento menos o raio, conforme demonstrado nas linhas 3 e 5. A partir da posição inicial do cursor um laço de repetição é iniciado para percorrer os objetos até o final do arquivo ou o final do anel de busca definido pelo raio, instruções descritas da linha 6 até a linha 14.

O método de recuperação do objeto é chamado quando já houve uma validação prévia de que o objeto faz parte do conjunto de resposta. Por esse motivo, esse método está presente em diferentes trechos das consultas descritas. O Algoritmo 12 descreve esse procedimento. Esse algoritmo recebe como entrada o arquivo sequencial de busca S, o ID do objeto que será recuperado e o conjunto de resposta δ. Utilizando o ID, na linha 2 é possível consultar a posição do objeto no arquivo de busca sequencial S. Desse modo, o objeto real é recuperado do arquivo na linha 3 e na linha 4 a distância real entre o objeto consultado e o objeto recuperado é calculada. Na linha 5 é conferido se essa distância é menor ou igual que o raio r informado na consulta, se a condição for verdadeira o objeto é adicionado ao conjunto de resposta.

Algoritmo 12: recuperaObjeto()

Entrada: O arquivo de busca sequencial S, o ID do objeto que será recuperado e o conjunto de resposta δ

Saída: Conjunto de resposta.

```
1 início
2     offset = ArvoreB+.cursor_ID(ID);
3     obj = le_objeto(offset);
4     d = calcula_distancia_real(obj, q);
5     se d ≤ r então
6         Conjunto resposta δ ← obj;
```

Algoritmo 10: Consulta dentro e fora da partição

Entrada: Arquivo de busca sequencial S, o objeto de consulta q, o raio r, o índice do pivô da partição atual i, o vetor de distâncias calculadas vet_dist e o conjunto de resposta δ
Saída: Conjunto de resposta.

```
1  início
2    ArvoreB+.abrir_cursor_proximo(i * c + vet_dist[i]);
3    cursor_direita_eof ← false, continue_direita ← true ;
4    cursor_esquerda_bof ← false, continue_esquerda ← false ;
5    se ArvoreB+.cursor_esquerdo_aberto() então
6      cursor_esquerda_bof ← false, continue_esquerda ← true ;
7    senão
8      cursor_direita_eof ← true, continue_direita ← false ;
9    faça
10     se continue_esquerda então
11       ok = ArvoreB+.objetoNaIntersecao(ArvoreB+.cursor_esquerdo_Chave());
12       se ok então
13         recuperaObjeto(S,ArvoreB+.cursorEsqueroID(), δ);
14     se continue_direita então
15       ok = ArvoreB+.objetoNaIntersecao(ArvoreB+.cursor_direito_chave());
16       se ok então
17         recuperaObjeto(S,ArvoreB+.cursorDireitoID(), δ);
18     se continue_esquerda então
19       cursor_esquerda_bof ← ArvoreB+.cursor_esquerdo();
20       dist_cursor_esquerda ← ArvoreB+.cursor_esquerdo_chave() − i * c;
21     se continue_direita então
22       cursor_direita_eof ← ArvoreB+.cursor_direita();
23       dist_cursor_direita ← ArvoreB+.cursor_direita_chave() − i * c ;
24     se (dist_cursor_esquerda < vet_dist[i] − r) || (dist_cursor_esquerda < 0 ||
        cursor_esquerda_bof então
25       continue_esquerda ← false;
26     se (dist_cursor_direita > vet_dist[i] + r) || (dist_cursor_direita > dist_max_i ||
        cursor_direita_eof então
27       continue_direita ← false;
28   enquanto (continue_esquerda || continue_direita);
29 retorna Conjunto resposta δ
```

4.3 Consulta aos k-vizinhos mais próximos

Uma consulta aos k-vizinhos mais próximos recebe como entrada um objeto de consulta q e um número k de objetos a serem retornados. Essa consulta retorna uma lista de até k objetos mais próximos, ordenados pela distância em relação a q. No método *GroupSim+*, inicia-se procurando na árvore $B+$ pela chave mais próxima de $\delta(q, p_i)$ na partição i representada pelo pivô p_i que contém o objeto de consulta q. A estratégia utilizada é a *start small and grow*, ou seja, uma constante Δ_r é utilizada como raio r inicial, verificam-se as entradas à esquerda e à direita até atingir o raio r, e caso não se tenha a garantia de que não há mais elementos candidatos a inserção no conjunto resposta, adiciona-se Δ_r a r. Para as demais partições, se $\delta(q, p_i) - max_i \leq r$ na partição i representada pelo

Algoritmo 11: Consulta dentro da partição

Entrada: Arquivo de busca sequencial S, o objeto de consulta q, o raio r, o índice do pivô da partição atual i, o vetor de distâncias calculadas vet_dist e o conjunto de resposta δ
Saída: Conjunto de resposta.

```
1  início
2    se (vet_dist[i] < r) então
3      ArvoreB+.abrir_cursor_proximo(i * c);
4    senão
5      ArvoreB+.abrir_cursor_proximo(i * c + vet_dist[i] − r);
6    faça
7      para cada i = 0, onde i ≤ |P| faça
8        vet_dist_pivos[i] ← ArvoreB+.cursor_anterior_chave(i)
9      ok = ArvoreB+.objetoNaIntersecao(vet_dist_pivos, vet_dist, r);
10     se ok então
11       recuperaObjeto(S,ArvoreB+.cursor_esquerdo_chave(), δ);
12     eof ←!ArvoreB+.cursor_direita;
13     dist_cursor ← ArvoreB+.cursor_direita_chave() − i * c
14   enquanto (!eof) e ArvoreB+.cursor_direita_chave() < ((i + 1) * c) e
       dist_cursor ← vet_dist[i] + r;
15 retorna Conjunto resposta δ
```

pivô p_i (onde max_i é a maior distância entre os objetos da partição), ou seja, se o raio da região de abrangência ultrapassa a partição que contém o objeto de consulta, devem ser verificadas as entradas em ordem decrescente a partir do limite max_i até $\delta(q, p_i) - r$. Para melhor desempenho, os nós folha da árvore $B+$ devem ser duplamente encadeados para permitir percorrê-los tanto em ordem ascendente quanto descendente. Em todos os casos, as distâncias armazenadas em $info[\ ]$ são utilizadas para verificar se cada objeto está dentro da região mínima de poda definida pela intersecção dos anéis obtidos a partir de cada pivô, e assim, recupera-se o objeto do ORAF apenas se o mesmo satisfizer essa condição.

O Algoritmo 13 descreve o funcionamento da consulta aos k-vizinhos mais próximos. Como entrada, o algoritmo recebe o arquivo de busca sequencial S (ORAF), o conjunto de pivôs P, o objeto de consulta q e o a quantidade de objetos a ser retornada, representada por k. Como saída é retornado o conjunto de resposta contendo até k objetos mais próximos de q. Na linha 2 é verificado se a quantidade de k informada é maior que a quantidade de elementos indexados, caso verdadeiro, o k é ajustado na linha 3. A linha 4 indica o preenchimento de um vetor com as distâncias do objeto q para cada elemento do conjunto P. Nas linhas 5 e 6, são armazenados a partição que o objeto pertence e o índice do pivô que representa essa partição, respectivamente. Também é armazenado um vetor contendo a distância da partição que contém o q em relação as outras partições, na linha 7. Na linha 8 um laço de repetição é iniciado para verificar quais partições devem ser consultadas. É verificado se a partição atual é a que contém o objeto. Caso positivo, o cursor é aberto no inicio da partição e a estrutura unidimensional é percorrida na ordem crescente, ou seja, para o lado direito de onde o cursor foi aberto (veja as linhas 9, 10

e 11). Caso a partição atual não seja a mais próxima de q, o cursor é aberto no último objeto pertencente a essa partição, e caso exista algum objeto a esquerda de onde o cursor foi aberto, a estrutura é percorrida para o lado esquerdo (veja as linhas 12 até a linha 17). As linhas 18 e 19 são executadas quando não há objetos a serem consultados a esquerda do cursor. Na linha 20, os cursores atuais são salvos. Na linha 21, após finalizar o laço de repetição a estrutura unidimensional será percorrida até que os k objetos mais próximos tenham sido encontrados ou até o fim da estrutura. Para uma melhor compreensão esse processo está descrito no Algoritmo 14.

Algoritmo 13: Consulta aos vizinhos mais próximos em uma *GroupSim+*

Entrada: Arquivo de busca sequencial S, conjunto de pivôs P, objeto de consulta q e a quantidade de vizinhos mais próximos k.
Saída: Conjunto de resposta.

```
1  início
2      se k > ArvoreB+.getNumeroDeObjetos então
3          k = ArvoreB+.getNumeroDeObjetos
4      vet_dist[] ← distância de q em relação a cada pivô de P;
5      dist_particao_Q ← a distância do pivô mais próximo do objeto de consulta q;
6      id_particao_Q ← a partição que o objeto de consulta pertence q;
7      vet_dist_particao_Q[] ← a distância da partição que q pertence em relação a todos pivôs;
8      para i = 0 até i < |P|) faça
9          se id_particao_Q == i então
10             ArvoreB+.abrir_cursor_proximo(i * c + vet_dist[i]);
11             continue_direita[i] ← true;
12         senão
13             cursor_esquerda() ← false, continue_esquerda ← false ;
14             ArvoreB+.abrir_cursor_esquerda(i * c + ArvoreB+.get_dist_max(i));
15             continue_direita[i] ← false;
16         se ArvoreB+.Se_Cursor_Esquerda_Aberto() então
17             continue_esquerda[i] ← true;
18         senão
19             continue_esquerda[i] ← false;
20         ArvoreB+.salvar_cursores(i);
21     Algoritmo 14 (S,P, q, k);
22 retorna Conjunto resposta δ
```

O Algoritmo 14 é chamado pelo Algoritmo 13, na linha 21, por isso, tem como entrada e saída os mesmos dados do Algoritmo 13. Como descrito anteriormente, a constante δ_r determina o raio r, como demonstrado na linha 3. Na linha 4, a variável $stopflag$ é inicializada com valor $false$, ela e o δ_r irão determinar o ponto de parada do laço de repetição que se inicia na linha 4 e finaliza na linha 29. Nas linhas 6 e 7 é verificado se os k objetos foram encontrados, ou se o limite do raio r foi alcançado. Caso uma dessas condições seja verdadeira, a variável $stopflag$ recebe o valor $true$ sinalizando o fim do laço de repetição. Na linha 8 é definido um alto valor para a variável $limite\_raio$, posteriormente esse valor será atualizado de acordo com o crescimento do raio, definido pela soma do δ_r e o r. As instruções do bloco de código iniciado na linha 9 até a 28

definem um laço de repetição que irá percorrer as partições. Basicamente, esse trecho define se é necessário percorrer a estrutura a partir do ponto onde o cursor foi aberto, para o lado esquerdo e direito, ou apenas para o esquerdo. Para tanto, na linha 10 o valor do objeto mais distante dessa partição é recuperado. O valor de $dist\_max_i$ define o final dessa partição, pois é o objeto mais distante do pivô que representa essa partição. Na linha 11 os cursores são recuperados. Na linha 12 é verificado se q pertence a partição atual, se a condição for verdadeira, a consulta acontecerá interna e externamente, como demonstrado nas linhas 16 e 18 que verificam se ainda há objetos a serem consultados para as duas direções (veja o Algoritmo 15 e o Algoritmo 16). Caso contrário, ou seja, q não pertence a partição atual, é verificado na linha 22 se essa partição está na região de consulta delimitada por r, se sim, o cursor é aberto na última posição dessa partição e a estrutura é percorrida pelo cursor esquerdo, conforme descrito no Algoritmo 15.

Algoritmo 14: Percorre a estrutura

Entrada: Arquivo de busca sequencial S, conjunto de pivôs P, objeto de consulta q e a quantidade de vizinhos mais próximos k.
Saída: Conjunto de resposta.

```
1  início
2      r =0;
3      stopflag ← false;
4      faça
5          r = r + δ_r;
6          se δ.numero_objetos() == k e δ.get_distancia((δ.numero_objetos() − 1) < r) então
7              stopflag ← true;
8          limite_raio ← ∞;
9          para i = 0 até i < |P|) faça
10             dist_max_i ← ArvoreB+.get_dist_max(i);
11             ArvoreB+.recupera_cursores(i);
12             se id_particao_Q == i então
13                 esquerda_menor_raio ← true;
14                 direita_menor_raio ← true;
15                 faça
16                     se continue_esquerda[i] então
17                         Algoritmo 15;
18                     se continue_direita[i] então
19                         Algoritmo 16;
20                 enquanto (continue_esquerda[i] e
                     esquerda_menor_raio)||(continue_direita[i] e direita_menor_raio);
21                 senão
22                     se vet_dist[i] − dist_max(i) ≤ r então
23                         esquerda_menor_raio ← true;
24                         faça
25                             se continue_esquerda[i] então
26                                 Algoritmo 15
27                         enquanto continue_esquerda[i]eesquerda_menor_raio;
28                     ArvoreB+.salvar_cursores(i);
29     enquanto !stopflag;
30 retorna Conjunto resposta δ
```

Algoritmo 15: Percorre a estrutura - *cursor_esquerdo*

Entrada: Arquivo de busca sequencial S, conjunto de pivôs P, objeto de consulta q e a quantidade de vizinhos mais próximos k.
Saída: Conjunto de resposta.

```
1  início
2      para cada i = 0, onde i ≤ |P| faça
3          vet_dist_pivos[i] ← ArvoreB+.cursor_anterior_Chave(i)
4      ok = ArvoreB+.objetoNaIntersecao(vet_dist_pivos, vet_dist, limite_raio);
5      se ok então
6          recuperaObjeto(S,ArvoreB+.cursor_esquerdo_chave(), δ);
7      bof ←!ArvoreB+.cursor_esquerda();
8      continue_esquerda[i] ← false;
9      se bof então
10         continue_esquerda[i] ← false;
11     dist_cursor_esquerdo ← ArvoreB+.cursor_esquerdo_chave() − i ∗ c
12     se (dist_cursor_esquerda < 0 então
13         continue_esquerda[i] ← false;
14     se (vet_dist_pivos[i] − dist_cursor_esquerda > r então
15         esquerda_menor_raio ← false;
16 retorna Conjunto resposta δ
```

O Algoritmo 15 descreve como a estrutura é percorrida para esquerda e o Algoritmo 16 para a direita. As estruturas são semelhantes e possuem como entrada e saída as mesmas informações. Como entrada, os algoritmos recebem o arquivo de busca sequencial S, o conjunto de pivôs P, o objeto de consulta q e o a quantidade de objetos a ser retornada, representada por k. O Algoritmo 15 demonstra como a estrutura é percorrida para o lado esquerdo, para isso, considerando o mapeamento unidimensional, nas linhas 2 e 3 é preenchido um vetor contendo a distância do objeto da posição a esquerda em relação a todos os pivôs. Na linha 4 utilizando esse vetor de distâncias é verificado se o o objeto a esquerda faz parte da região de consulta, se sim, a linha 6 é executada, chamando o Algoritmo 12 descrito anteriormente. A linha 7 valida se o arquivo chegou ao início, ou seja, não há mais objetos a serem lidos pelo cursor esquerdo. Se essa condição for verdadeira, na linha 10 $continue\_esquerda[i]$ recebe $false$ definindo o ponto de parada. A linha 11 verifica a distância do objeto a esquerda. Utilizando essa distância, na linha 12 é verificado se chegamos ao início da partição e na linha 13 se o raio foi atingido. Essa abordagem é semelhante a descrita no Algoritmo 16, que por se tratar de outra direção, altera apenas as validações referentes ao final das partições ou arquivo e não ao início.

Algoritmo 16: Percorre a estrutura - *cursor_direito*

Entrada: Arquivo de busca sequencial S, conjunto de pivôs P, objeto de consulta q e a quantidade de vizinhos mais próximos k.
Saída: Conjunto de resposta.

```
1  início
2      para cada i = 0, onde i ≤ |P| faça
3          vet_dist_pivos[i] ← ArvoreB+.cursor_direito_Chave(i)
4      ok = ArvoreB+.objetoNaIntersecao(vet_dist_pivos, vet_dist, limite_raio);
5      se ok então
6          recuperaObjeto(S, ArvoreB+.cursor_direito_chave(), δ);
7      eof ←!ArvoreB+.cursor_direito();
8      se eof então
9          continue_direita[i] ← false;
10     dist_cursor_direito ← ArvoreB+.cursor_direito_chave() − i ∗ c
11     se ArvoreB+.cursor_direito_chave() > dist_max_i + i ∗ c então
12         continue_direita[i] ← false;
13     se vet_dist_pivos[i] − dist_cursor_direito > r então
14         direito_menor_raio ← false;
15 retorna Conjunto resposta δ
```

4.4 Considerações Finais

Este capítulo apresentou a descrição do método de acesso métrico proposto nessa dissertação, denominado *GroupSim+*, bem como ilustrações e pseudo-códigos que contribuem para a compreensão do seu funcionamento, tanto quanto a construção do índice quanto para a execução das consultas por abrangência e aos k-vizinhos mais próximos. O MAM *GroupSim+* combina técnicas de mapeamento unidimensional, particionamento do espaço e armazenamento de um vetor de distâncias do objeto em relação aos pivôs, a fim de, computar as regiões mínimas de poda. No próximo capítulo são apresentados os experimentos realizados e uma análise a respeito dos resultados obtidos.

Capítulo 5

Experimentos e Análise dos Resultados

Este capítulo descreve os experimentos realizados utilizando conjuntos de dados reais. A abordagem utilizada permite avaliar o desempenho do método proposto em relação aos métodos correlatos já consolidados. O capítulo está organizado da seguinte maneira, a Seção 5.1 descreve as métricas de avaliação, a infraestrutura usada nos experimentos, os conjuntos de dados e os parâmetros utilizados. Na Seção 5.2 são detalhados os experimentos realizados demonstrando as melhorias obtidas pelo método proposto em relação aos trabalhos correlatos. Uma análise do comportamento dos resultados obtidos é apresentada ao longo dessa seção.

5.1 Método para a Avaliação

Os experimentos foram implementados utilizando como base a biblioteca de métodos de acesso do Grupo de Banco de Dados e Imagens do ICMC/USP (ARBORETUM, 2019). Esta biblioteca disponibiliza diversas estruturas de indexação, e classes padronizadas (*class templates*) para criação e implementação de novas métricas de comparação, de objetos, e de manipulação de resultados, desenvolvidas em C++. A partir do código disponível nesta biblioteca foram desenvolvidos os demais MAMs comparados no trabalho descrito aqui, sendo eles: o *iDistance*, o *GroupSim* e o *GroupSim+*. As implementações dos métodos *iDistance* e *GroupSim* foram disponibilizadas pelos autores do trabalho (LIMA et al., 2016) e (RAZENTE et al., 2017b). Dentre elas estão: a árvore B+, estrutura utilizada no método proposto, e o MAM *OmniB-Forest*, um dos trabalhos correlatos descrito anteriormente no Capítulo 3 na Seção 3.1.2. Uma máquina com o sistema operacional Linux 64 bits, equipada com CPU Intel Core i7-4770, 8 GB de memória principal e disco rígido de 1 TB foi utilizada na execução dos experimentos.

Os métodos considerados na experimentação foram avaliados tanto em relação ao desempenho da construção dos índices quanto em relação ao desempenho da realização de consultas por similaridade baseadas em raio e k-vizinhos mais próximos. A parametrização dos experimentos considerou a utilização da função de distância Euclidiana e a heurística

MaxSum (SOCORRO et al., 2011) para a seleção dos conjuntos de pivôs.

O objetivo principal do método de experimentação usado consistiu em avaliar o desempenho da nova estrutura em relação aos trabalhos correlatos considerando os seguintes critérios: o tempo, cálculos de distância e acessos ao disco. Os seguintes conjuntos de dados foram utilizados na experimentação: *Colors* (FIGUEROA et al., 2007), *Covertype* (DHEERU; TANISKIDOU, 2017), *Eigenfaces* (TURK; PENTLAND, 1991), *KDD Cup* 1999 (CUP, 1999), *Letter* (DHEERU; TANISKIDOU, 2017), *Nasa* (FIGUEROA et al., 2007) e *Pendigits* (DHEERU; TANISKIDOU, 2017). A Tabela 2 apresenta as principais características dos conjuntos de dados: a cardinalidade e a quantidade de dimensões.

Tabela 2 – Conjuntos de dados considerados nos experimentos.

Identificador	Conjunto	Instâncias	Dimensões
1	colors	112.682	112
2	covertype10	581.012	10
3	eigenfaces	11.900	16
4	kddcup99r6	4.898.431	6
5	letter	20.000	16
6	nasa	40.150	20
7	pendigits	10.992	16

Os parâmetros foram definidos de modo semelhante ao trabalho desenvolvido por Chen et al. (2017), no qual foi realizada uma experimentação que compara os métodos de indexação baseados em pivôs. Em cada experimento um parâmetro é variado e os demais são fixados de acordo com o valor padrão, conforme listados na Tabela 3. As consultas por abrangência foram executadas com o raio variando de 1% à 10% da maior distância calculada entre os pares de objetos pertencentes ao mesmo conjunto. As consultas aos k-vizinhos mais próximos foram executadas com k variando de 10 em 10 até 100 elementos. A variação dos pivôs busca avaliar o desempenho da busca em estruturas construídas com diferentes pivôs. Desse modo, a árvore de indexação foi construída com pivôs variando de 3 à 15. Para cada conjunto de dados foram selecionados 100 elementos aleatoriamente para formar o conjunto de centros de consulta. Esse conjunto foi utilizado para buscas em todos os métodos de acesso avaliado

Tabela 3 – Parâmetros de configuração dos experimentos.

Parâmetro	Valor	Padrão
Quantidade de pivôs	3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	6
r	1%, 2%, 3%, 4%, 5%, 6%, 7%, 8%, 9%, 10%	4%
k	10, 20, 30, 40, 50, 60, 70, 80, 90, 100	10

Os experimentos realizados tem o objetivo de avaliar se a combinação do particionamento do espaço com o armazenamento do vetor de distâncias para os demais pivôs pode contribuir para o estreitamento das regiões mínimas de poda em relação aos métodos correlatos da literatura, *OmniB-Forest*, *iDistance* e *GroupSim*. A hipótese é que essa

abordagem diminui a quantidade de cálculos de distância, pois cada objeto armazena o vetor de distâncias já calculadas. Uma análise do comportamento dos resultados obtidos é apresentada a seguir.

5.2 Experimentos

Para todos os métodos avaliados, a inclusão de um objeto na estrutura de indexação envolve o cálculo das distâncias do elemento para cada elemento do conjunto de pivôs, a inclusão no final do arquivo ORAF, e a inclusão da entrada nas árvores $B+$ (indexação do par <chave,valor> dado pela distância de indexação e $offset$ do objeto no ORAF). Assim, os gráficos da Figura 15 apresentam a quantidade de acessos a disco (1a, 2a, 3a, 4a, 5a, 6a, 7a), cálculos de distância (1b, 2b, 3b, 4b, 5b, 6b, 7b) e tempos de execução (1c, 2c, 3c, 4c, 5c, 6c, 7c) gastos na construção dos métodos considerando a variação de pivôs de 3 até 15. Analisando esses gráficos é possível observar que o número de acessos a disco e o tempo de construção seguiu um mesmo padrão em todas as bases testadas, Nesses experimentos o método *OmniB-Forest* teve um desempenho discrepante, enquanto os outros métodos apresentaram comportamentos semelhantes. Esse comportamento pode ser justificado, pela *OmniB-Forest* usar mais de uma árvore B+, enquanto os demais métodos utilizam apenas uma (aumento linear em relação ao número de pivôs). O número médio de distâncias calculadas foi o mesmo para todos os métodos utilizados (a indexação em todos os métodos avaliados é dada pelo cálculo de distância do objeto para cada elemento do conjunto de pivôs).

A Figura 16 apresenta os gráficos obtidos com a realização de consultas por abrangência executadas com um raio variando de 1% a 10% da maior distância calculada entre todos os pares de elementos de cada conjunto (eixo das abcissas), com índices criados utilizando 6 pivôs. Analisando os gráficos da Figura 16, referente ao número médio de acessos a disco (1a, 2a, 3a, 4a, 5a, 6a e 7a), é possível notar que o método *iDistance* apresenta um número de acessos inferior ao do *GroupSim+*. Entretanto, os métodos *OmniB-Forest*, *GroupSim* e *GroupSim+* apresentam comportamento similar, resultando em quantidades inferiores de cálculos de distância quando comparados com o *iDistance*, conforme apresentado nos gráficos da Figura 16 (1b, 2b, 3b, 4b, 5b, 6b e 7b). Os gráficos de acessos a disco referem-se a contagem de acessos a disco nas árvores B+ dos métodos. Além desses acessos a disco, para cada cálculo de distância realizado, há um acesso aleatório no ORAF. Isso ocorre devido os métodos utilizarem todos os pivôs para definição da região mínima de poda enquanto o *iDistance* utiliza apenas um pivô em cada partição. Desse modo, ele acessa o disco uma quantidade menor de vezes, porém, tem um número médio de distâncias calculadas muito maior do que os demais métodos e com isso precisa recuperar uma quantidade maior de elementos do ORAF para calcular a distância real para o elemento de consulta, influenciando diretamente no tempo total de execução.

Com relação ao tempo de execução das consultas apresentado nos gráfico da Figura 16 (1c, 2c, 3c, 4c, 5c, 6c, 7c), o *GroupSim+* obteve o menor tempo em quatro das sete bases apresentadas, sendo elas, a *Colors*, *Covertype10*, *Nasa* e *Pendigits*. Considerando o conjunto Nasa, o método proposto apresentou redução de 41% para o raio r = 1% e 47% para o raio r = 10% em relação ao *GroupSim*. Se comparado ao *iDistance*, o *GroupSim+* apresentou redução de 64% para o raio r = 1% e 62% para o raio r = 10%. Analisando a base *Colors* a redução foi de 66% para o raio r = 1% e 9% para o raio r = 10% em relação ao *GroupSim*, e comparado ao *iDistance* a redução foi de 81% para o raio r = 1% e 11% para o raio r = 10%.

É importante observar que os métodos *GroupSim* e *GroupSim+*, considerando o tempo de resposta, sempre estão entre os melhores resultados independente da base e da quantidade de pivôs, enquanto o *OmniB-Forest* e a *iDistance* tem uma variação no desempenho. Esse comportamento pode ser observado no gráfico da Figura 16 (4c), onde o *iDistance* obtém o menor tempo, mas nos gráficos (3c) e (5c) apresenta o pior resultado entre os métodos. Do mesmo modo, há uma variação do desempenho do *OmniB-Forest* se observados os gráficos (3c) e (4c).

A Figura 17 apresenta os gráficos obtidos com a realização de consultas aos vizinhos mais próximos executadas com um k variando de 10 em 10 até 100. Os índices foram criados utilizando o valor padrão de 6 pivôs. De maneira geral, o comportamento de todos os métodos na realização de consultas aos k-vizinhos mais próximos foi semelhante ao comportamento observado nas consultas por abrangência. Isso ocorre devido ao fato de que a otimização nesses algoritmos é realizada pela convergência da região de abrangência (definida por um raio) que contém k elementos, ou seja, aumenta-se ou diminui-se o raio de busca até que se tenha a garantia de que não há outros elementos com distâncias inferiores além dos k elementos já encontrados.

Analisando os gráficos da Figura 17 referentes ao número médio de acessos a disco (1a, 2a, 3a, 4a, 5a, 6a e 7a) pode-se observar que houve um comportamento similar na maioria das bases, tendo o *iDistance* com a menor quantidade de acessos a disco e o *OmniB-Forest* com a maior quantidade. Os gráficos (3a) e (4a) apresentaram outro padrão, onde o método *GroupSim+* obteve um valor superior aos demais métodos, e no gráfico (3a) o *GroupSim* apresentou o menor número de acessos a disco. Considerando o número médio de distâncias calculadas apresentado na Figura 17 (1b, 2b, 3b, 4b, 5b, 6b e 7b) o comportamento é semelhante ao das consultas por abrangência. O método *iDistance* apresenta valores muito superiores aos demais métodos que demonstram um comportamento similar, com quantidades de distâncias calculadas muito próximas. Apenas nos gráficos (1b), (5b) e (7b) o *OmniB-Forest* apresentou valores inferiores aos métodos *GroupSim*, *GroupSim+* e *iDistance*.

Observando o tempo de resposta apresentado nos gráficos da Figura 17 (1c, 2c, 3c, 4c, 5c, 6c, 7c), o método proposto obteve o menor tempo nas bases *Nasa* e *Pendigits* (

veja os gráficos (6c) e (7c) respectivamente). No conjunto *Nasa* a redução do tempo em relação ao método *GroupSim* foi de 21% para k = 10 e 20% para k = 100. Se comparado ao *iDistance* a redução é de 58% para k = 10 e 39% para k = 100. Para os conjuntos *Colors* e *Covertype10* apresentou tempo similar. Em todas as bases os menores tempos foram apresentados pelos métodos *GroupSim* e *GroupSim+*, com valores significativamente inferiores aos demais métodos. Esse comportamento é importante pois, independente da base de dados o *GroupSim+* apresenta um bom desempenho.

Os resultados obtidos nos experimentos descritos motivaram a investigação do impacto da escolha da quantidade de pivôs na construção dos índices. A Figura 18 apresenta os resultados para consultas aos 10-vizinhos mais próximos executadas em índices construídos com 3 a 15 pivôs (eixo das abcissas). O aumento no número de pivôs impactou negativamente no tempo de execução do método *OmniB-Forest* para os conjuntos de dados testados. Os resultados obtidos permitiram observar que a partir de 4 pivôs não houve diminuição nas regiões mínimas de poda e a inclusão de pivôs aumentou linearmente o tempo para computação da intersecção dos anéis. Os métodos *GroupSim* e *GroupSim+* apresentaram comportamento praticamente constante no tempo de execução para todos os conjuntos de pivôs testados. Apesar dos gráficos indicarem uma tendência de melhora no tempo de execução do *iDistance* com o aumento do número de pivôs, a inclusão de novos elementos em um cenário dinâmico pode reverter essa tendência. Pretende-se investigar essa hipótese em trabalho futuro.

Figura 15 – Construção dos métodos com variação de pivôs de 3 até 15. (1a-7a) acessos a disco. (1b-7b) cálculos de distância. (1c-7c) tempo total em segundos.

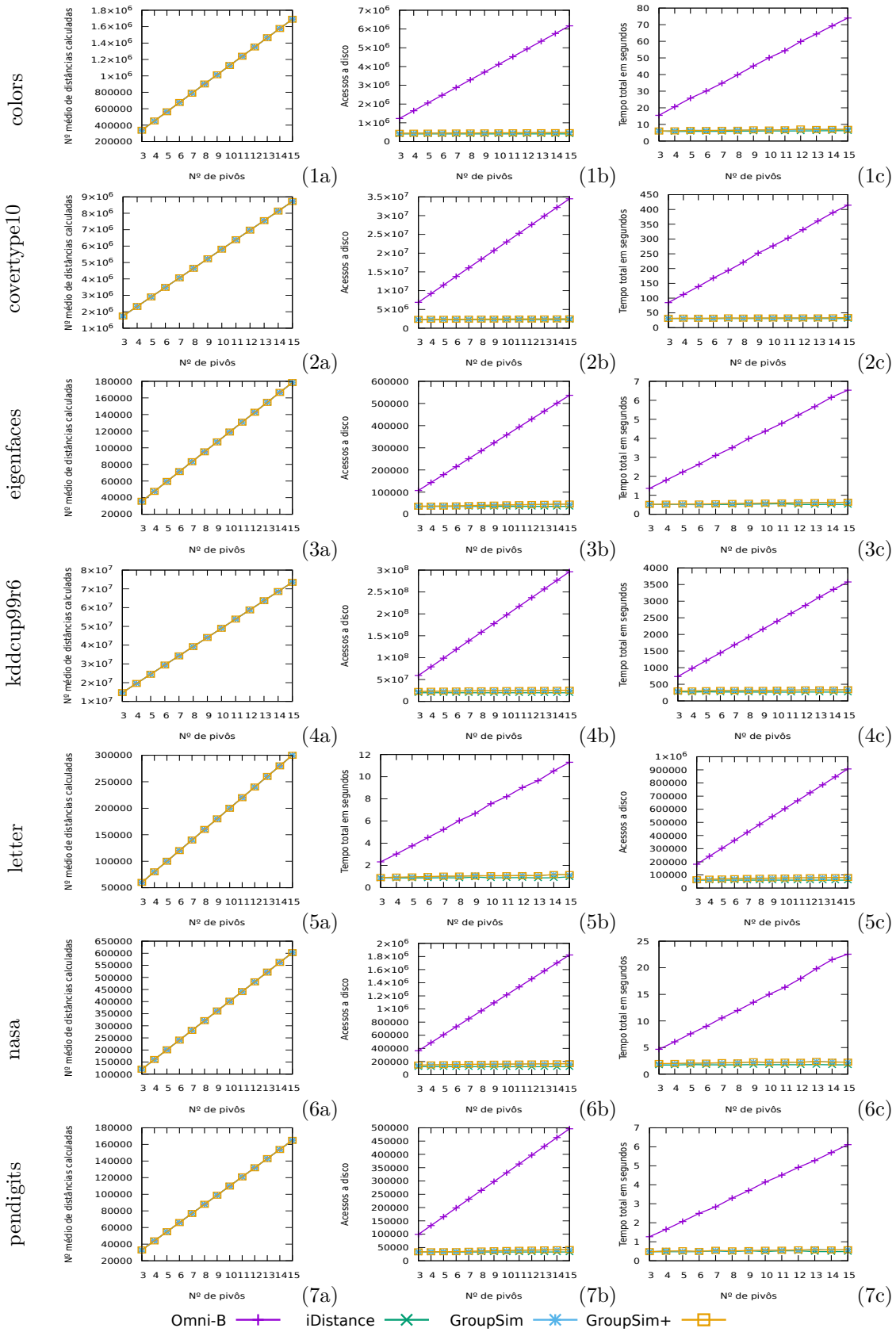


Figura 16 – Consultas por abrangência com o raio variando de 1% a 10% da maior distância calculada entre todos os pares de elementos de cada conjunto. (1a-7a) número médio de acessos a disco. (1b-7b) número médio de cálculos de distância. (1c-7c) tempo total em segundos.

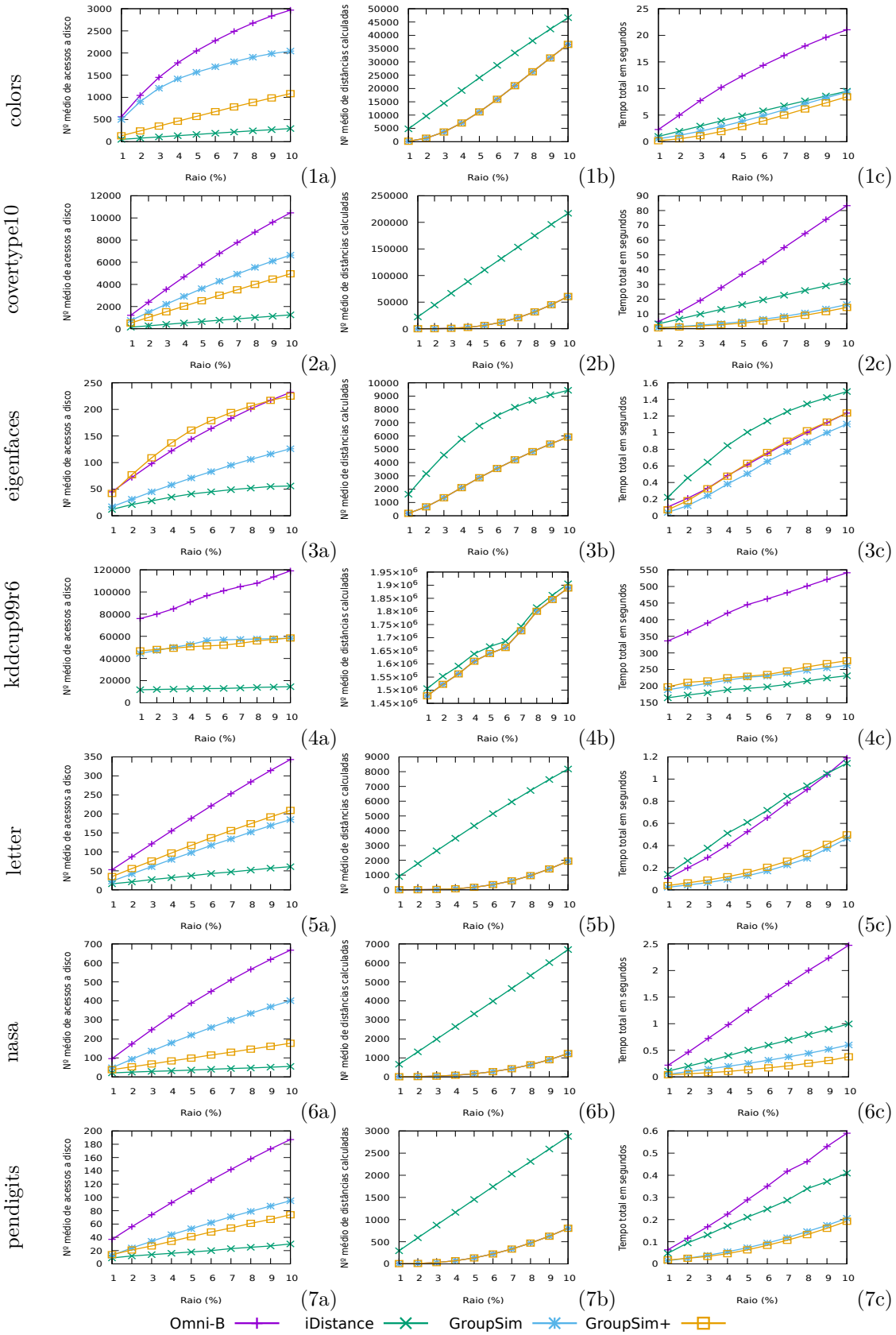


Figura 17 – Consulta aos vizinhos mais próximos. (1a-7a) número médio de acessos a disco. (1b-7b) número médio de cálculos de distância. (1c-7c) tempo total em segundos.

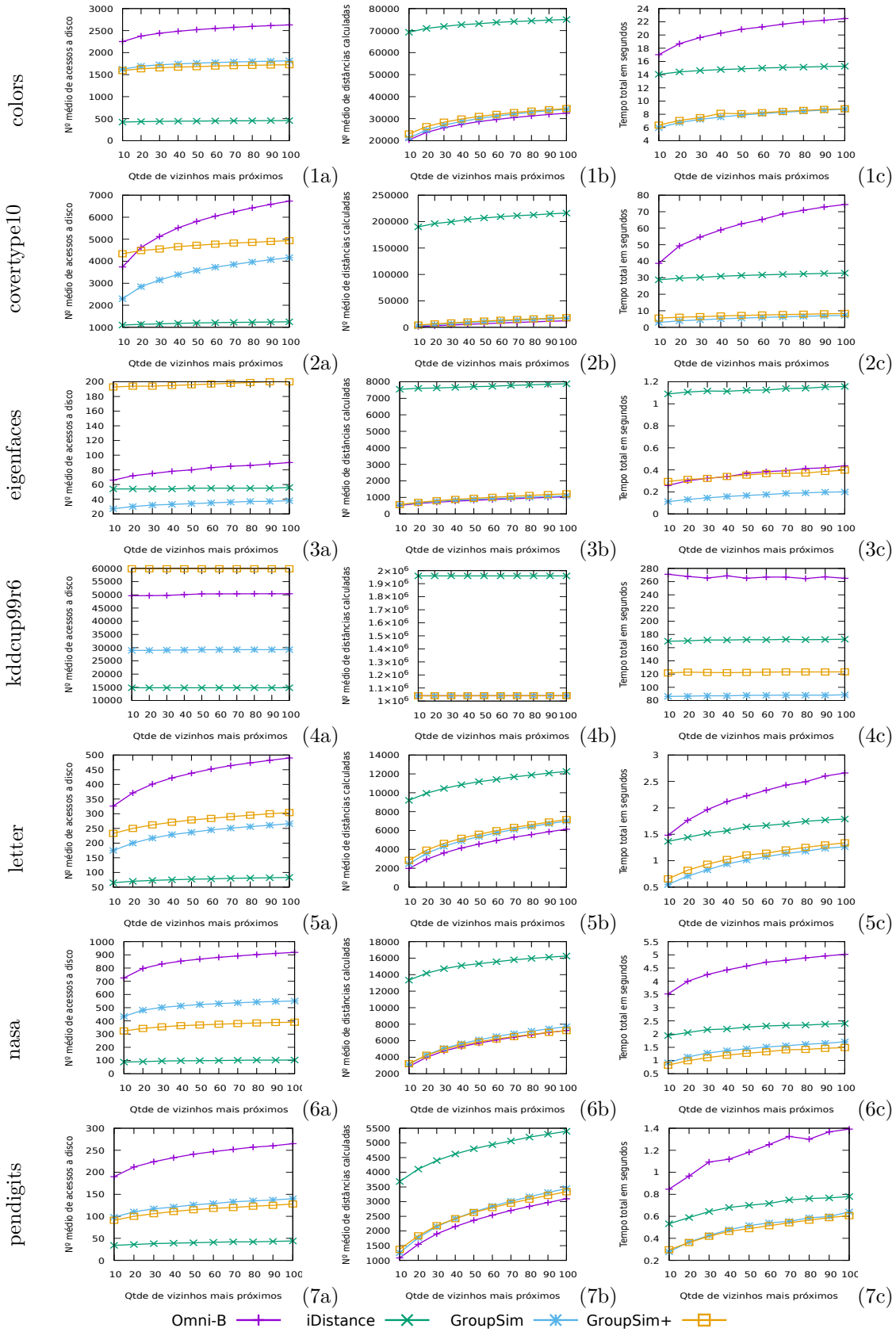
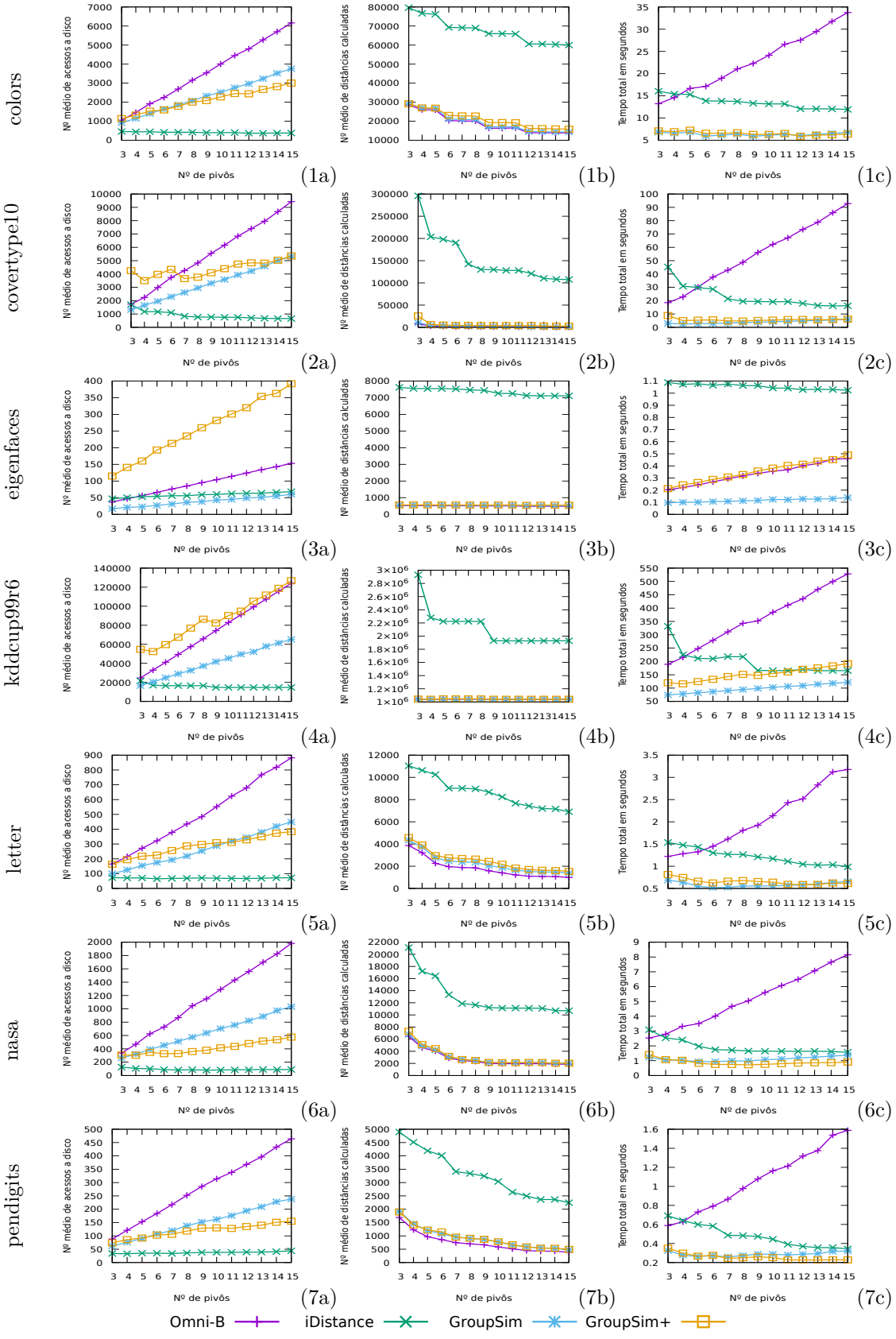


Figura 18 – Consultas aos 10-vizinhos mais próximos executadas em índices construídos com 3 a 15 pivôs. (1a-7a) número médio de acessos a disco. (1b-7b) número médio de cálculos de distância. (1c-7c) tempo total em segundos.



5.3 Considerações Finais

Nesse capítulo foram apresentados diversos experimentos com conjuntos de dados reais, com diferentes tamanhos e dimensões, visando a comparação do desempenho dos métodos *OmniB-Forest*, *iDistance*, *GroupSim* e *GroupSim+*. É importante destacar que a principal contribuição do método proposto, é que, independente do conjunto de dados e da quantidade de pivôs utilizados na construção do índice a estrutura obtém um bom desempenho, enquanto os demais métodos tem uma variação do desempenho de acordo com o conjunto de dados.

Capítulo 6

Conclusão

O método de indexação e recuperação de dados proposto considera as contribuições existentes na literatura e combina estratégias de mapeamentos unidimensionais, particionamento do espaço e armazenamento de um vetor de distâncias do objeto em relação ao conjunto de pivôs. É importante destacar que a estratégia explora o uso de uma árvore B+ na indexação e recuperação de dados por similaridade. Essa abordagem visou aumentar o desempenho da recuperação por similaridade de dados por meio do estreitamento das regiões de busca e diminuição do número de cálculos de distância durante a consulta, a um mesmo custo de indexação.

Experimentos foram realizados com sete conjuntos de dados reais com diferentes tamanhos, dimensões e dispersões. O MAM *GroupSim+* foi comparado com outros três métodos aqui descritos, que apresentam bons resultados conforme a literatura, são eles: o *OmniB-Forest*, o *iDistance* e o *GroupSim*. Os resultados obtidos mostram que o método proposto obtém um bom desempenho considerando os critérios estabelecidos, independente da base de dados, enquanto os algoritmos *OmniB-Forest* e *iDistance* possuem uma variação de acordo com a base de dados. Desse modo, o método aqui proposto tem grande potencial para lidar com diferentes bases de dados. A seguir são listadas as principais contribuições do trabalho de mestrado descrito nesta dissertação.

6.1 Principais Contribuições

As principais contribuições alcançadas com o desenvolvimento do trabalho de mestrado descrito nesta dissertação são:

- Criação de um novo MAM, denominado *GroupSim+*, que aumenta o desempenho da recuperação por similaridade de dados por meio do estreitamento das regiões de busca e diminuição do número de cálculos de distância, a um mesmo custo de indexação;

- ❏ Criação de algoritmos específicos para o MAM GroupSim+ para as principais consultas por similaridades empregadas na literatura da área, consulta por abrangência e consulta aos vizinhos mais próximos.

6.2 Trabalhos Futuros

As contribuições obtidas com a realização do trabalho de mestrado descrito aqui motivam o desenvolvimento dos seguintes trabalhos futuros:

- ❏ Investigar a incorporação dos objetos nas folhas da árvore B+, a fim de diminuir a quantidade de acessos aleatórios a disco;
- ❏ Analisar o MAM proposto com outros conjuntos de dados reais, como conjuntos de dados complexos que envolvam a extração de características extraídas a partir de redes profundas;
- ❏ Comparar o MAM proposto com outros MAM descritos na literatura, como a *PM-tree* (SKOPAL et al., 2004);
- ❏ Examinar mais a fundo os resultados experimentais obtidos, como as características de cada MAM construído e a utilização de testes de significância estatística.

6.3 Contribuições em Produção Bibliográfica

A pesquisa desenvolvida nessa dissertação gerou a publicação de um artigo científico apresentado no Simpósio Brasileiro de Banco de Dados (SBBD 2019): O artigo intitulado *"Explorando o uso de árvores B+ na Indexação de Dados por Similaridade"* apresentou os resultados obtidos nessa pesquisa.

- ❏ Farias, J. N. B.; Barioni, M. C. N.; Razente, H. L. "Explorando o uso de árvores B+ na Indexação de Dados por Similaridade. In: Anais do XXXIV Simpósio Brasileiro de Banco de Dados. SBC, 2019. p. 163-168.

Referências

AGRAWAL, R.; FALOUTSOS, C.; SWAMI, A. Efficient similarity search in sequence databases. In: SPRINGER. *International conference on foundations of data organization and algorithms*. 1993. p. 69–84. Disponível em: <https://doi.org/10.1007/3-540-57301-1_5>.

ARBORETUM. *Arboretum: biblioteca de métodos de acesso do Grupo de Banco de Dados e Imagens do ICMC/USP*. 2019. Https://bitbucket.org/gbdi/arboretum.

ASUNCION, A.; NEWMAN, D. *UCI machine learning repository*. 2007.

BARIONI, M. C. N. *Operações de consulta por similaridade em grandes bases de dados complexos*. 2006. 145 f. Tese (Doutorado em Ciências de Computação e Matemática Computacional) — Universidade de São Paulo.

BARIONI, M. C. N.; RAZENTE, H.; MARCELINO, A. M.; TRAINA, A. J.; TRAINA, C. Open issues for partitioning clustering methods: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Wiley Online Library, v. 4, n. 3, p. 161–177, 2014. Disponível em: <https://doi.org/10.1002/widm.1127>.

BRAZDIL, P.; CARRIER, C. G.; SOARES, C.; VILALTA, R. *Metalearning: Applications to data mining*. [S.l.]: Springer Science & Business Media, 2008.

CHA, S.-H. Comprehensive survey on distance/similarity measures between probability density functions. *City*, v. 1, n. 2, p. 1, 2007.

CHÁVEZ, E.; NAVARRO, G.; BAEZA-YATES, R.; MARROQUÍN, J. L. Searching in metric spaces. *ACM computing surveys (CSUR)*, ACM, v. 33, n. 3, p. 273–321, 2001. Disponível em: <https://doi.org/10.1145/502807.502808>.

CHEN, L.; GAO, Y.; ZHENG, B.; JENSEN, C. S.; YANG, H.; YANG, K. Pivot-based metric indexing. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 10, n. 10, p. 1058–1069, 2017. Disponível em: <https://doi.org/10.14778/3115404.3115411>.

CIACCIA, P.; PATELLA MARCO, Z.; PAVEL. M-tree: An effcient access method for similarity search in metric spaces. In: SCIENTIFIC LITERATURE DIGITAL LIBRARY (CITESEER). *Proceedings of the 23rd VLDB conference, Athens, Greece*. [S.l.], 1997. p. 426–435.

COMER, D. Ubiquitous b-tree. *ACM Computing Surveys (CSUR)*, ACM, v. 11, n. 2, p. 121–137, 1979. Disponível em: <https://doi.org/10.1145/356770.356776>.

CUP, K. Dataset. *available at the following website http://kdd. ics. uci. edu/databases/kddcup99/kddcup99. html*, v. 72, 1999.

DHEERU, D.; TANISKIDOU, E. K. *UCI Machine Learning Repository*. 2017. Disponível em: <http://archive.ics.uci.edu/ml>.

FACELI, K.; LORENA, A. C.; GAMA, J.; CARVALHO, A. C. P. d. L. et al. Inteligência artificial: Uma abordagem de aprendizado de máquina. 2011.

FIGUEROA, K.; NAVARRO, G.; CHAVES, E. *Metric Spaces Library*. 2007. Http://www.sisap.org/metricspaceslibrary.html.

FILHO, R. F. S.; SOUSA, E. d.; TRAINA, A. J. M.; JR, C. T. *Desmistificando o Conceito de Consultas por Similaridade: A Busca de Novas Aplicações na Medicina*. [S.l.]: 2º Workshop de Informática Médica, 2001.

JAGADISH, H.; OOI, B.; TAN, K.; YU, C.; ZHANG, R. iDistance: an adaptive b+tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems*, ACM, v. 30, n. 2, p. 364–397, 2005. ISSN 0362-5915. Disponível em: <https://doi.org/10.1145/1071610.1071612>.

KASTER, D. d. S. *Tratamento de condições especiais para busca por similaridade em bancos de dados complexos*. 2012. Tese (Doutorado) — Universidade de São Paulo.

KRUSKAL, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, JSTOR, v. 7, n. 1, p. 48–50, 1956. Disponível em: <https://doi.org/10.1090/S0002-9939-1956-0078686-7>.

LIMA, R. L. B.; RAZENTE, H.; BARIONI, M. C. N. Um novo método de indexação para consultas por similaridade utilizando mapeamentos unidimensionais baseados em focos globais. Universidade Federal de Uberlândia, 2016.

PIXABAY. *PixaBay Stunning Free Images. Over 1.4 million royalty free stock photos and videos shared by our generous community. Using Creative Commons CC0 (Public Domain Dedication), free for commercial use and no attribution required.* 2018. Disponível em: <https://pixabay.com/>.

POLA, I. R. V. *Explorando conceitos da teoria de espaços métricos em consultas por similaridade sobre dados complexos*. 2010. 169 f. Tese (Doutorado em Ciências de Computação e Matemática Computacional) — Universidade de São Paulo.

POLA, I. R. V.; TRAINA, C.; TRAINA, A. J. M. The mm-tree: A memory-based metric tree without overlap between nodes. In: SPRINGER. *East European Conference on Advances in Databases and Information Systems*. 2007. p. 157–171. Disponível em: <https://doi.org/10.1007/978-3-540-75185-4_13>.

RAZENTE, H.; LIMA, R. B.; BARIONI, M. C. Similarity search through one-dimensional embeddings. In: *ACM Symposium on Applied Computing*. Marrakech, Marrocos: [s.n.], 2017. p. 874–879. Disponível em: <10.1145/3019612.3019674>.

RAZENTE, H.; LIMA, R. L. B.; BARIONI, M. C. N. Similarity search through one-dimensional embeddings. In: *Proceedings of the Symposium on Applied Computing*. New York, NY, USA: ACM, 2017. (SAC '17), p. 874–879. ISBN 978-1-4503-4486-9. Disponível em: <http://doi.acm.org/10.1145/3019612.3019674>.

SAMET, H. *Foundations of multidimensional and metric data structures*. [S.l.]: Morgan Kaufmann, 2006.

SANTANA, A. P.; QUEIRÓ, J. F. Introdução à algebra linear. *Gradiva, Lisboa*, p. 149–150, 2010.

SANTOS, L. F. D. *Explorando variedade em consultas por similaridade*. 2012. 72 p. Tese (Doutorado) — Universidade de São Paulo.

SKOPAL, T.; POKORNỲ, J.; SNASEL, V. Pm-tree: Pivoting metric tree for similarity search in multimedia databases. In: *ADBIS (Local Proceedings)*. [S.l.: s.n.], 2004.

SMEULDERS, A. W.; WORRING, M.; SANTINI, S.; GUPTA, A.; JAIN, R. Content-based image retrieval at the end of the early years. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 22, n. 12, p. 1349–1380, 2000. Disponível em: <https://doi.org/10.1109/34.895972>.

SOCORRO, R.; MICO, L.; ONCINA, J. A fast pivot-based indexing algorithm for metric spaces. *Pattern Recognition Letters*, v. 32, n. 11, p. 1511–1516, 2011. ISSN 0167-8655. Disponível em: <http://dx.doi.org/10.1016/j.patrec.2011.04.016>.

SOUZA, B. F. de; PRUDÊNCIO, R. B.; CARVALHO, A. de. Meta-aprendizado para recomendação de algoritmos. *Jornadas de atualização em Informática*, p. 159–208, 2001.

TRAINA, C.; FILHO, R. F. S.; TRAINA, A. J. M.; VIEIRA, M. R.; FALOUTSOS, C. The omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. *The VLDB Journal*, v. 16, n. 4, p. 483–505, Oct 2007. ISSN 0949-877X. Disponível em: <https://doi.org/10.1007/s00778-005-0178-0>.

TRAINA, C.; TRAINA, A.; SEEGER, B.; FALOUTSOS, C. Slim-trees: High performance metric trees minimizing overlap between nodes. In: ZANIOLO, C.; LOCKEMANN, P. C.; SCHOLL, M. H.; GRUST, T. (Ed.). *Advances in Database Technology – EDBT 2000*. Springer Berlin Heidelberg, 2000. p. 51–65. Disponível em: <https://doi.org/10.1007/3-540-46439-5_4>.

TURK, M.; PENTLAND, A. Eigenfaces for recognition. *Journal of cognitive neuroscience*, MIT Press, v. 3, n. 1, p. 71–86, 1991. Disponível em: <https://doi.org/10.1162/jocn.1991.3.1.71>.

WILSON, D. R.; MARTINEZ, T. R. Improved heterogeneous distance functions. *Journal of artificial intelligence research*, 1997. Disponível em: <https://doi.org/10.1613/jair.346>.

ZEZULA, P.; AMATO, G.; DOHNAL, V.; BATKO, M. *Similarity search: the metric space approach*. Springer Science & Business Media, 2006. v. 32. Disponível em: <https://doi.org/10.1007/0-387-29151-2>.