UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

**VD-Tree: Uma estratégia para redução da
sobreposição de nós em Métodos de Acesso Métricos
utilizando o Diagrama de Voronoi**

André Toshio Asanome Moriyama

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências
de Computação e Matemática Computacional (PPG-CCMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

André Toshio Asanome Moriyama

**VD-Tree: Uma estratégia para redução da
sobreposição de nós em Métodos de Acesso Métricos
utilizando o Diagrama de Voronoi**

Dissertação apresentada ao Instituto de Ciências
Matemáticas e de Computação – ICMC-USP,
como parte dos requisitos para obtenção do título
de Mestre em Ciências – Ciências de Computação e
Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e
Matemática Computacional

Orientador: Prof. Dr. Caetano Traina Júnior

USP – São Carlos
Janeiro de 2022

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

M862v Moriyama, André
VD-Tree: Uma estratégia para redução da
sobreposição de nós em Métodos de Acesso Métricos
utilizando o Diagrama de Voronoi / André Moriyama;
orientador Caetano Traina Junior. -- São Carlos,
2022.
98 p.

Dissertação (Mestrado - Programa de Pós-Graduação
em Ciências de Computação e Matemática
Computacional) -- Instituto de Ciências Matemáticas
e de Computação, Universidade de São Paulo, 2022.

1. Estrutura de indexação. 2. Método de acesso
métrico. 3. Dados complexos. 4. Consultas por
similaridade. 5. Diagrama de Voronoi. I. Traina
Junior, Caetano, orient. II. Título.

André Toshio Asanome Moriyama

**VD-Tree: A strategy to reduce the overlapping nodes in
Metric Access Methods using Voronoi Diagrams**

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Master in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Caetano Traina Júnior

USP – São Carlos
January 2022

RESUMO

MORIYAMA, A. T. A. **VD-Tree: Uma estratégia para redução da sobreposição de nós em Métodos de Acesso Métricos utilizando o Diagrama de Voronoi**. 2022. 95 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Os avanços na tecnologia proporcionaram o aumento crescente na geração de dados e nos novos tipos de dados, tornando necessário estender os SGBDs para possibilitar armazenar, recuperar e organizar novos tipos de dados como imagens, vídeos e áudios, sendo estes conhecidos como dados complexos. Para as consultas em dados complexos, não é adequado comparar objetos utilizando as relações de Ordem e Identidade, sendo então a opção mais utilizada a comparação por similaridade. Dessa maneira, com a necessidade de desenvolver novos índices para as comparações baseadas em similaridade, surgiram os Métodos de Acesso Métricos (MAMs). Entre as diversas estratégias para indexar os dados, as baseadas em árvore se destacam por possibilitar um equilíbrio entre o tempo de construção do índice e a aceleração da consulta, sendo utilizada junto com a estratégia de árvore, uma estratégia para definir a região dos nós. Entre as diversas estratégias para definir regiões, o *raio de cobertura* está dentre as mais comumente utilizadas por flexibilizar a posição do objeto na estrutura, possibilitando o controle da ocupação dos nós e a redução no custo da construção da estrutura. Porém, esta estratégia possui o problema da sobreposição de nós, que aumenta o custo para obter as respostas exatas ao realizar as consultas por similaridade. Outra estratégia que não possui o problema da sobreposição, mas que sofre com o alto custo de construção, é a baseada no Diagrama de Voronoi. Buscando reduzir o problema da sobreposição de nós, aumentando o mínimo possível o custo da construção da árvore, neste projeto de mestrado foi proposto o MAM VD-Tree que busca acelerar as consultas por similaridade por meio da redução da sobreposição, obtida com reorganizações baseadas no Diagrama de Voronoi. Resultados experimentais mostraram que o método é capaz de acelerar consultas por similaridade e reduzir a sobreposição de nós na maioria dos casos, em comparação com seu principal competidor, o Slim-Tree. A melhora no tempo gasto ocorre devido ao método criar organizações melhores dos objetos na estrutura e reduzir a sobreposição dos nós, com o custo de criar mais nós para indexar os dados.

Palavras-chave: Slim-tree, Diagrama de Voronoi, Busca por Similaridade, Métodos de Acesso Métricos.

ABSTRACT

MORIYAMA, A. T. A. **VD-Tree: A strategy to reduce the overlapping nodes in Metric Access Methods using Voronoi Diagrams**. 2022. 95 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Advances in the information technology have increased the amount of data generated daily and new types of data, making it necessary to extend DBMS to enable storing, retrieving, and organizing new types of data such as images, videos, and audio, known as complex data. It is not suitable for queries on complex data to compare objects using Order or Identity relations, so comparisons by similarity are the most employed option. With the necessity of developing new indices for comparisons based on similarity, many studies proposed several Metric Access Methods (MAMs). One of the most commonly used strategies to index complex data, tree-based strategies are commonly employed since they maintain a balance between the cost to create the index and the cost to execute the queries. Accordingly, together with the tree strategy, it is necessary to use a strategy to define the region of the nodes. Among the several strategies to define regions, the coverage radius strategy is commonly used to make the objects' position in the structure more flexible, making it possible to control the occupation of nodes and reduce the cost of building the structure. However, this strategy has the problem of overlapping nodes, which increases the cost of getting the exact answers when performing similarity queries. Another strategy that does not have the overlap problem but suffers from the high construction cost is based on the Voronoi Diagram. Seeking to reduce the problem of overlapping nodes, increasing as little as possible the cost of constructing the tree, we propose here the VD-Tree MAM to speed up similarity queries by reducing the overlap between nodes, obtained with reorganizations based on the Voronoi Diagram. Experimental results showed that the method could speed up similarity queries with better distributions of the objects in the structure and reduce overlapping nodes in most cases, compared to its main competitor Slim-Tree, with the cost of requiring more nodes to index the data.

Keywords: Slim-tree, Voronoi Diagram, Similarity search, Metric Access Methods.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de histograma de cor de uma imagem. Extraído utilizando o software Fiji.	23
Figura 2 – Exemplo de imagens com histogramas de cores iguais.	24
Figura 3 – Exemplo de consulta por abrangência	26
Figura 4 – Exemplo de consulta por abrangência pelas cidades que estão a no máximo 50km de São Carlos	27
Figura 5 – Exemplo de consulta k vizinhos mais próximos	27
Figura 6 – Estrutura em árvore.	33
Figura 7 – Exemplo de como a sobreposição de nós afeta a busca.	37
Figura 8 – Exemplo de árvore para o cálculo do Fat-Factor.	38
Figura 9 – Exemplo de uma sub-árvore antes de realizar o Slim-Down.	40
Figura 10 – Exemplo de uma sub-árvore após a execução do Slim-Down.	40
Figura 11 – Exemplo da construção de um Diagrama de Voronoi.	42
Figura 12 – Ilustração das regiões criadas pelo NOBH-tree em um espaço euclidiano. . .	44
Figura 13 – Fluxograma da inserção de elementos no VD-Tree.	48
Figura 14 – Exemplo da reorganização realizada na VD-fit.	50
Figura 15 – Comparação do tempo de execução de consultas por k -NN e abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana	58
Figura 16 – Comparação do tempo de execução de consultas por k -NN e abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan	60
Figura 17 – Comparação do tempo de execução e acessos a disco em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana	62
Figura 18 – Comparação do tempo de execução e acessos a disco em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana	63
Figura 19 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana	64
Figura 20 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana	65
Figura 21 – Comparação do tempo de execução e cálculos de distância em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana	66

Figura 22 – Comparação do tempo de execução e cálculos de distância em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana	67
Figura 23 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana	68
Figura 24 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana	69
Figura 25 – Comparação do tempo de execução e acessos a disco em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan	71
Figura 26 – Comparação do tempo de execução e acessos a disco em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan	72
Figura 27 – Comparação do tempo de execução e cálculos de distância em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan	73
Figura 28 – Comparação do tempo de execução e cálculos de distância em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan	74
Figura 29 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan	75
Figura 30 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan	76
Figura 31 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan	77
Figura 32 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan	78
Figura 33 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan e variando a geração do raio de busca por centro de consulta. . . .	79
Figura 34 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan e variando a geração do raio de busca por centro de consulta. . . .	80

Figura 35 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan e variando a geração do raio de busca por centro de consulta. . . .	81
Figura 36 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan e variando a geração do raio de busca por centro de consulta. . . .	82

LISTA DE SÍMBOLOS

T — Relação

y — Condição

s — Objeto ou elemento

$\sigma_{(y)}T$ — Operador de seleção

s_q — Centro de consulta

$\sigma_{(Rng(s_q, \xi))}T$ — Operador de seleção por abrangência

$\sigma_{(kNN(s_q, \xi))}T$ — Operador de seleção por k vizinhos mais próximos

$s_i Rng(d, \xi) s_j$ — Operador de comparação por abrangência

$s_i kNN(d, \xi) s_j$ — Operador de comparação por k vizinhos mais próximos

ξ — Raio de cobertura

δ — Função de distância métrica

$\mathbb{M}$ — Espaço métrico

$\mathbb{S}$ — Domínio de elementos

$\mathcal{N}$ — Nó

s^{rep} — Representante

s — Elemento

SUMÁRIO

1	INTRODUÇÃO	17
1.1	Considerações iniciais	17
1.2	Definição do problema	18
1.3	Contribuição	20
1.4	Organização do trabalho	20
2	CONSULTAS POR SIMILARIDADE	21
2.1	Considerações iniciais	21
2.2	Recuperação de informação em SGBDs	22
2.3	Métodos de extração de características	22
2.4	Funções de distância	24
2.5	Operadores de seleção	25
2.5.1	<i>Consultas por abrangência (Range)</i>	26
2.5.2	<i>Consultas por k vizinhos mais próximos (k-NN)</i>	26
2.5.3	<i>Extensões dos operadores</i>	28
2.6	Espaço métrico	28
2.7	Considerações finais	28
3	MÉTODOS DE ACESSO MÉTRICO	31
3.1	Considerações iniciais	31
3.2	Métodos de Acesso Métricos (MAMs)	32
3.2.1	<i>Estratégias baseadas em raio de cobertura</i>	33
3.2.2	<i>Diagrama de Voronoi</i>	41
3.2.3	<i>Raio de cobertura, hiperplanos e Diagrama de Voronoi</i>	43
3.3	Considerações finais	44
4	REDUÇÃO DE SOBREPOSIÇÃO COM O DIAGRAMA DE VORONOI	47
4.1	Considerações iniciais	47
4.2	VD-Tree	47
4.2.1	<i>VD-fit</i>	49
4.2.2	<i>Vantagens e desvantagens da VD-fit</i>	52
4.3	Considerações finais	53
5	EXPERIMENTOS E RESULTADOS	55

5.1	Considerações iniciais	55
5.2	Materiais e configurações	56
5.3	Comparações de tempo de execução	57
5.3.1	<i>Comparações de tempo de execução utilizando a função de distância Euclidiana</i>	57
5.3.2	<i>Comparações de tempo de execução utilizando a função de distância Manhattan</i>	59
5.4	Comparações de cálculos de distância e acessos a disco	61
5.4.1	<i>Comparações utilizando a função de distância Euclidiana</i>	61
5.4.2	<i>Comparações utilizando a função de distância Manhattan</i>	70
5.5	Resultados das construções das árvores	83
5.6	Considerações Finais	85
6	CONCLUSÃO	87
6.1	Considerações iniciais	87
6.2	Contribuições	88
6.2.1	<i>Publicações</i>	89
6.3	Trabalhos futuros	89
6.4	Agradecimentos	90
	REFERÊNCIAS	91

INTRODUÇÃO

1.1 Considerações iniciais

Os Sistemas de Gerenciamento de Banco de Dados (SGBDs) foram inicialmente projetados para armazenar apenas dados escalares, dos quais são exemplos os números, datas e pequenas cadeias de caracteres (CODD, 1990). Com os avanços da tecnologia, tornou-se cada vez mais necessário armazenar e recuperar novos tipos de dados, sendo estes conhecidos como dados complexos (ZEZULA *et al.*, 2006). Alguns exemplos de dados complexos são imagens, vídeos, áudios e séries temporais. Considerando o grande aumento na quantidade de dados complexos sendo gerados diariamente, é essencial o uso de índices para acelerar as consultas (BOZKAYA; ÖZSOYOGLU, 1999; CHEN *et al.*, 2017; KOUAHLA *et al.*, 2019).

Ao realizar consultas sobre dados complexos, não é possível utilizar a Relação de Ordem ($<, \leq, >, \geq$) para comparar os objetos, uma vez que não há uma forma clara de definir ordem entre eles. Apesar de ser possível utilizar a Relação de Identidade ($=, \neq$), na grande maioria dos casos a busca por objetos idênticos tem pouco valor semântico. Dessa forma, os índices desenvolvidos baseados nas relações de identidade e ordem não são adequados para dados complexos. As estratégias mais utilizadas para comparar dados complexos é medindo a similaridade entre os objetos (POLA; TRAINA-JR; TRAINA, 2014). É comum utilizar métodos de extração de características para obter descrições das características dos dados complexos, e comparar as características utilizando uma função de distância. Um espaço métrico é definido pelo domínio de dados e uma função de distância. O espaço métrico possui propriedades bem definidas que podem ser utilizadas durante as buscas de consultas por similaridade. Para consultas por similaridade, a teoria dos espaços métricos é uma das mais utilizadas, por ser aplicável para vários tipos de objetos complexos (CHEN *et al.*, 2020). Para se utilizar a teoria dos espaços métricos é necessário que a função de distância utilizada seja métrica, e para isso ela deve atender a quatro propriedades: não-negatividades, identidade dos indiscerníveis, simetria e desigualdade triangular.

Devido à necessidade de extrair características e realizar cálculos de distância sobre objetos complexos, consultas por similaridade são operações custosas. Para grandes conjuntos de dados, é essencial ter suporte de estruturas para indexar o espaço de busca, de modo a acelerar consultas. Para realizar a indexação de objetos complexos surgiram os Métodos de Acesso Métricos (MAMs). Os MAMs são estruturas que possibilitam o acesso eficiente aos dados, utilizando as propriedades do espaço métrico para a realizar podas no espaço de busca. Desse modo, é essencial que a função de distância escolhida para comparar os dados seja métrica, possibilitando o uso de MAMs (CIACCIA; PATELLA, 2002; HANYF; SILKAN, 2020).

1.2 Definição do problema

Considerando que comparações por similaridade são operações custosas, e com o crescente aumento na quantidade de dados complexos gerados, tornou-se cada vez mais necessário o uso índices para acelerar as consultas por similaridade. Para lidar com esse problema, diversos MAMs vem sendo propostos baseados em estratégias como hash, pré-computação de distâncias e hierárquicas (POLA; TRAINA-JR; TRAINA, 2014). Especificamente, MAMs hierárquicos consistem em hierarquicamente dividir o espaço de busca em regiões, elegendo um elemento como representante para cada região. MAMs hierárquicos são comumente estruturados como árvores (KOUAHLA *et al.*, 2019), de maneira que as regiões geradas são conhecidas como nós. O nó mais alto na hierarquia é o nó raiz, e os nós mais baixos na hierarquia são os nós folhas. Duas características importantes em MAMs hierárquicos são:

1. **Permitir ou não atualizações após a construção da estrutura:** caso permita, é uma estrutura dinâmica, do contrário é uma estrutura estática.
2. **Ser uma estrutura balanceada ou não:** Estruturas balanceadas possuem o mesmo número de níveis da raiz para qualquer nó folha. Em estruturas não balanceadas, é possível que exista sub-árvores muito grandes, o que aumenta o custo de consultas que necessitam explorar essas sub-árvores

Um dos primeiros MAMs hierárquicos e dinâmicos para dados complexos baseado em disco foi a estrutura M-Tree (CIACCIA; PATELLA; ZEZULA, 1997). Essa estrutura utiliza de *raios de cobertura* para determinar o limite das regiões, sendo estas armazenadas nos nós, e constrói uma estrutura em formato de árvore utilizando nós internos, que armazenam ponteiros para outros nós, e nós folha que armazenam os elementos. Os elementos podem ser associados a apenas um representante, e somente se eles estão dentro do *raio de cobertura* do nó. Porém, devido à possibilidade de ocorrer sobreposição entre os *raios de cobertura*, nem todo elemento dentro do *raio de cobertura* de um representante está associado a ele.

Um dos maiores problemas dos MAMs hierárquicos baseados em *raio de cobertura* é a sobreposição de nós, que ocorre quando o *raio de cobertura* de dois ou mais nós estão

sobrepostos. Para garantir que a resposta obtida por uma consulta por similaridade esteja correta, é necessário explorar todas as sub-árvores que cobrem a região de busca. Desse modo, quanto mais nós estão sobrepostos, maior é a chance da região de busca ser coberta por mais nós, o que aumenta o número de cálculos de distância e de acessos a disco necessários para responder a consulta.

Para reduzir o problema da sobreposição presente na M-Tree, foi proposta a Slim-Tree (TRAINA-JR *et al.*, 2002a). De forma semelhante à M-Tree, a estrutura da Slim-Tree utiliza dois tipos de nós para organizar os elementos, sendo eles os (i) nós folha, que armazenam os dados, e (ii) os nós diretórios, que armazenam ponteiros para outros nós. A Slim-Tree propôs técnicas como o Fat-Factor e o Bloat-Factor para medir a sobreposição em árvores, e propôs o Slim-Down e novas heurísticas de particionamento de nós para reduzir a sobreposição de nós. Apesar das técnicas propostas pela Slim-Tree reduzirem significativamente a sobreposição de nós, o problema ainda persiste, principalmente em nós próximos ao topo da árvore (TRAINA-JR *et al.*, 2002b).

Outra estratégia utilizada para lidar com o problema da sobreposição de nós é baseada no armazenamento de informações extras que auxiliam na filtragem de nós que devem ser explorados. Alguns exemplos de estruturas que utilizam dessa técnica são a B^+ -tree (ZHOU *et al.*, 2003) e a BM^+ -tree (ZHOU *et al.*, 2005).

Outra estratégia consiste em fazer o uso de heurísticas de particionamento do espaço para reduzir a sobreposição de nós de um MAM hierárquico. Uma estratégia livre do problema da sobreposição de nós é o Diagrama de Voronoi. O Diagrama de Voronoi é construído de forma que todos os elementos dentro de cada região estão mais próximos do representante de sua região do que de qualquer outro representante. Um exemplo de estrutura baseada no Diagrama de Voronoi é a GNAT (BRIN, 1995). Ela não possui o problema de sobreposição, porém é computacionalmente custosa de construir, devido ao custo de garantir as regras do Diagrama de Voronoi, sendo esta uma estrutura estática e desbalanceada. A EGNAT (URIBE *et al.*, 2006) foi proposta como uma extensão da GNAT para possibilitar atualizações. Porém, a estrutura ainda possui o problema de ser desbalanceada, e devido ao Diagrama de Voronoi ser sensível a atualizações em seus representantes, a estrutura tende a deteriorar conforme atualizações que afetam os representantes são realizadas.

Estratégias baseadas em *raio de cobertura* sofrem com o problema de sobreposição de nós, e estratégias baseadas em Diagrama de Voronoi, são computacionalmente custosas e em sua grande maioria são estáticas, desbalanceadas. Neste trabalho, focou-se em combinar a estrutura baseada em *raio de cobertura* com uma heurística de redução de sobreposição baseada no Diagrama de Voronoi. A contribuição desta proposta é detalhada a seguir.

1.3 Contribuição

Para acelerar as consultas por similaridade por meio da redução da sobreposição de nós em estruturas hierárquicas, foi proposto o método VD-Tree (Voronoi Diagram Tree). Trata-se de um método de acesso métrico que combina a estrutura em árvore baseada em *raio de cobertura*, que é rápida de construir e dinâmica, associada com a heurística do Diagrama de Voronoi, que por sua vez organiza os elementos rigidamente, para reduzir a sobreposição entre nós. Para reduzir a sobreposição entre nós, a VD-Tree realiza reorganizações dos elementos utilizando a heurística do Diagrama de Voronoi. Manter a heurística do Diagrama de Voronoi para toda a árvore é uma operação computacionalmente custosa, e por isso quando um nó precisa ser reorganizado a VD-Tree restringe as reorganizações considerando apenas os nós do nível seguinte da subárvore e o próprio nó. Outra característica da reorganização é que elas são realizadas apenas quando há uma quebra de nós, ou quando há uma troca nos representantes em um nó diretório.

A principal contribuição deste trabalho é a técnica VD-Tree, que combina uma estrutura flexível com a heurística rígida do Diagrama de Voronoi para reduzir a sobreposição entre nós, mantendo uma estrutura dinâmica e balanceada, sem necessitar de armazenar informações extras na estrutura. Uma análise experimental é apresentada, demonstrando que a VD-Tree reduz a sobreposição de nós ao ser comparada com a Slim-Tree e acelera consultas por similaridade com custo de criação apenas um pouco maior.

1.4 Organização do trabalho

Esta dissertação está organizada da seguinte maneira:

- No [Capítulo 2](#) é explicado como são realizadas as comparações por similaridade, descreve os operadores de comparação por similaridade e as propriedades do espaço métrico.
- No [Capítulo 3](#) são apresentados os fundamentos básicos sobre MAMs, as diferentes estratégias utilizadas para indexar os dados. Também são apresentados os conceitos sobre o Diagrama de Voronoi e o estado da arte dos principais conceitos apresentados neste capítulo.
- No [Capítulo 4](#) é apresentado o VD-Tree, um MAM que tem como objetivo reduzir a sobreposição entre nós. No capítulo são apresentados seu funcionamento e principais características.
- No [Capítulo 5](#) é apresentada a análise experimental, descrevendo os materiais e parâmetros utilizados e as métricas coletadas do método proposto, comparando-o com abordagens existentes.
- No [Capítulo 6](#) é apresentada a conclusão sobre este trabalho, bem como descrita a publicação resultante da técnica proposta e possibilidades de trabalhos futuros.

CONSULTAS POR SIMILARIDADE

2.1 Considerações iniciais

O modelo relacional foi inicialmente desenvolvido para considerar valores simples, conhecidos como dados escalares, como datas, números e pequenas cadeias de caracteres. Com a evolução da tecnologia, novos tipos de dados surgiram, tais como vídeos, séries temporais e imagens, também conhecidos como dados complexos. Diferentemente dos dados escalares, os dados complexos não podem ser comparados pela Relação de Ordem, e apesar de ser possível utilizar da Relação de Identidade para comparar os objetos, não é a abordagem mais adequada para dados complexos. A comparação de dados complexos é comumente realizada por meio da similaridade, sendo geralmente utilizada uma representação do objeto obtida por meio de um método de extração de características, e definida uma função de distância para a comparação dos objetos. Os operadores de busca mais comuns para a similaridade são (i) o operador de consultas por abrangência (*Range*) e (ii) o operador de consulta por k vizinhos mais próximos (k -NN).

Este capítulo apresenta os conceitos fundamentais relacionados ao presente trabalho de Mestrado, e está organizado da seguinte forma: Na [Seção 2.2](#) são apresentadas as particularidades da recuperação e comparação de dados complexos. Na [Seção 2.3](#) são apresentados os conceitos básicos sobre métodos de extração de características e exemplos. Na [Seção 2.4](#) são apresentados os principais conceitos das funções de distância e o conceito de descritor. Na [Seção 2.5](#) são apresentados os operadores básicos para buscas por similaridade e variações desses operadores. Na [Seção 2.6](#) são apresentadas as características do espaço métrico e a necessidade do uso de métodos de acesso métricos (MAMs). Por fim, na [Seção 2.7](#) são feitas as considerações finais sobre os conceitos apresentados neste capítulo.

2.2 Recuperação de informação em SGBDs

Hospitais, redes sociais e *e-commerces* geram grandes quantidades de dados complexos como vídeos, áudios e séries temporais. Com a evolução da tecnologia, a quantidade e variedade de dados complexos disponíveis aumenta de maneira acelerada. Com esse aumento, surgiu a necessidade de armazenar, recuperar e extrair conhecimento eficientemente.

Para dados complexos, comparações por Relação de Ordem ($<$, $\leq$, $>$, $\geq$) não são possíveis, e devido a raramente ser interessante procurar por dois dados complexos iguais, a Relação de Identidade ($=$ e $\neq$) não é a opção mais adequada (FALOUTSOS, 1996). Dessa forma, a comparação por similaridade é a mais frequentemente utilizada. Para realizar consultas por similaridade em dados complexos, geralmente são utilizadas de representações do conteúdo dos objetos complexos. Além disso, como a Relação de Identidade e a Relação de Ordem não são adequadas, é necessário também definir uma estratégia de como comparar as representações, sendo necessário definir uma função de distância para tal. É possível resumir as tarefas principais necessárias para realizar consultas por similaridade em três:

- Definir o domínio de dados;
- Definir o aspecto de comparação de interesse;
- Definir como comparar os dados.

A etapa de definir o domínio de dados deve ser a primeira a ser realizada, de modo que seja escolhida uma estratégia adequada para a extração do aspecto de comparação de interesse. Escolhidos os dados que serão trabalhados, a etapa seguinte é a definição do aspecto de comparação de interesse (características), sendo possível utilizar desde o dado completo até as representações obtidas por meio de métodos de extração de características (*Feature Extraction Method* - *FEMs*), como as cores e formas presentes em uma imagem. Por fim, é definido o método utilizado para comparar os valores obtidos do aspecto de comparação de interesse, sendo utilizadas geralmente as funções de distância métricas, para aplicar métodos que possibilitam recuperar as informações de maneira mais eficiente.

2.3 Métodos de extração de características

Para comparações por similaridade em dados complexos, é comum a utilização de representações sucintas dos dados complexos, sendo essas representações obtidas por meio de métodos de extração de características. Os métodos de extração de características comumente extraem características relevantes do dado complexo em vetores de valores numéricos, sendo estes conhecidos como **Vetores de características**. Alguns exemplos de métodos de extração de características, comumente utilizados na literatura, são:

- Imagens: Haralick (HARALICK; SHANMUGAM; DINSTEIN, 1973), *Local Binary Patterns* (OJALA; PIETIKÄINEN; HARWOOD, 1996), *Color Layout* (KASUTANI; YAMADA, 2001), *Color Structure* (KASUTANI; YAMADA, 2001), etc.
- Áudios: *Mel Frequency Cepstral Coefficients* (LOGAN, 2000), *Beat Histogram* (GRUHNE; DITTMAR; GÄRTNER, 2009), etc.

Um dos métodos de extração de características mais utilizados para imagens é o histograma de cores (HAFNER *et al.*, 1995), ilustrado na Figura 1. O histograma de cores descreve a distribuição de cores de uma determinada região de interesse, com a vantagem de ser invariante à rotação e translação de um objeto, porém com a desvantagem de trazer poucas informações sobre a semântica da imagem. O modelo mais utilizado para representar as cores é o RGB (*Red-Green-Blue*), porém existem diversos outros como HSI (*Hue-Saturation-Intensity*), HSV (*Hue-Saturation-Value*) e HSL (*Hue-Saturation-Lightness*) (SHAH; DESAI, 2017), que podem ser interessantes dependendo da aplicação. A construção do histograma de cores, considerando o modelo de representação de cores RGB, é realizada analisando cada *pixel* da região de interesse e contabilizando os 3 canais de cores que compõe cada *pixel*. O método mais simples para armazenar esse valores é em três vetores com 255 posições cada, que representam as contagem de vermelho (R), verde (G) e azul (B), sendo cada uma das posições denominadas de *bins*. É possível também armazenar os valores agrupando cores similares em faixas.

Figura 1 – Exemplo de histograma de cor de uma imagem. Extraído utilizando o software Fiji.

Fonte: Elaborada pelo autor.

É possível que mais de um objeto complexo possua os mesmos valores nos vetores de características. Como exemplo, as imagens da Figura 2 possuem os mesmos valores ao gerar um histograma de cores. Para lidar com esse problema, além da possibilidade de escolher outra característica da imagem, é possível combinar vetores de características obtidos de outros métodos

de extração de características para descrever a imagem com maior precisão (POPOVA; NESHOV, 2013; SMEULDERS *et al.*, 2000).

Figura 2 – Exemplo de imagens com histogramas de cores iguais.

Fonte: Elaborada pelo autor.

2.4 Funções de distância

Obtidos os vetores com características relevantes para representar os dados complexos, é necessário definir como os vetores serão comparados. Para consultas por similaridade, é comum utilizar uma função de distância para a comparação dos vetores de características. É possível realizar uma associação da similaridade entre objetos complexos com a distância entre objetos do mundo real, em que a similaridade é inversamente proporcional à distância entre dois objetos, ou seja, quanto menos distantes, mais similares são dois objetos (TRAINA-JR *et al.*, 2019).

Comparações por similaridade são normalmente mais custosas que comparações por Relação de Ordem ou Relação de Identidade, portanto é importante que as funções de distância utilizadas atendam as propriedades do espaço métrico para possibilitar o uso de métodos de acesso métricos (ZEZULA *et al.*, 2006). Considere-se os vetores de características $x = (V_{x1}, \dots, V_{xn})$ e $y = (V_{y1}, \dots, V_{yn})$, em que V_{y1} é a primeira característica de um vetor de características e n é o número de características (ou dimensionalidade) de um vetor de características. Alguns exemplos de funções de distância que atendem as propriedades do espaço métrico são:

- Euclidiana: $\sqrt{\sum_{i=1}^n (v_{xi} - v_{yi})^2}$
- Manhattan: $\sum_{i=1}^n |v_{xi} - v_{yi}|$
- Jaccard: $\frac{|x \cap y|}{|x \cup y|}$

Outra função de distância comumente utilizada é a *Ledit* (*Levenshtein distance*), também conhecida como distância de edição (*edit distance*). Ela mede a similaridade entre duas cadeias de caracteres (*strings*) por meio da contagem da quantidade mínima de operações atômicas (inserção, remoção ou troca de letra) para transformar uma cadeia de caracteres (*string*) em outra (YU *et al.*, 2016). Considerando uma *string* St , n como o número de caracteres em uma *string* e St_i sendo um caractere na posição i de St , as operações atômicas são definidas como:

- Inserir um caractere c_1 na *string* St na posição i : $inserir(St, i, c_1) = St_1St_2 \dots St_i c_1 St_{i+1} \dots St_n$.
- Deletar um caractere na *string* St na posição i : $deletar(St, i) = St_1St_2 \dots St_{i-1}St_{i+1} \dots St_n$.
- Substituir um caractere c_1 na *string* St na posição i : $substituir(St, i, c_1) = St_1St_2 \dots St_{i-1} c_1 St_{i+1} \dots St_n$.

Definido o método de extração de características e a função de distância, tem-se o descritor utilizado para comparar os dados complexos. Comparações por similaridade utilizam um descritor, que pode ser definido como $d = \langle \text{método de extração de características, função de distância} \rangle$ definido para o domínio dos dados complexos (TORRES *et al.*, 2009).

Comparações de dados complexos utilizando descritores consistem em extrair as características dos dados complexos utilizando o método de extração de características definido pelo descritor, e comparar os vetores de características obtidos utilizando da função de distância, gerando um **espaço de distâncias**. Buscas por similaridade em dados complexos são realizadas no espaço de distância gerado pelo descritor, sendo comumente utilizados MAMs para acelerar o processo de busca.

Para realizar buscas por similaridade em dados complexos, os operadores mais comumente utilizados são o operador de seleção por abrangência e o operador de seleção dos k vizinhos mais próximos, descritos a seguir.

2.5 Operadores de seleção

O operador de seleção pode ser definido como $\sigma_{(y)}T$, em que T é uma relação e y é uma condição. Portanto, o operador de seleção retorna um subconjunto de T que satisfaça a condição y . Para executar o operador de seleção é necessário um operador de comparação, que é responsável por determinar se a condição y foi satisfeita ao comparar dois objetos. As duas consultas por similaridade mais comumente utilizadas são a consultas por abrangência (*Range query*), e as consultas por k vizinhos mais próximos (*k Nearest Neighbors*).

Consultas por similaridade utilizam um objeto associado ao operador de comparação chamado de **centro de consulta**. Para consultas por abrangência, o centro de consulta é utilizado para recuperar os objetos que estão com distância menor ou igual a um valor de raio. Para consultas por k -NN, são retornados os k elementos mais próximos do centro de consulta.

Figura 3 – Exemplo de consulta por abrangência

Fonte: Elaborada pelo autor.

É importante observar que as consultas por similaridade são operações **determinísticas**, ou seja, se a base de dados não mudar, as consultas sempre apresentam o mesmo resultado. Outra observação importante é que, apesar da similaridade passar a ideia de ser algo aproximado, a comparação por similaridade é exata.

2.5.1 Consultas por abrangência (Range)

Em consultas por abrangência, o operador de seleção é representado como $\sigma_{(Rng(d,\xi),s_q)}T$, em que d é o descritor, T é uma relação, ξ é o raio de busca e s_q é um objeto utilizado como centro de consulta. Consultas por abrangência retornam um subconjunto de T , em que a distância de s_q para todos os elementos retornados é menor ou igual ao ξ .

O operador de comparação por abrangência é representado como $s_qRng(d,\xi)s_j$, em que s_q e s_j são os objetos sendo comparados, d é o descritor sendo utilizado para comparar os dados e ξ é o raio de busca. O operador de comparação avalia cada objeto de T utilizando o descritor d , e retorna **TRUE** apenas se $\delta(s_q, s_j) \leq \xi$. É importante observar que, a cada objeto avaliado, se sua distância é menor ou igual o ξ , ele é adicionado à resposta.

A Figura 3 ilustra um exemplo de consulta por abrangência em um espaço euclidiano bi-dimensional, em que a estrela é o centro de consulta, a seta vermelha é raio de busca e os círculos em verde fazem parte da resposta. A Figura 4 ilustra uma consulta pelas cidades que estão a no máximo 50 km de São Carlos, em que as cidades que estão dentro do círculo fazem parte da resposta.

2.5.2 Consultas por k vizinhos mais próximos (k -NN)

O operador de seleção para consultas por k -NN pode ser representado como $\sigma_{(k-NN(d,k),s_q)}T$, que de modo semelhante é aplicado sobre uma relação T , utiliza um objeto s_q como centro de

Figura 4 – Exemplo de consulta por abrangência pelas cidades que estão a no máximo 50km de São Carlos

Fonte: Elaborada pelo autor.

Figura 5 – Exemplo de consulta k vizinhos mais próximos

Fonte: Elaborada pelo autor.

consulta e utiliza de um descritor d , porém utiliza um valor k para indicar quantos elementos mais próximos devem ser retornados, em vez de utilizar um raio de busca ξ . Outra diferença importante é que, diferente da consulta por abrangência, não é possível adicionar objetos à resposta antes de avaliar todos os objetos, devido à necessidade de que a resposta apenas pode conter os k elementos mais próximos.

O operador de comparação por k -NN é representado como $s_qk - NN(d, k)s_j$, em que s_q e s_j são os objetos sendo comparados, d é o descritor utilizado para avaliar a similaridade entre os objetos e k é a quantidade de elementos mais próximos do centro de consulta que devem ser retornados. Esse operador retorna **TRUE** apenas se o objeto s_j é um dos objetos mais próximos de s_q .

A Figura 5 ilustra uma consulta por k -NN, em que a estrela é o centro de consulta, e os círculos em verde são os k vizinhos mais próximos da estrela.

2.5.3 Extensões dos operadores

Além dos operadores de k -NN e Range, existem extensões tais como a busca pelos elementos mais distantes, em vez dos mais próximos, para k -NN. Nesse caso, busca-se pelos k elementos mais distantes do centro de consulta. Para consultas por abrangência, são buscados os elementos que atendem a condição $\delta(s_q, s_j) > \xi$, ou seja, que estão fora do raio de busca (TRAINA-JR *et al.*, 2019). Outros exemplos de extensões são a *Reverse k Nearest Neighbor* que, a partir de um centro de consulta s_q , retorna todos os elementos que possuem s_q como um dos k vizinhos mais próximos (TAO; PAPADIAS; LIAN, 2004), e a *Reverse k Farthest Neighbor*, que retorna os elementos que possuem s_q como um dos k vizinhos mais distantes (TRAN; TANIAR; SAFAR, 2009).

2.6 Espaço métrico

Um Espaço Métrico $\mathbb{M}$ pode ser definido como um par $\mathbb{M} = \langle \mathbb{S}, \delta \rangle$, em que $\mathbb{S}$ é um domínio de dados e δ é uma função de distância que atende as seguintes propriedades para cada elemento $s_i, s_j, s_k \in \mathbb{S}$:

- **Não negatividade:** $\delta(s_i, s_j) \geq 0$
- **Identidade dos indiscerníveis:** $\delta(s_i, s_j) = 0 \Rightarrow s_i = s_j$
- **Simetria:** $\delta(s_i, s_j) = \delta(s_j, s_i)$
- **Desigualdade triangular:** $\delta(s_i, s_j) \leq \delta(s_i, s_k) + \delta(s_k, s_j)$

É importante observar que um espaço métrico possui limitações a quais informações podem ser utilizadas, sendo possível apenas utilizar os elementos inteiros (não existe “parte” dos elementos, por exemplo utilizando apenas a longitude de uma coordenada geográfica), a distância entre os elementos e a contagem dos elementos, dificultando assim a otimização dos algoritmos de busca. Devido às limitações e necessidade de medir as distâncias por meio das funções de distância, que são operações custosas, a busca por similaridade é tipicamente mais custosa que as outras buscas (ZEZULA *et al.*, 2006). Portanto, é necessário que sejam utilizadas funções de distância que atendam as propriedades do espaço métrico, possibilitando o uso de métodos de acesso métricos para acelerar as buscas por similaridade.

2.7 Considerações finais

Neste capítulo foram apresentados os conceitos fundamentais para realizar comparações entre dados complexos e buscas por similaridade. Foi apresentado como as representações dos dados complexos são extraídas utilizando métodos de extração de características, as funções de distâncias que medem a similaridade entre as representações extraídas, e os dois principais operadores de comparação por similaridade, assim como variações dos operadores. Foram

apresentadas, também, as propriedades do espaço métrico, sendo assim importante que a função de distância escolhida atenda a essas propriedades e possibilitem o uso de MAMs.

Comparações por similaridade comumente exigem a extração de características e uma função de distância, o que consequentemente aumenta o custo das buscas por similaridade. Desta maneira, para grandes quantidades são necessárias estratégias de indexação para acelerar as buscas por similaridade, o que é foco desta dissertação de Mestrado. Neste capítulo foram apresentados os conceitos fundamentais para o embasamento deste trabalho, que é melhorar estratégias de indexação métricas. No próximo capítulo são apresentados os principais conceitos sobre Métodos de Acesso Métricos (MAMs), bem como suas vantagens e deficiências, fundamentando assim a técnica desenvolvida neste trabalho de Mestrado.

MÉTODOS DE ACESSO MÉTRICO

3.1 Considerações iniciais

Consultas por similaridade são geralmente mais custosas que consultas utilizando as relação de identidade (RI) e ordem (RO), e por isso é essencial o uso de estruturas de indexação para acelerar as consultas realizadas sobre grandes conjuntos de dados. De maneira semelhante às consultas, estruturas de indexação desenvolvidas para consultas baseadas em Relação de Identidade e Relação de Ordem não são adequadas para dados complexos, então é necessário que novas estruturas sejam desenvolvidas. Para consultas por similaridade, são utilizados os Métodos de Acesso Métricos (MAMs), que tiram vantagem das propriedades do espaço métrico para realizar podas no espaço de busca. Para o uso de MAMs é essencial que a função de distância atenda as propriedades do espaço métrico.

Diversas estratégias vêm sendo desenvolvidas, sendo alguns exemplos de estratégias a pré-computação de distâncias, estruturas *hash* e estruturas hierárquicas. As estruturas hierárquicas são uma das mais comumente utilizadas. Construídas em formato de árvore utilizando *raios de cobertura*, estruturas hierárquicas possuem como um dos principais problemas a sobreposição de nós (KOUAHLA *et al.*, 2019). A sobreposição de nós faz com que seja necessário explorar mais nós para garantir que a resposta esteja exata, o que aumenta o custo das consultas por similaridade,

Neste capítulo, primeiramente são apresentadas as características gerais sobre os métodos de acesso métricos na [Seção 3.2](#) e características de estruturas hierárquicas. Em seguida, na [Subseção 3.2.1](#) são apresentados os MAMs hierárquicos baseados em *raio de cobertura*, os trabalhos relacionados que utilizam essa estratégia, e é descrito o principal problema que este trabalho de Mestrado abordou, que é a sobreposição de nós. Em seguida, na [Subseção 3.2.2](#) são apresentados o conceito de Diagrama de Voronoi, técnicas que utilizam essa estratégia e suas principais características. São apresentadas, também, estratégias que combinam *raios*

de cobertura com o Diagrama de Voronoi ou hiperplanos na [Subseção 3.2.3](#) e suas principais características. Por fim, são apresentadas as considerações finais sobre os conceitos apresentados neste capítulo na [Seção 3.3](#).

3.2 Métodos de Acesso Métricos (MAMs)

Diversas estruturas de índice vêm sendo desenvolvidas utilizando estratégias como *hash*, pré-computação de distâncias e estruturas hierárquicas. Estratégias baseadas em *hash* utilizam uma função *hash* para prover o acesso direto a região de busca, removendo a necessidade de navegar pela estrutura ([ZEZULA et al., 2006](#)). Estratégias baseadas em pré-computação de distâncias consistem em calcular e armazenar as distâncias entre os objetos, possibilitando consultas rápidas ao custo de uma construção ([HANYF; SILKAN, 2020](#)). Estruturas hierárquicas organizam as regiões selecionando recursivamente *representantes* para cada região, construindo assim a hierarquia da estrutura.

Duas características importantes sobre os MAMs são:

- A estrutura é estática ou dinâmica.
- A estrutura foi desenvolvida para disco ou memória principal.

Estruturas dinâmicas são estruturas que permitem operações de atualização, como remoções e inserções, após sua construção. Ao contrário das dinâmicas, estruturas estáticas não permitem atualizações na estrutura após serem construídas, sendo comumente reconstruídas caso o conjunto de dados seja alterado. Inicialmente, estruturas de indexação foram desenvolvidas apenas para memória principal. Estruturas desenvolvidas para memória principal possuem como características serem utilizadas para pequenos conjuntos de dados, comumente estáticas e são rápidas de construir e realizar consultas ([POLA; TRAINA-JR; TRAINA, 2014](#)). Estruturas desenvolvidas para disco são comumente desenvolvidas para grandes conjuntos de dados, são em grande maioria dinâmicas e possuem mecanismos para o acesso e recuperação otimizados do disco ([ZEZULA et al., 2006](#)). As estruturas baseadas na estratégia hierárquica são uma das mais comumente utilizadas ([VIEIRA et al., 2010](#)).

Além das características comuns de MAMs, elas possuem outra característica importante, que é se a estrutura é construída *top-down* ou *bottom-up*. Estruturas construídas *bottom-up* apenas aumentam de nível quando ocorre um split no nó mais alto da árvore. Estruturas construídas *top-down* aumentam de nível sempre que um nó é adicionado na folha de nível mais baixo.

A maneira mais comum de construir uma estrutura hierárquica é utilizando a estratégia em árvore que é representada na [Figura 6](#). Cada região é denominada de nó, com o nó mais alto na árvore sendo o nó raiz, e os nós mais baixos na árvore sendo os nós folhas. Uma das maneiras de representar os nós, para facilitar a explicação, é considerando os nós que possuem o mesmo

Figura 6 – Estrutura em árvore.

Fonte: Elaborada pelo autor.

nó como raiz de sua sub-árvore como nós filhos dessa raiz, e o nó que é a raiz dessa sub-árvore é o nó pai.

Diversas estratégias podem ser aplicadas para definir as regiões dos nós, sendo alguns exemplos os *raios de cobertura*, hiperplanos e estratégias híbridas. Existem também trabalhos que combinam estratégias hierárquicas com outras estratégias como as baseadas em *hash* (DOHNAL *et al.*, 2003; ZHOU *et al.*, 2005) para particionar o espaço. A Tabela 1 sumariza as principais estratégias utilizadas para definir as regiões dos nós, com suas principais características, problemas, e os trabalhos que utilizam essas estratégias para indexar os dados.

Na Tabela 2 estão sumarizadas as principais características dos trabalhos relacionados. As estruturas que possuem as características focadas neste trabalho de Mestrado estão marcadas com um *checkmark* verde, e as que não possuem estão em branco. A última linha contém as características da proposta, o VD-Tree. O VD-Tree constrói uma árvore balanceada e dinâmica, com uma técnica para reduzir a sobreposição entre nós utilizando a estratégia do Diagrama de Voronoi e sem requerer armazenar informações extras, sendo esta uma propriedade importante, pois não aumenta o tamanho do espaço de memória ocupado.

3.2.1 Estratégias baseadas em raio de cobertura

Uma das estratégias para determinar os limites das regiões é utilizando *raios de cobertura*. Para indicar se um elemento está associado a uma região, o elemento deve possuir uma distância para o *representante* da região, sendo menor ou igual ao *raio de cobertura* do nó. Em consultas utilizando estruturas hierárquicas, para garantir que a resposta esteja exata, é necessário explorar todas as sub-árvores que se qualificam. Em estratégias baseadas em *raios de cobertura*, são todas as sub-árvores que possuem *raios de cobertura* que cobrem a região de busca.

Inicialmente, foram desenvolvidas apenas estratégias estáticas e baseadas em memória principal. O VP-tree é um exemplo de estrutura hierárquica estática, que constrói uma árvore balanceada baseando-se em *representantes*, chamados de *vantage points*, e *raios de cobertura* (YIANILOS, 1993). O VP-tree divide um conjunto de elementos em dois subconjuntos, consi-

Tabela 1 – Tabela de estratégias de estruturas.

Estratégias	Características	Problemas	Soluções
<i>Raio de cobertura</i>	Utilizam <i>raios de cobertura</i> para definir as regiões dos nós. Flexível na escolha de onde armazenar os nós, o que possibilita o uso eficiente de memória por meio do controle da ocupação dos nós.	Sobreposição de nós aumenta o custo das consultas.	Estratégias baseadas em reorganizar elementos para reduzir a sobreposição (Slim-Tree). Estratégias que adicionam informações na estruturas para melhorar a capacidade de filtragem durante as buscas (B+-tree, BM+-tree).
Diagrama de Voronoi	Possui regras rígidas quanto à distribuição dos elementos nos nós. Livre de sobreposição. Sensível à troca de representantes.	Comumente cria estruturas estáticas e desbalanceadas devido às regras rígidas. Possui alto custo computacional para ser criada. Estruturas degeneram com atualizações que afetam representantes.	Estratégias baseadas puramente em Diagrama de Voronoi (GNAT). Estratégias que possibilitam atualizações (EGNAT).
<i>Raio de cobertura</i> , hiperplanos e Diagrama de Voronoi	Estratégias comumente dinâmicas e baseadas em memória. Combinam <i>raio de cobertura</i> com a estratégia do Diagrama de Diagrama de Voronoi para criar as regiões.	Essa estratégia sofre com o problema de ter regiões muito amplas.	Estratégia para melhorar a escolha de pivôs e o particionamento das regiões (NOBH-tree, IM-tree, XM-tree).

Fonte: Elaborada pelo autor.

Tabela 2 – Tabela com os trabalhos relacionados e as principais características.

Abordagem	Permite Atualizações	Árvore Balanceada	Reduz Sobreposição	Não Requer Informações Extras	Voronoi
Slim-Tree	✓	✓	✓	✓	
M-Tree	✓	✓	✓		
BM+-tree	✓	✓	✓		
GNAT				✓	✓
EGNAT	✓				✓
NOBH	✓			✓	✓
Proposta: VD-Tree	✓	✓	✓	✓	✓

Fonte: Elaborada pelo autor.

derando a distância dos elementos para o *vantage point*, e recursivamente aplica essa divisão para construir a árvore (ZEZULA *et al.*, 2006). Para dividir os subconjuntos, são calculadas as distâncias dos elementos para o *vantage point*, e é encontrada a mediana das distâncias. Como resultado, tem-se a divisão entre um subconjunto com os elementos com distância menor ou igual a mediana, e outro com os elementos cuja a distância é maior que a mediana. Outra estrutura estática, mas baseada em disco, é o MVP-tree (BOZKAYA; ÖZSOYOGLU, 1999). O MVP propôs duas extensões para o VP-tree, sendo elas: (i) o uso de múltiplos *raios de cobertura* centrados nos *vantage points* para aumentar o tamanho dos nós para adaptar a estrutura para o uso em disco; e (ii) o uso de dois *vantage points* em vez de apenas um, permitindo assim que a árvore possua múltiplas sub-árvores, consequentemente diminuindo sua altura.

A primeira estratégia dinâmica baseada em disco é o M-Tree (CIACCIA; PATELLA; ZEZULA, 1997), que utiliza de *raios de cobertura* para determinar o limite dos nós. Os *raios de cobertura* são definidos como a distância do *representante* até o elemento mais distante do nó. A estrutura do M-Tree é composta por dois tipos de nós, sendo eles os nós folhas (que armazenam os elementos), e os nós internos (que armazenam ponteiros para outros nós da árvore). Em cada nó folha, são armazenados os elementos, um identificador para o objeto e a distância para o elemento *representante*. Em cada entrada dos nós diretório, são armazenados o *raio de cobertura*, o ponteiro para o nó pai, a distância para o *representante* e o elemento *representante* do nível inferior.

A construção de um M-Tree é iniciada com a criação de um nó e a inserção dos elementos nesse nó. Quando o nó está cheio, é chamada uma função de quebra de nós, que é dividida em duas etapas principais: a escolha dos novos *representantes* e a distribuição de elementos. O M-Tree é uma estrutura *bottom-up*, então quando a quebra de nós ocorre na raiz, a árvore aumenta um nível. O M-Tree propôs as seguintes estratégias para a escolha de *representantes*:

- *m_RAD*: Considera todos os pares de nós e distribuições de elementos, e escolhe o que a soma dos raios resulta no menor valor possível.
- *mM_RAD*: Semelhante ao *m_RAD*, minimiza o máximo dos dois raios.
- *M_LB_DIST*: Considera o *representante* antigo como um dos novos *representantes* e seleciona o elementos mais distantes do *representante* antigo para ser o outro novo *representante*.
- *random*: Seleciona aleatoriamente os novos *representantes*.
- *sampling*: Seleciona aleatoriamente elementos em um subconjunto. Cada par de elementos desse subconjunto é selecionado como possível representante e é testada a distribuição dos elementos, sendo selecionado o par que minimiza os *raios de cobertura*.

Para a etapa da distribuição dos elementos foram propostas duas estratégias, sendo a principal distribuir os elementos baseando nas distâncias para os *representantes*, sendo cada elemento inserido no nó do *representante* mais próximo. Outra estratégia proposta consiste em

calcular e armazenar todas as distâncias dos elementos para os dois *representantes*. Considerando O_{p1} como um dos *representantes*, O_{p2} como o outro *representante* e N como a lista de elementos, a estratégia inicia inserindo o vizinho mais próximo de O_{p1} em seu nó e o removendo da lista N . Em seguida, é inserido o vizinho mais próximo de O_{p2} no nó que tem O_{p2} como *representante*, e o elemento é removido da lista também (CIACCIA; PATELLA; ZEZULA, 1997). Esse processo é realizado até que N esteja vazia, sendo esta estratégia chamada de *balanced*.

Com a primeira quebra de nós, as possibilidades de onde novos elementos podem ser inseridos na estrutura aumenta, fazendo com que seja necessário definir uma estratégia para escolher onde é o nó mais adequado para inserção do novo elemento. Para a escolha de onde o novo elemento deve ser inserido, é necessário definir uma política de escolha de nó. Apesar de buscas por igualdade possuírem pouca semântica em análises exploratórias em dados complexos, ela pode ser utilizada em MAMs como uma política de escolha de nós para garantir que não há elementos duplicados na estrutura. A estratégia proposta pelo M-Tree consiste em verificar qual sub-árvore cobre o elemento a ser inserido, e caso o elemento seja coberto por mais de uma sub-árvore, é escolhida aquela em que o *representante* possui a menor distância para o elemento. Se nenhuma sub-árvore cobre o novo elemento, é escolhida a sub-árvore que minimiza o aumento do raio necessário para cobrir o elemento. Após descer até o nível dos nós folhas, três situações podem ocorrer na escolha do nó onde o elemento deve ser inserido: (i) apenas um nó folha cobre o novo elemento; (ii) mais de um nó folha cobre o elemento; e (iii) nenhum nó folha cobre o novo elemento. As três situações são tratadas seguindo a mesma política de escolha de nós utilizada para chegar até os nós folhas, com a diferença de que o elemento é inserido no nó escolhido, em vez de descer para o outro nó.

Uma característica importante do M-Tree é a de que os elementos estão dentro do nó, apenas se estão cobertos pelo *raio de cobertura* do nó, porém nem todo elemento coberto está no nó. Isso ocorre devido à possibilidade dos *raios de cobertura* estarem sobrepostos.

A sobreposição de nós é um dos maiores problemas em estruturas hierárquicas baseadas em raio de cobertura. A Figura 7 (a) ilustra um exemplo de como uma consulta sofre com o problema da sobreposição. Quando as buscas são realizadas, para garantir que a resposta esteja exata, é necessário explorar todas as sub-árvores que cobrem o centro de busca. A Figura 7 (b) ilustra esse caso, em que é necessário descer em duas subárvores. Quanto mais sub-árvores precisam ser exploradas, maior é o número necessário de cálculos de distância e acessos a disco, aumentando o custo de responder as consultas por similaridade.

Para tratar o problema da sobreposição, diversas estratégias foram propostas, como a redução da sobreposição entre nós (TRAINA-JR *et al.*, 2002a), a melhora da capacidade de filtrar nós durante as buscas (ZHOU *et al.*, 2003; SKOPAL; POKORNÝ; SNÁSEL, 2004; ZHOU *et al.*, 2005) e o uso de outras estratégias de particionamento de espaço que não sofrem com a sobreposição de nós (BRIN, 1995; URIBE *et al.*, 2006; POLA; TRAINA-JR; TRAINA, 2014).

Para tratar o problema da sobreposição existente no M-Tree, foi proposto o Slim-Tree

Figura 7 – Exemplo de como a sobreposição de nós afeta a busca.

Fonte: Elaborada pelo autor.

(TRAINA-JR *et al.*, 2002a). O Slim-Tree também constrói uma árvore dinâmica, baseada em disco, balanceada mas que utiliza a estratégia de reorganização dos elementos para reduzir os raios de cobertura. Isso faz com que menos nós estejam sobrepostos por meio da redução dos raios de cobertura. A estrutura do Slim-Tree é semelhante à estrutura do M-Tree, sendo os nós folhas iguais, porém os nós diretório armazenam também o número de entradas ocupadas no nó. Foram propostas novas políticas de escolha de nó, sendo elas:

- *random*: O nó é escolhido aleatoriamente.
- *mindist*: É escolhido o nó que possui a menor distância entre o *representante* do nó e o novo elemento.
- *minoccup*: O nó que possui menos elementos é escolhido.

O Slim-Tree também propôs três novas políticas de quebra de nós, sendo elas:

- *random*: Os *representantes* são escolhidos aleatoriamente entre os elementos dentro do nó. Os elementos são inseridos no nó do *representante* mais próximo, exceto se o nó está cheio.
- *minMax*: São considerados todos os pares de elementos como possíveis *representantes*, e para cada possível par é testada a distribuição de elementos. O par escolhido é aquele que minimiza os novos *raios de cobertura*.
- *Minimal Spanning tree MST*: Gera a árvore geradora mínima (*minimal spanning tree*) dos elementos e remove a aresta com a maior distância, gerando dois grupos. Dentro de cada grupo, é escolhido o elemento central como os *representantes*.

Uma das principais contribuições do Slim-Tree é o método Fat-Factor, utilizado para medir o grau de sobreposição de uma árvore. Para calcular o grau de sobreposição da árvore, o

Figura 8 – Exemplo de árvore para o cálculo do Fat-Factor.

Fonte: Elaborada pelo autor.

Fat-Factor propõem que, ao invés de tentar calcular a área da sobreposição, pode-se realizar a contagem dos elementos que estão em sobreposição. Essa estratégia cria uma medida global de quantos elementos poderiam estar em outro nó (TRAINA-JR *et al.*, 2002a). A fórmula para o cálculo do Fat-Factor é definida na Equação 3.1:

$$Fat(L) = \frac{l_c - H_{min} * N}{N} \cdot \frac{1}{M_{min} - H_{min}} \quad (3.1)$$

Na equação, L é uma árvore métrica, N é o número de elementos indexados, H_{min} é a altura mínima da árvore, M_{min} é o número mínimo de nós requeridos para armazenar os dados e l_c é o número de nós que precisam ser acessados para realizar uma consulta pontual para cada elemento na árvore. Essa equação resulta em um valor entre 0 e 1, denominado Fat-Factor. Caso o valor de Fat-Factor seja zero, a árvore não possui elementos em sobreposição, enquanto uma árvore com o Fat-Factor igual a um possui todos os elementos em sobreposição.

A definição de Fat-Factor assume duas hipóteses. Na primeira, são levadas em consideração apenas consultas por abrangência para estimar o quão boa é a árvore métrica (TRAINA-JR *et al.*, 2002a). Assumindo que a consulta por k vizinhos mais próximos pode ser considerada uma consulta por abrangência com o raio igual à distância do elemento mais distante entre os k vizinhos mais próximos, essa hipótese deixa de ser restritiva (BERCHTOLD *et al.*, 1997). Na segunda hipótese é assumido que a distribuição dos centros das consultas por abrangência segue a distribuição dos objetos. Isso é razoável, uma vez que é esperado que as consultas são mais provavelmente feitas em regiões do espaço em que a densidade de objetos é mais alta (TRAINA-JR *et al.*, 2002a).

Um exemplo de uma árvore para o cálculo do Fat-Factor é ilustrado na Figura 8, em que a árvore possui um nível de altura, oito elementos com dois em sobreposição. Devido à necessidade de acessar dois nós nos dois elementos em sobreposição, são necessários dez acessos a disco para responder uma consulta pontual para cada elemento. Considerando esses valores e aplicando na fórmula, como pode ser visto na Equação 3.2, é obtido um Fat-Factor de 0,25.

$$Fat(L) = \frac{10 - 1 * 8}{8} * \frac{1}{2 - 1} = \frac{2}{8} = 0,25 \quad (3.2)$$

Apesar do Fat-Factor ser uma das formas de medir o quão “boa” é a árvore construída, ela apenas considera o grau de sobreposição na árvore. Considerando uma situação em que sejam comparadas duas árvores, em que uma possui um valor de Fat-Factor mais alto utilizando um número menor nós, e outra árvore com um valor de Fat-Factor baixo, porém utilizando mais nós para indexar os dados, o Fat-Factor pode não refletir de maneira adequada em qual das árvores é menos custoso realizar consultas. Apesar de possuir menos sobreposição, consultas na segunda árvore podem exigir mais acessos a disco. Assim, o Slim-Tree também propôs o Bloat-Factor, que modifica o cálculo do Fat-Factor para penalizar o uso de mais nós. O Bloat-Factor possibilita comparar árvores construídas com métodos de escolha de representante e distribuição de elementos diferentes. A fórmula do Bloat-Factor é apresentada na [Equação 3.3](#).

$$rFat(T) = \frac{l_c - H_{min} * N}{N} * \frac{1}{M_{min} - H_{min}} \quad (3.3)$$

Calculada a sobreposição existente na árvore, é possível utilizar outra técnica proposta pelo Slim-Tree, o Slim-Down. Sendo uma das principais contribuições propostas pelos autores do Slim-Tree, a técnica Slim-Down é aplicada em uma árvore já construída para reduzir a sobreposição entre os nós. O Slim-Down encontra em um nó o elemento mais distante do *representante*, e avalia se há outro nó que cobre esse elemento. Caso exista, esse elemento é movido para o outro nó, e o *raio de cobertura* do nó antigo é atualizado. A [Figura 9](#) ilustra uma sub-árvore antes da execução do Slim-Down, em que os elementos em laranja são os representantes, os elementos em cinza são os elementos, as setas indicam a qual representante cada elemento está associado e o elemento em vermelho é o mais distante. A [Figura 10](#) ilustra a árvore após a execução do Slim-Down, em que o elemento mais distante é movido para o nó do canto inferior direito, o *raio de cobertura* em vermelho é o raio antigo e a faixa verde ilustra a redução da *raio de cobertura*. Uma característica importante do Slim-Down é que a técnica não garante a ocupação mínima dos nós, sendo possível que algum nó fique vazio após a realização da operação ([TRAINA-JR et al., 2002a](#)).

Outra estratégia para lidar com o problema da sobreposição é por meio da melhoria da capacidade de filtragem dos nós durante as buscas, em vez de reduzir a sobreposição entre nós. Essa estratégia consiste em adicionar informações extras na estrutura, aumentando o tamanho da estrutura de indexação, mas possibilitando podas que utilizam essas informações.

Uma dos primeiros MAMs que utiliza essa estratégia, foi o M+-tree ([ZHOU et al., 2003](#)), que é uma extensão do M-Tree combinado com um MVP-tree. O M+-tree constrói *bottom-up* uma árvore dinâmica, balanceada e baseada em disco e foca em melhorar o particionamento binário do MVP-tree com a estrutura do M-Tree ([ZHOU et al., 2003](#)). Um conceito importante

Figura 9 – Exemplo de uma sub-árvore antes de realizar o Slim-Down.

Fonte: Elaborada pelo autor.

Figura 10 – Exemplo de uma sub-árvore após a execução do Slim-Down.

Fonte: Elaborada pelo autor.

para o M+-tree é a *key dimension*, sendo definida como a dimensão que mais afeta o cálculo de distância. A *key dimension* é utilizada para lidar com a sobreposição ao eliminar sub-árvores que não são necessárias explorar para obter a resposta. Para definir se a sub-árvore pode ser filtrada utilizando a *key dimension*, é utilizada a fórmula a seguir. Considerando $a = (V_{a1}, V_{a2}, \dots, V_{an})$ e $b = (V_{b1}, V_{b2}, \dots, V_{bn})$ como dois objetos com n dimensões, key sendo a *key dimension* e r como o raio de busca, então $|V_{akey} - V_{bkey}| \leq \delta(a, b)$, se $\delta(a, b) \leq r$ então $|V_{akey} - V_{bkey}| \leq r$ (ZHOU *et al.*, 2003). Com a *key dimension*, o M+-tree particiona o nó em dois subespaços, chamados de *twin nodes*, que auxiliam em melhorar o espalhamento dos dados em nós e em reduzir a altura das árvores. O M+-tree utiliza estruturas de nós semelhantes aos do M-Tree, com a diferença que o nó diretório armazena também o número da *key dimension*, o ponteiro para ambos os *twin nodes*. A estrutura dos *twin nodes* possibilita reduzir níveis da árvore.

Como uma extensão do M+-tree, surgiu o BM+-tree (ZHOU *et al.*, 2005), uma estrutura dinâmica, baseada em disco e balanceada, propondo o uso da técnica *binary hyperplanes*. Essa técnica aprimora a ideia da *key dimension* do M+-tree para utilizar duas *key dimensions* para particionar os dados. Essa técnica foi proposta a partir da observação de que cancelar a segunda dimensão mais importante no cálculo de distância gera uma grande perda de informação, do mesmo modo que quando o processo de redução de dimensionalidade é aplicado, são escolhidos mais de uma das dimensões mais importantes (ZHOU *et al.*, 2005). Dessa maneira, o BM+-tree utiliza de *binary hyperplanes* para particionar o espaço dentro do nó para gerar as *twin nodes*, sendo o *binary hyperplanes* alterável de acordo com a distribuição dos dados nos subespaços. Essa flexibilidade do posicionamento do hiperplano possibilita melhorar a capacidade de espalhamento dos elementos dentro dos *twin nodes*, e com a melhor organização dos elementos a altura da árvore tende a ser menor do que o M-Tree.

Existem outras estratégias que utilizam de informações extras para lidar com a sobreposição, como o PM-tree (SKOPAL; POKORNÝ; SNÁSEL, 2004). O PM-tree cria pivôs e armazena as distâncias dos elementos para os pivôs para possibilitar filtrar sub-árvores. Apesar de acelerar as consultas, essa estratégia sofre com um aumento no custo e tempo gasto na construção da estrutura e um aumento do espaço utilizado para armazenar e manter a estrutura.

Outra estratégia hierárquica que não sofre com o problema de sobreposição é utilizando hiperplanos para dividir o espaço. Entre as diversas estratégias baseadas em hiperplanos, o Diagrama de Voronoi é uma das mais comumente utilizadas, que pelas suas rígidas regras geram estruturas sem sobreposição, conforme detalhado a seguir.

3.2.2 Diagrama de Voronoi

O Diagrama de Voronoi é uma estratégia para particionar o espaço em regiões sem sobreposição, baseando-se em um conjunto de elementos escolhidos para serem os representantes das regiões, sendo estes conhecidos como *seeds*. Considere-se um conjunto de n regiões $P = (p_1, p_2, \dots, p_n)$, s_i como um elemento no conjunto de dados, s_i^{rep} como a *seed* de uma região p_i e δ como uma função de distância. O Diagrama de Voronoi organiza as regiões de maneira que, para cada elemento s_i em uma partição p_i , o mesmo está mais próximo da sua *seed* s_i^{rep} do que de qualquer outra *seed* de outra região, de modo que $\delta(s_i, s_i^{rep}) < \delta(s_i, s_j^{rep}), \forall j \neq i$.

A Figura 11 ilustra a construção de um Diagrama de Voronoi representado em duas dimensões em um espaço euclidiano. Em (a) são definidas as *seeds* como os pontos em vermelho e os pontos cinzas são os outros elementos. O próximo passo, ilustrado em (b), consiste em calcular as distâncias entre todas *seeds* e identificar a meia-distância entre cada *seed* utilizando hiperplanos. Em (c), cada região é obtida pela intersecção dos hiperplanos, criando regiões em que cada elemento está mais próximo da *seed* da sua região do que de qualquer outra *seed*.

Uma das primeiras estruturas a utilizar puramente o Diagrama de Voronoi, foi a *Geo-*

Figura 11 – Exemplo da construção de um Diagrama de Voronoi.

Fonte: Elaborada pelo autor.

metric Near-neighbor Access Tree GNAT (BRIN, 1995). Essa é uma estrutura que tem como principais características ser estática, desbalanceada e sem o problema da sobreposição. A construção do GNAT pode ser resumida em criar o Diagrama de Voronoi recursivamente, de maneira que o Diagrama de Voronoi é baseado nos representantes do nível inferior. A construção foca em três principais etapas, sendo a primeira escolher um conjunto de representantes. O método de escolha dos representantes inicia considerando aleatoriamente um elemento, e em seguida escolhendo o elemento mais distante do primeiro. O próximo elemento escolhido é o mais distante considerando os dois que já foram selecionados, sendo essas etapas executadas até que seja obtido o número desejado de representantes. Essa estratégia é executada três vezes, obtendo três possíveis candidatos de conjuntos de representantes, para então ser escolhido o conjunto em que os elementos estão mais distantes. A segunda etapa é associar os elementos restantes de acordo com a estratégia do Diagrama de Voronoi, de modo que cada elemento restante esteja no nó do representante mais próximo. Na terceira etapa, para cada representante, é calculada uma tabela de raios, em que cada linha é composta pela menor e maior distâncias do representante para cada conjunto de elementos associado aos outros representantes. Estas etapas são executadas recursivamente para construir a estrutura de árvore. A terceira etapa é utilizada principalmente para realizar podas durante as consultas, devido à possibilidade do raio de busca cobrir mais de uma região em uma consulta por abrangência.

Para possibilitar realizar atualizações no GNAT, foi proposta o EGNAT (PAREDES; NAVARRO, 2009). O EGNAT propôs as modificações na estrutura e a técnica *ghost hyperplanes* para possibilitar atualizações ao custo de inserir sobreposição na estrutura, além de otimizar o uso da estrutura em disco. O EGNAT possui dois tipos de nós: os *buckets* que são as folhas, e os *gnat* que são os nós diretórios. A construção do EGNAT é realizada em *top-down*, sendo explorada a sub-árvore com o representante mais próximo do novo elemento até chegar em um *bucket*, onde o novo elemento é inserido. Os nós são inicialmente criados como *buckets*, e quando o nó enche ele é transformado em um *gnat*, selecionando um conjunto de elementos para

serem os representantes e inserindo o resto dos elementos em novos *buckets*.

Para deletar elementos em *buckets*, basta remover o elemento do *bucket*. Porém, para deletar elementos em nós *gnat*, existe o problema da estrutura ser sensível a troca de representantes. Para isso, foi proposta a técnica *ghost hyperplanes*. Uma situação que pode ocorrer durante a deleção de elementos de um nó *gnat* é a de que deleção de elementos torne possível colocar todos os elementos em um *bucket*, porém um *gnat* não é transformado em um *bucket*. Devido ao custo de transformar um *gnat* em um *bucket* ser considerável, o EGNAT realiza reconstruções das sub-árvores periodicamente para lidar com essa situação. A *ghost hyperplanes* consiste em substituir o representante removido sem alterar a tabela de raios e armazenar a distância do antigo representante para o novo. Como a tabela de raios não é recalculada e o representante muda, o formato de região também muda, sendo necessário manter ambos os hiperplanos que definia o antigo e novo formato da região, introduzindo a possibilidade de sobreposição de regiões na estrutura. Devido à possibilidade de regiões estarem sobrepostas, são realizadas também alterações nas estratégias de inserção e busca de elementos para considerar essa possibilidade.

3.2.3 Raio de cobertura, hiperplanos e Diagrama de Voronoi

Uma estratégia que combina *raios de cobertura* com Diagrama de Voronoi é o NOBH-tree (*Non-Overlapping Balls and Hyperplanes trees* (POLA; TRAINA-JR; TRAINA, 2014)). Os métodos de acesso métricos NOBH-tree formam uma família de estruturas baseadas em memória, dinâmicas que criam regiões sem sobreposição utilizando *raios de cobertura* e a estratégia do Diagrama de Voronoi. Em cada nível, para definir as regiões, são selecionados dois elementos como representantes, e é calculada a distância entre esses representantes sendo esta chamada de *node leap*, a qual é utilizada como valor do raio. Por fim, é utilizado um hiperplano que divide o espaço em duas regiões, em que elementos em cada uma das regiões estão mais próximos do representante da sua região do que do outro representante. Considerando s_1 e s_2 como os representantes escolhidos, a Figura 12 ilustra em um espaço euclidiano as regiões formadas, em que: (a) é a região dentro do *raio de cobertura* de s_1 ; (b) é a região dentro do *raio de cobertura* de s_2 ; (c) é a região fora dos *raios de cobertura* e do lado esquerdo do hiperplano; (d) a região fora dos *raios de cobertura* e do lado direito do hiperplano; (e) é a região em sobreposição à esquerda do hiperplano; e (f) é a região em sobreposição à direita do hiperplano. Cada membro da família do NOBH-tree é composta por variações de quais regiões são utilizadas para indexar os dados. Essa família de MAMs sofre com o problema do alto custo de construir a estrutura e de que inserções na região externa aos raios de cobertura podem deteriorar a estrutura.

Outra estratégia que indexa os dados utilizando regiões é o IM-tree, que constrói *top-down* uma estrutura dinâmica, desbalanceada, que combina *raios de cobertura* com hiperplanos (KOUAHLA; MARTINEZ, 2012). De maneira semelhante ao NOBH-tree, essa é uma estratégia que gera regiões sem sobreposição combinando *raios de cobertura* e hiperplanos, porém ao invés de dividir a região em que os nós estão sobrepostos, o hiperplano apenas divide a região fora

Figura 12 – Ilustração das regiões criadas pelo NOBH-tree em um espaço euclidiano.

Fonte: Adaptada de [Pola, Traina-Jr e Traina \(2014\)](#).

dos *raios de cobertura*, gerando apenas 5 regiões. Essa estrutura, apesar de possuir a vantagem de não ter sobreposição, sofre com inserções nas regiões fora dos raios de cobertura. Com o aumento dos dados, inserções nessas regiões se tornam comuns, o que degenera a estrutura, tornando necessário realizar reconstruções da estrutura.

Para tratar as situações em que há muitos elementos na região externa dos raios de cobertura no IM-tree, foi proposto o XM-tree ([KOUAHLA et al., 2019](#)). O XM-tree gera uma estrutura dinâmica, baseada em memória, desbalanceada e sem sobreposição, e propõe alterações tanto nas formas como as regiões são determinadas quanto na estrutura. Além dos nós folhas e dos nós diretórios, foi proposto o uso de um novo tipo de nó chamado de *extended node*, que consiste em listas encadeadas com os elementos da região externa dos raios de cobertura.

3.3 Considerações finais

Neste capítulo foram apresentadas as diversas abordagens utilizadas para construir MAMs. Como pode ser observado, estruturas baseadas em raio de cobertura apresentam flexibilidade ao inserir elementos, provendo estruturas dinâmicas e com o uso eficiente de memória, o que diminui o custo de construir e manter a estrutura. Porém, a flexibilidade traz o problema da sobreposição, que torna necessário explorar mais nós para garantir que a resposta está exata, aumentando o custo de realizar consultas. Também foram apresentadas estruturas baseadas em regras rígidas, como as do Diagrama de Voronoi, que ao contrário das estruturas baseadas em *raios de cobertura*, possibilitam realizar consultas explorando menos nós, mas que causam aumento no custo para construir a estrutura e maior dificuldade de manter a estrutura.

Apesar de diversos trabalhos serem propostos para tratar o problema da sobreposição, as estratégias existentes focam em melhorar a capacidade de filtrar nós em buscas adicionando informações na árvore, aumentando o custo de criar, manter e armazenar. Trabalhos que focam

em combinar estratégias de particionar o espaço com a flexibilidade das estratégias baseadas em *raios de cobertura*, sofrem com o problema de que inserções em determinadas regiões podem deteriorar a estrutura, de maneira que conforme mais dados são inseridos e mais atualizações de representantes são necessárias, maiores são as chances da estrutura deteriorar.

Buscando balancear o aumento do custo em manter regras rígidas e reduzir a sobreposição de nós sem adicionar informações extras na estrutura, foi proposto o VD-Tree, apresentado no [Capítulo 4](#). Nossa estratégia utiliza como base a estrutura da Slim-Tree, mas a combina com as regras do Diagrama de Voronoi para realizar reorganizações durante sua construção. Como resultado, gera-se uma estrutura que mantém a flexibilidade na alocação dos elementos nos nós proporcionada por uma estratégia baseada em raio de cobertura e ao mesmo tempo reduz a sobreposição dos nós por meio da redução dos *raios de cobertura*.

REDUÇÃO DE SOBREPOSIÇÃO COM O DIAGRAMA DE VORONOI

4.1 Considerações iniciais

Neste trabalho de Mestrado é proposto o método de acesso métrico VD-Tree (Voronoi Diagram Tree), que tem como objetivo aplicar reorganizações durante a construção da árvore para reduzir a sobreposição dos nós e acelerar consultas por similaridade. A estratégia utiliza a ideia do particionamento rígido do Diagrama de Voronoi para melhorar a organização dos elementos nos nós, mantendo a estrutura balanceada e dinâmica por meio do uso de *raios de cobertura*. O VD-Tree utiliza-se de mais nós para obter uma melhor organização, reduzindo o tempo gasto e o número de cálculos de distância realizados ao custo de realizar mais acessos a disco.

Este capítulo está organizado da seguinte forma. Na [Seção 4.2](#) são apresentadas as principais características do método, as escolhas das políticas de escolha e de quebra de nós. Na [Subseção 4.2.1](#) é apresentado o principal componente da técnica, sendo explicado como são realizadas as reorganizações, o algoritmo utilizado para realizá-las, e as principais características da técnica. Na [Subseção 4.2.2](#) são apresentadas as vantagens e desvantagens do método proposto. Por fim, na [Seção 4.3](#) são apresentadas as considerações finais sobre o método.

4.2 VD-Tree

O VD-Tree indexa os dados utilizando *raios de cobertura* e reorganizações baseadas no Diagrama de Voronoi para reduzir a sobreposição entre nós. Desse modo, ele permite manter a estrutura dinâmica e balanceada aproveitando a flexibilidade de posicionar elementos dada pelos *raios de cobertura*, e reduzir a sobreposição entre nós, obtida usando a reorganização dos elementos baseados no Diagrama de Voronoi.

Figura 13 – Fluxograma da inserção de elementos no VD-Tree.

Fonte: Elaborada pelo autor.

Na Figura 13 é apresentado um fluxograma que ilustra os passos para a inserção de um novo elemento na estrutura. A inserção é realizada de forma semelhante às árvores Slim-Tree e M-Tree, iniciando em (a) com uma política de escolha de nós. Caso a inserção seja realizada sem o nó estar cheio, são realizadas as atualizações nas informações (número de objetos na sub-árvore e alterações nos raios de cobertura) no resto da árvore, conforme ilustrado em (c). Caso o nó esteja cheio, é realizada a quebra de nós em (b), sendo então aplicada a principal técnica deste trabalho, que é a reorganização dos nós ilustrada em (e). A reorganização pode resultar em 3 possibilidades: a primeira possibilidade é a reorganização ser realizada com sucesso, passando então para (c); a segunda possibilidade é necessidade de realizar uma quebra de nós, voltando para (b); a terceira possibilidade é a necessidade de trocar os representantes dos níveis superiores devido à quebra de nós ilustrada em (d), sendo então necessário realizar novamente a reorganização.

Para a inserção de novos elementos, é utilizada a política de inserção *mindist*, que sempre explora a sub-árvore com o representante mais próximo do elemento sendo inserido, e insere no nó folha que tem o representante mais próximo do novo elemento. A *mindist* permite manter a estratégia de Diagrama de Voronoi por sempre inserir no nó do representante mais próximo.

Para a construção do VD-Tree, é possível utilizar todas as estratégias de quebra de nós presentes na M-Tree e Slim-Tree. Porém, identificou-se que a política de quebra de nós recomendada para construir o VD-Tree é a MST, uma vez que ela permite a escolha de representantes que estão no centro dos grupos, o que auxilia em balancear a distribuição dos elementos na estrutura.

O principal foco deste trabalho de Mestrado é reduzir a sobreposição dos nós da estru-

tura. Para isso, são realizadas reorganizações dos elementos durante a construção da árvore, sendo esta reorganização baseada no Diagrama de Voronoi. A técnica proposta para realizar as reorganizações é denominada VD-fit, sendo este o principal componente da estrutura proposta VD-Tree.

4.2.1 VD-fit

A técnica VD-fit reorganiza os elementos de forma que todos aqueles que estejam em sobreposição sigam a regra do Diagrama de Voronoi, em que eles estão no nó do representante mais próximo. Uma vez que manter Diagrama de Voronoi em toda estrutura é um processo computacionalmente custoso, foi proposto que as reorganizações considerem apenas os elementos que estão em regiões sobrepostas, e os nós afetados são apenas o nó que vai ser reorganizado e os seus filhos diretos do nível seguinte. Outra característica importante da reorganização é que ela é realizada apenas quando há quebra de nós ou troca de representantes. Como o objetivo é balancear o uso do Diagrama de Voronoi com os raios de cobertura, essas características permitem manter a flexibilidade da estrutura, enquanto reduz a sobreposição.

A reorganização dos elementos é realizada conforme ilustrado na [Figura 14](#), sendo este um exemplo em um espaço euclidiano. Em (a) são apresentados os nós que vão ser reorganizados e os representantes. No exemplo, os pontos em vermelho são os representantes, os pontos em cinza são os elementos e as setas indicam quais nós são representados pelos pontos em vermelho. O próximo passo é ilustrado em (b), em que são selecionados todos os elementos que estão em sobreposição e marcados em azul, sendo a área amarela a região que os nós estão sobrepostos e as linhas em azul ilustram a separação pelo Diagrama de Voronoi. Em seguida (c), para cada ponto azul é realizada uma busca para verificar se há um representante mais próximo em outro nó. Nesta etapa, a seta tracejada aponta para o representante mais próximo. Caso possua um representante mais próximo, os pontos azuis são movidos e os raios são atualizados. Em (d) é ilustrado o resultado da reorganização dos elementos, em que é possível notar a redução das áreas sobrepostas.

O [Algoritmo 1](#) apresenta o pseudocódigo correspondente ao método VD-fit, e detalha como é o processo de reorganização realizado. O VD-fit recebe como entrada:

- $\mathcal{N}^{reorg}$: O nó diretório que deve ter seus filhos reorganizados.

Como saída, o algoritmo retorna:

- NULO: caso a reorganização seja concluída sem a necessidade de quebra de nós.
- $\mathcal{N}^{filho}$: O nó filho que deve ser quebrado.

A reorganização inicia recebendo o nó que vai ser reorganizado, e recuperando em uma lista os nós filhos na linha 2. Na linha 3 é realizado um laço para explorar todos os nós filhos,

Figura 14 – Exemplo da reorganização realizada na VD-fit.

Fonte: Adaptada de [Moriyama et al. \(2021\)](#).

Algoritmo 1 – VD-fit: reorganização dos nós da VD-Tree.

Entrada: $\mathcal{N}^{Reorg}$: nó a ser reorganizado

Saída: Os nós filhos com os elementos em sobreposição reorganizados baseando no Diagrama de Voronoi

```

1  início
2  /* Recupera a lista dos nós filhos de  $\mathcal{N}^{Reorg}$  */
3   $\mathbb{N}^{filhos} \leftarrow Filhos(\mathcal{N}^{Reorg});$ 
4  /* Laço para verificar todos os nós filhos de  $\mathcal{N}^{Reorg}$  */
5  para  $\mathcal{N} \in \mathbb{N}^{filhos}$  faça
6       $s^{rep} \leftarrow recupereRepresentante(\mathcal{N});$ 
7      para  $s_i \in \mathcal{N}$  faça
8           $s_{mov} \leftarrow NULL;$ 
9           $dist^{MIN} \leftarrow \infty;$ 
10         para  $s_j \in \mathcal{N}^{Reorg}$  faça
11              $\xi^{cov} \leftarrow recupereRaioCobertura(s_j);$ 
12             /* Validando as regras do Diagrama de Voronoi */
13             se  $\delta(s_i, s_j) < \min(\delta(s_i, s^{rep}), \xi^{cov}, dist^{MIN})$  então
14                 /* Elemento a ser movido para outro nó */
15                  $s_{mov} \leftarrow s_i;$ 
16                 /* Armazenando o nó filho de  $s_j$  */
17                  $\mathcal{N}^{Ins} \leftarrow NoFilho(s_j);$ 
18                  $dist^{MIN} \leftarrow \delta(s_i, s_j);$ 
19             fim
20         fim
21         /* Se a reorganização é necessária */
22         se  $s_{mov} \neq NULL$  então
23             /* Deletar  $s_{del}$  do nó */
24              $DeletarElemento(s_{mov});$ 
25             /* Reinserir no nó  $\mathcal{N}^{Ins}$  */
26              $del \leftarrow InserirElemento(s_{del}, \mathcal{N}^{Ins});$ 
27             se  $del \neq TRUE$  então
28                 AtualizarRaios( $\mathcal{N}^{Reorg}$ );
29                 para  $\mathcal{N} \in \mathbb{N}^{filhos}$  faça
30                     AtualizarRaios( $\mathcal{N}$ );
31                 fim
32                 retorna  $\mathcal{N}^{ins};$ 
33             fim
34         fim
35     fim
36 fim

```

sendo recuperado o representante do nó na linha 4. Na linha 5 é realizado um laço sobre os elementos do nó filho. Nas linhas 8 a 15 é utilizado um laço para iterar sobre os elementos do nó com os representantes, e são comparadas as distâncias dos elementos para o representante atual com as distâncias desses mesmos elementos para os outros representantes por meio da condição da linha 10, o que corresponde à busca do nó ideal para o elemento sendo verificado. Na linha 16, é verificado se foi encontrado um nó com o representante mais próximo, e caso a condição seja verdadeira, são realizadas a remoção do elemento no nó antigo e a inserção no nó do representante mais próximo nas linhas 17 e 18.

Durante a inserção, é possível que o nó mais próximo do elemento a ser movido esteja cheio. Ao tentar inserir no novo nó na linha 18, será retornado o valor *falso*, fazendo o algoritmo entrar na condição da linha 19. Nesse caso, são realizadas as atualizações nos *raios de cobertura* do nó sendo reorganizado e dos nós filhos, devido às movimentações dos elementos nas linhas 20 até 23. Por fim é retornado o nó que necessita ser quebrado na linha 24. Quando ocorre uma quebra de nós pela reorganização, há duas situações que precisam ser tratadas, que é o caso do nó com os representantes estar cheio ou não. Caso o nó não esteja cheio, é realizada uma quebra de nós e a técnica é executada novamente. Caso o nó esteja cheio, então é realizada a quebra do nó sendo reorganizado, e em seguida são armazenados os representantes dos dois novos nós resultantes. Nesse passo, são realizadas as atualizações necessárias nos nós superiores, se necessário até chegar na raiz. Após as atualizações na árvore, são realizadas buscas para cada representante da lista, e executada a operação de reorganização.

Ao final da execução do algoritmo, a estrutura resultante está organizada de maneira que todos os elementos que estavam em sobreposição agora estão no nó do representante mais próximo entre os nós sobrepostos.

4.2.2 *Vantagens e desvantagens da VD-fit*

A VD-fit possui a vantagem de reduzir a sobreposição dos nós sem a necessidade de armazenar mais informações para construir a estrutura. Assim, evita-se tanto o aumento do espaço de armazenagem necessário, quanto o aumento de leituras nas operações de consulta. Para isso, a técnica proposta considera apenas os elementos representantes e os elementos que estão em sobreposição nos nós filhos, movendo os elementos em regiões sobrepostas para outros nós baseando no Diagrama de Voronoi. Além disso, por ser uma técnica baseada na reorganização dos elementos, a VD-fit não altera a estrutura dos nós da árvore. Dessa maneira, é válido admitir a possibilidade de aplicar a técnica de reordenação de objetos baseada no Diagrama de Voronoi em outras estruturas baseadas em árvore. De maneira geral, conclui-se que a redução da sobreposição de nós acelera as consultas e operações na árvore, uma vez que reduz os sub-espço de busca durante as consultas por meio da poda de mais nós.

Apesar de reduzir a sobreposição sem reduzir o espaço disponível para os dados, as reorganizações aumentam o custo da construção e da atualização da árvore. Isso ocorre porque a

VD-fit exige mais leituras a disco para recuperar os nós filhos, que serão reorganizados durante a operação de construção. Outra desvantagem é o fato da técnica não evitar a possibilidade de ser necessário usar mais nós, sendo esse problema refletido no aumento de acessos a disco para responder as consultas.

De maneira geral, o uso do VD-Tree deve ser condicionado ao melhor cenário de aplicação, considerando um compromisso entre o custo de realizar cálculos de distância e número de acessos a disco. No próximo capítulo, são apresentados experimentos considerando este cenário.

4.3 Considerações finais

Neste capítulo foi apresentando o método VD-Tree, que realiza reorganizações baseadas no Diagrama de Voronoi para reduzir a sobreposição de nós por meio da técnica VD-fit. O VD-Tree foca em melhorar as consultas por meio da redução da sobreposição entre nós, sendo esta redução obtida apenas com reorganizações. Por serem reorganizações baseadas no Diagrama de Voronoi, há um aumento no custo para construir a árvore e no número de nós utilizados, porém a árvore é mantida balanceada e dinâmica, o que é conseguido por manter o uso dos *raios de cobertura*.

Desse modo, apesar de empregar as regras do Diagrama de Voronoi, comparado às estruturas baseadas no Diagrama de Voronoi, o VD-Tree mantém a estrutura flexível sem um aumento exagerado no custo. Comparando com as estruturas que armazenam informações extras para melhorar a capacidade de filtragem durante as buscas, o VD-Tree foca em reduzir a sobreposição possibilitando realizar as consultas sem grandes alterações, e mantém mais espaço para os dados em cada nó. Apesar das estratégias que combinam a *raios de cobertura* com o Diagrama de Voronoi ou hiperplanos não possuem o problema da sobreposição, essas estruturas apresentam grande dificuldade em manter a flexibilidade da estrutura, fazendo com que inserções de novos elementos deteriore rapidamente a estrutura. Por fim, estratégias que realizam reorganizações como a Slim-Tree, apesar reduzirem a sobreposição de nós, o fazem apenas parcialmente e ainda há espaço para melhorias. No próximo capítulo, são apresentados resultados experimentais do VD-Tree, que são comparados com outros MAMs existentes na literatura.

EXPERIMENTOS E RESULTADOS

5.1 Considerações iniciais

Neste capítulo, são apresentados os resultados experimentais do método VD-Tree, avaliando sua capacidade em reduzir a sobreposição de nós e os custos de realizar consultas por similaridade. São reportados resultados da execução de consultas por abrangência e por k -NN, utilizando conjuntos de dados do mundo real. Para variar a cardinalidade e a dimensionalidade, foram utilizados quatro métodos de extração de características aplicados em cinco conjunto de dados.

Como *baseline*, foram realizadas comparações com o MAM Slim-Tree ([TRAINA-JR et al., 2002a](#)), que de maneira semelhante ao VD-Tree, utiliza de reorganizações para reduzir a sobreposição entre nós. Para realizar as comparações sobre as construções das árvores, são obtidos os valores de Fat-Factor e Bloat-Factor para medir a redução de sobreposição nos nós. Para as consultas, são comparados os tempos de execução, acessos a disco e cálculos de distância.

O capítulo está organizado da seguinte maneira: Na [Seção 5.2](#) são apresentados os conjuntos de dados e os extratores de características utilizados, as configurações da máquina onde os experimentos foram realizados e as configurações das árvores e consultas. Na [Seção 5.3](#) são apresentadas as comparações de tempo entre os MAMs para realizar as consultas por abrangência e k -NN. Na [Seção 5.4](#) são apresentados os custos de realizar as consultas considerando acessos a disco e cálculos de distância. Na [Seção 5.5](#) são apresentadas as informações coletadas nas construções das árvores e as observações sobre os resultados. Na [Seção 5.6](#) são apresentadas as conclusões sobre os experimentos.

5.2 Materiais e configurações

Os experimentos foram realizados em um computador com processador Intel(R) Core(TM) i7-2600S CPU @ 2.80GHz, com 8 GB de memória RAM e sistema operacional Ubuntu 18.04 LTS. As estruturas foram implementadas em C++, utilizando a biblioteca *arboretum*¹.

Foram utilizados 5 conjuntos de imagens, e foram aplicados 4 métodos de extração de características, obtendo 20 combinações de representações. A [Tabela 4](#) descreve os conjuntos de dados utilizados para realizar os experimentos, com os nomes dos conjuntos de dados e o número de objetos. A [Tabela 3](#) descreve os métodos de extração de características utilizados nos experimentos, apresentando o nome do extrator, o acrônimo, que tipo de características são descritas e o número de características geradas.

Tabela 3 – Métodos de extração de características utilizados para realizar os experimentos.

Método de extração de características	Acrônimo	Tipo	#Características
Color Layout	CL	Cor	16
Color Structure	CS	Cor	128
Edge Histogram	EH	Formato	150
Local Binary Patterns	LBP	Textura	177

Fonte: Adaptada de [Moriyama et al. \(2021\)](#).

Tabela 4 – Conjunto de dados utilizados nos experimentos.

Conjunto de dados	Número de objetos
Mammoset (OLIVEIRA et al., 2017)	3.457
Flickr-FireSmoke (CAZZOLATO et al., 2017)	5.556
Covid-19 (COHEN et al., 2020)	5.933
Letters (HAJDER, 2020)	15.340
Dogs (KHOSLA et al., 2011)	20.580

Fonte: Adaptada de [Moriyama et al. \(2021\)](#).

Para a construção das árvores, o tamanho dos nós foi definido experimentalmente como 32.768 bytes. Foram escolhidas duas funções de distância para os testes, a Euclidiana e a Manhattan. A política de escolha de nós é a *mindist*, e a política de quebra de nós escolhida foi a MST. Foram realizadas consultas por abrangência e consultas por *k*-NN. Para realizar as consultas, foram escolhidos aleatoriamente como centro de consulta 1% dos objetos de cada combinação de métodos de extração de características com conjunto de dados. Para as consultas por *k*-NN, os valores de *k* foram escolhidos como cinco valores que variam com base na sequência de *Fibonacci*, iniciando no valor 3 até o 21 (3, 5, 8, 13, 21). Para as consultas por abrangência, os valores de ξ são obtidos por meio de uma distribuição logarítmica, utilizando o

¹ Disponível em bitbucket.org/gbdi/arboretum/src

ξ como limite superior, cujo valor é obtido por meio da execução de uma consulta k -NN com $k = 21$ e um centro de consulta aleatório que retorna um ξ maior que 0. Por exemplo, se ao executar o k -NN com $k = 21$ é obtido $\xi = 1,5$, os valores de ξ devem estar dentro do intervalo $(0, 1.5]$. Em seguida é calculada a escala como a diferença entre o valor do ξ e 0 dividido pelo $\log$ de um somado com o número de intervalos, resultando em seis intervalos obtidos nos experimentos. Após definida a escala, são calculados os limites superiores de cada intervalo usando a soma do limite inferior com o $\log$ de dois somado com o número do intervalo e multiplicando o resultado do $\log$ pela escala. Após o cálculo do limite superior, o limite inferior é atualizado com o valor do limite superior e o cálculo é repetido para o próximo intervalo. Assim, são obtidos os valores $\{0,53, 0,84, 1,06, 1,24, 1,38, 1,5\}$.

5.3 Comparações de tempo de execução

Uma das principais vantagens em reduzir a sobreposição de nós nas consultas por similaridade é a redução da quantidade de tempo gasto para responder as consultas. Os resultados dos experimentos ao realizar consultas por k -NN e abrangência utilizando a função de distância Euclidiana são apresentados na imagem [Figura 15](#). O eixo y representa as médias dos tempos de execução ao realizar as consultas utilizando o VD-Tree, e o eixo x representa as médias dos tempos utilizando o Slim-Tree. As variações de formatos dos pontos indicam a variação dos valores de k em (a) e a variação dos *raios de cobertura* em (b). As variações de cores representam os diferentes métodos de extração de características. Quando os pontos estão dentro da região amarela, o VD-Tree gastou menos tempo para responder à consulta, do contrário o Slim-Tree gastou menos tempo.

5.3.1 Comparações de tempo de execução utilizando a função de distância Euclidiana

O método VD-Tree foi mais rápido que o Slim-Tree conforme o número de objetos aumentou em consultas por k -NN. Apesar da maior parte dos resultados dos experimentos de tempo das consultas por abrangência serem mais rápidos utilizando o VD-Tree, a diferença de tempo gasto para responder as consultas foi menor quando comparado com o Slim-Tree. O aumento no número de dimensões não afetou o comportamento dos resultados dos experimentos de tempo apresentados pelo VD-Tree e o Slim-Tree, porém as propriedades dos dados criaram situações em que o método utilizou mais tempo para responder as consultas por k -NN e abrangência. O conjunto de dados Letters possui imagens de letras em diferentes estilos de fontes, com fundo branco. Dessa maneira, todos os objetos possuem cores iguais ou muito semelhantes, e como todas as imagens são variações dos fontes das letras do alfabeto, elas também tem formas semelhantes. Considerando essas propriedades, o VD-Tree apresentou resultados piores devido às reorganizações baseadas no Diagrama de Voronoi terem dificuldade de lidar com muitos

Figura 15 – Comparação do tempo de execução de consultas por k -NN e abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana

Fonte: Adaptada de [Moriyama et al. \(2021\)](#).

objetos com os mesmos vetores de características. Ao realizar reorganizações com tais objetos, todos são movidos para um mesmo representante, o que aumenta a quantidade de quebra de nós realizadas e o número de nós utilizados para indexar os dados, além de aumentar o custo de realizar as consultas. Esse problema também ocorre devido ao método de inserção *mindist* que sempre insere no mesmo nó e o método de quebra de nós MST que utiliza de agrupamento para a quebra de nós. Mesmo com as variações dos valores de k , o método manteve o comportamento em que na maioria dos casos o VD-Tree é mais rápido. De maneira semelhante às consultas por k -NN, nas consultas por abrangência, o método manteve o mesmo comportamento com as variações de ξ .

O VD-Tree foi capaz de reduzir a quantidade de tempo gasto para responder a ambas as consultas por k -NN e abrangência, na maioria dos casos. Foi observada que o VD-Tree, em média, é 1.5 vezes mais rápido que o Slim-Tree para ambas as consultas por k -NN e por abrangência. De maneira semelhante, devido às características do conjunto de dados Letters possuir uma grande quantidade de imagens de cores muito semelhantes ou iguais, o VD-Tree teve um desempenho pior com extratores baseados em cores. Isso ocorre por existirem muitos objetos com vetores de características iguais, que resultam em distância zero. Com muitos objetos com distância zero, poucos objetos podem ser movidos, o que dificulta melhorar a organização dos objetos.

5.3.2 Comparações de tempo de execução utilizando a função de distância Manhattan

A Figura 16 ilustra os resultados de tempo obtidos utilizando a função de distância Manhattan, sendo utilizada a mesma legenda e formato dos gráficos apresentados anteriormente. Os experimentos de tempo utilizando a função de distância Manhattan apresentaram um comportamento semelhante em consultas por k -NN, porém para as consultas por abrangência resultaram em pequenas reduções de tempo gasto, com pequeno aumento nas situações em que a técnica utilizou mais tempo. De maneira semelhante aos experimentos utilizando a distância Euclidiana, o comportamento se manteve ao aumentar o número de objetos em consultas por k -NN. No entanto, em consultas por abrangência o impacto do problema em utilizar os extratores de cores no conjunto de dados Letters foi ainda maior. O aumento no número de dimensões não alterou o comportamento em consultas por k -NN, mas em consultas por abrangência houve problemas semelhantes com o conjunto de dados Letters. O método manteve o mesmo comportamento com as variações de k . As variações dos valores de ξ afetaram significativamente os resultados das consultas, em que vários casos apresentaram consultas obtendo poucos objetos sendo retornados em consultas com raios menores. Devido a essa possibilidade o gráfico (c) da imagem ilustra os resultados de variar o ξ para cada centro de consulta, sendo apresentado um comportamento similar aos experimentos e resultados obtidos pelas consultas por abrangência utilizando a função de distância Euclidiana.

Figura 16 – Comparação do tempo de execução de consultas por k -NN e abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan

Fonte: Elaborada pelo autor.

Para consultas por k -NN utilizando a função de distância Manhattan, o método foi média 1.5 vezes mais rápido. Para as consultas por abrangência, foi em média 1.3 vezes mais rápido com raios fixos e 1.4 vezes mais rápidos com raios variáveis.

5.4 Comparações de cálculos de distância e acessos a disco

O número de cálculos de distância realizados e o número de acessos a disco são valores importantes que refletem a redução do tempo gasto para responder as consultas por similaridade. O aumento nos números de cálculos de distância e acessos a disco são devidos ao aumento na quantidade de nós que são explorados e na quantidade de elementos comparados para responder às consultas. Considerando a importância desses valores, nesta seção são analisados o impacto do método em reduzir os cálculos de distância e acessos a disco.

Nas figuras a seguir são apresentados os gráficos da média dos gastos de cálculos de distância e acessos a disco. As variações dos símbolos, de maneira semelhante, representam as variações de k e ξ das consultas por k -NN e abrangência. Os pontos em azul são os resultados obtidos pelo VD-Tree, enquanto os laranja são os resultados utilizando o Slim-Tree.

5.4.1 Comparações utilizando a função de distância Euclidiana

Os primeiros experimentos apresentados utilizam a função de distância Euclidiana para a construção das árvores e a execução das consultas por abrangência e k -NN.

Na [Figura 17](#) até a [Figura 20](#) são apresentados os gráficos com as médias de acessos a disco para consultas por k -NN e abrangência. Quanto mais próximo do eixo x, menos acessos a disco são necessários, e quanto mais próximo do eixo y, menos tempo foi gasto para responder as consultas. Na [Figura 21](#) até a [Figura 24](#) são apresentados os resultados das consultas k -NN e por abrangência com as médias de cálculos de distância. Pontos próximos de zero no eixo x utilizam menos tempo e pontos mais próximos de zero no eixo y indicam menos cálculos de distância.

Foram observados que o VD-Tree foi de até 3,4 vezes mais rápido e em média houve uma redução de 46% dos cálculos de distância necessários para responder consultas por k -NN, com o aumento de 24% mais acessos a disco. Para consultas por abrangência, foram necessários em média 44% menos cálculos de distâncias e o VD-Tree foi 3,2 vezes mais rápido para responder as consultas, ao custo de utilizar 23% mais acessos a disco. Uma vez que a estrutura utiliza mais nós para organizar os elementos com menos sobreposição, foram necessário mais acessos a disco para responder às consultas. No entanto, a redução dos cálculos de distância possibilitou que o tempo necessário para responder as consultas fosse reduzido.

Figura 17 – Comparação do tempo de execução e acessos a disco em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana

Figura 18 – Comparação do tempo de execução e acessos a disco em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana

Fonte: Adaptada de Moriyama *et al.* (2021).

Figura 19 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana

Fonte: Adaptada de [Moriyama et al. \(2021\)](#).

Figura 20 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana

Fonte: Adaptada de Moriyama *et al.* (2021).

Figura 21 – Comparação do tempo de execução e cálculos de distância em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana

Fonte: Adaptada de [Moriyama et al. \(2021\)](#).

Figura 22 – Comparação do tempo de execução e cálculos de distância em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana

Fonte: Adaptada de Moriyama *et al.* (2021).

Figura 23 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana

Fonte: Adaptada de [Moriyama et al. \(2021\)](#).

Figura 24 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana

Fonte: Adaptada de [Moriyama et al. \(2021\)](#).

5.4.2 Comparações utilizando a função de distância Manhattan

Foram executadas a construção e as consultas por similaridade utilizando a função de distância Manhattan, para testar o impacto de alterar a função de distância. Para os experimentos, foram utilizadas as mesmas configurações nos outros parâmetros, sendo obtidos os resultados de realizar consultas por k -NN e abrangência. As legendas são as mesmas utilizadas para as figuras anteriores.

As Figuras 25 a 28 mostram as comparações entre o Slim-Tree e o VD-Tree, considerando o tempo gasto para executar a consulta e o número de acessos a disco, considerando também o tempo gasto e o número de cálculos de distância. Um comportamento similar pôde ser observado, em que a redução do número de cálculos de distância é grande o suficiente para reduzir o tempo gasto para responder as consultas, mesmo com o aumento nos acessos a disco. Como pode ser observado nos resultados de tempo e em como os números de acessos a disco e cálculos de distância influenciam nos resultados de tempo, o comportamento se manteve o mesmo ao aumentar o número de objetos e o número de dimensões nos experimentos. As variações de k também mantiveram o mesmo comportamento, em que o número de acessos a disco aumenta, porém o número de cálculos de distância são reduzidos.

Os resultados das consultas por abrangência utilizando a função de distância Manhattan apresentaram um comportamento similar, em que o VD-Tree apresenta redução no número de cálculos de distância e tempo ao custo do aumento de acessos a disco. De maneira semelhante, foram reduzidos os números de cálculos de distância significativamente, de modo que mesmo com o aumento nos acessos a disco, o tempo é reduzido. Como pode ser observado da Figura 29 até a Figura 32, há resultados muito próximos com a variação do ξ . Isso ocorre devido ao espalhamento dos dados e a escolha dos centros de consultas. Como os valores de ξ são os mesmos para todos os centro de consultas, em casos como a combinação do extrator CL com o conjunto de dados *Mammaset*, em muitas consultas apenas o maior raio obteve uma quantidade significativamente maior de elementos. Nesses experimentos, conforme o número de objetos aumentou, o comportamento dos resultados aproximou-se dos experimentos utilizando a função de distância Euclidiana. O método obteve melhores resultados nos métodos de extração de características CL e LBP, enquanto nos métodos de extração de características CS e EH, apenas em conjuntos de dados maiores houve melhora na redução do custo de executar as consultas.

Considerando a possibilidade da escolha aleatória de um centro de consulta gerar valores de ξ em que os primeiros raios retornem apenas o objeto de busca, foram realizados testes com os valores de ξ variando para cada centro de consulta, ilustrados nas Figuras 33 até a Figura 36. Os resultados utilizando ξ variando para cada centro de consulta possui um comportamento semelhante aos experimentos anteriores, de que há uma considerável redução de cálculos de distância, que faz com que o tempo para realizar as consultas seja reduzido, e haja um aumento no número de acessos a disco. Esse comportamento mostra que o método sofre em consultas que retornam poucos elementos, principalmente em conjuntos de dados pequenos. Isso ocorre devido

Figura 25 – Comparação do tempo de execução e acessos a disco em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan

Fonte: Elaborada pelo autor.

Figura 26 – Comparação do tempo de execução e acessos a disco em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan

Fonte: Elaborada pelo autor.

Figura 27 – Comparação do tempo de execução e cálculos de distância em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan

Fonte: Elaborada pelo autor.

Figura 28 – Comparação do tempo de execução e cálculos de distância em consultas por k -NN entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan

Fonte: Elaborada pelo autor.

ao método utilizar mais nós para construir as árvores. Quando são realizadas consultas com raios que retornam muito poucos elementos, são necessárias poucas podas, o que contribui para que a árvore construída pelo método, que utiliza mais nós e por consequência possivelmente construa árvores mais altas, apresente resultados piores. Com os valores de ξ variando para cada centro de consulta, o comportamento dos resultados foi semelhante ao de variar o número de objetos e o número de dimensões.

Figura 29 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan

Fonte: Elaborada pelo autor.

A utilização do VD-Tree com a função de distância Manhattan foi até 2,8 vezes mais rápido para responder consultas por k -NN, o número de cálculos de distância foi reduzido em média em 44,47% ao custo de aumentar em média 29,94% o número de acessos a disco. Para consultas por abrangência com ξ fixos, o VD-Tree foi até 3,7 vezes mais rápido que o

Figura 30 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan

Fonte: Elaborada pelo autor.

Figura 31 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan

Fonte: Elaborada pelo autor.

Figura 32 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan

Fonte: Elaborada pelo autor.

Figura 33 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan e variando a geração do raio de busca por centro de consulta.

Fonte: Elaborada pelo autor.

Figura 34 – Comparação do tempo de execução e cálculos de distância em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan e variando a geração do raio de busca por centro de consulta.

Fonte: Elaborada pelo autor.

Figura 35 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan e variando a geração do raio de busca por centro de consulta.

Fonte: Elaborada pelo autor.

Figura 36 – Comparação do tempo de execução e acessos a disco em consultas por abrangência entre o Slim-Tree e o VD-Tree utilizando a função de distância Manhattan e variando a geração do raio de busca por centro de consulta.

Fonte: Elaborada pelo autor.

Slim-Tree, em média houve uma melhora de 26,43% na redução dos cálculos de distância e um aumento médio de 11,16% no número de acessos a disco. Em consultas por abrangência com o ξ variando para cada centro de consulta, o VD-Tree foi 2,6 vezes mais rápido e reduziu o número de cálculos de distância em 46,07%, enquanto houve um aumento em 27,21% em acessos a disco.

5.5 Resultados das construções das árvores

Nesta seção, são avaliadas as árvores construídas utilizando o VD-Tree e o Slim-Tree. Para comparar as estruturas, foram coletadas as seguintes informações:

- O tempo de construção.
- O número de nós folha.
- O número de nós diretório.
- A altura.
- O Fat-Factor.
- O Bloat-Factor.

As Tabelas 5 e 6, mostram na primeira coluna os IDs representando as combinações do conjuntos de dados e extratores utilizados para construir as árvores dos experimentos, sendo as legendas dos IDs mostradas logo abaixo das tabelas.

Essas informações estão organizadas na Tabela 5, em que cada linha da tabela são os conjuntos de dados utilizados e as colunas são os métodos de extração de características. Foi observado que o VD-Tree reduz a sobreposição medida pelo Fat-Factor, porém utiliza mais nós para organizar melhor os elementos. O VD-Tree também utiliza mais tempo para construir a árvore, devido à necessidade realizar reorganizações que exigem cálculos de distâncias e acessos a disco. No entanto, foi observado que o método reduz em até 43% a sobreposição de nós medido pelo Fat-Factor.

A Tabela 6 apresenta os dados obtidos da construção das árvores, utilizando o mesmo formato, porém com a função de distância Manhattan. As árvores construídas utilizando o VD-Tree com a função de distância Manhattan também obtiveram um aumento no custo de tempo e o número de nós, ocorrendo em alguns casos até mesmo aumento na altura da árvore. Porém, foram reduzidos os valores de Fat-Factor e em alguns casos também de Bloat-Factor, demonstrando que o VD-Tree constrói árvores maiores e em mais tempo, mas reduz significativamente a sobreposição entre os nós. Diferentemente das árvores construídas utilizando a distância Euclidiana, as árvores construídas utilizando a Manhattan não apresentaram redução na sobreposição em todos os casos, sendo a exceção a combinação do conjunto de dados Dogs com o extrator CL.

Tabela 5 – Tempo de construção, número de nós folhas e índices, altura, Fat-Factor e Bloat-Factor das árvores construídas pelo Slim-Tree e o VD-Tree utilizando a função de distância Euclidiana

ID	DS	FEM	VD-Tree				Fat	Bloat	Slim-Tree				Fat	Bloat
			Const(s)	Folhas	Diretório	Altura			Const(s)	Folhas	Diretório	Altura		
1	CL		28,835	60	1	2	0,549347	1,062090	3,165	34	1	2	0,609873	0,650542
1	CS		48,634	318	24	3	0,108181	0,182827	5,452	256	11	3	0,222181	0,292169
1	EH		49,814	526	50	4	0,101856	0,247530	6,221	314	16	3	0,189351	0,258279
1	LBP		43,517	435	34	4	0,048550	0,082843	6,491	355	19	3	0,184796	0,240757
2	CL		636,936	168	1	2	0,365007	0,696451	15,775	103	1	2	0,551741	0,640528
2	CS		146,499	1331	106	4	0,187697	0,392695	22,559	853	38	4	0,346995	0,448963
2	EH		148,389	1958	208	4	0,129163	0,351096	26,520	1144	76	4	0,235954	0,360609
2	LBP		144,499	1777	168	4	0,063142	0,130732	26,787	1281	72	4	0,191891	0,276062
3	CL		30,081	60	1	2	0,397920	0,887687	3,314	30	1	2	0,408625	0,440067
3	CS		38,626	308	21	3	0,167476	0,291069	5,161	233	11	3	0,262146	0,336445
3	EH		42,657	506	43	4	0,204119	0,500182	5,700	261	13	3	0,343534	0,414082
3	LBP		36,839	441	39	4	0,075182	0,138013	6,389	365	21	3	0,176220	0,253068
4	CL		315,355	123	1	2	0,112981	0,213606	11,939	80	1	2	0,179693	0,219001
4	CS		151,349	1356	233	5	0,142657	0,444972	16,391	606	27	3	0,313576	0,385233
4	EH		106,703	1093	97	4	0,118177	0,238243	17,910	771	37	4	0,239791	0,326946
4	LBP		118,659	1218	108	4	0,038237	0,072261	19,594	995	56	4	0,158389	0,237016
5	CL		9,807	37	1	2	0,371999	0,765919	1,933	19	1	2	0,416189	0,416210
5	CS		32,136	186	16	3	0,100903	0,170761	3,096	161	8	3	0,159294	0,224647
5	EH		24,169	309	35	4	0,171616	0,422703	3,421	185	12	3	0,291756	0,402210
5	LBP		24,203	281	22	3	0,054454	0,097496	3,817	239	14	3	0,094655	0,141133

Conjuntos de Dados (DS): 1 - Mammoset, 2 - Flickr-FireSmoke, 3 - Covid-19, 4 - Letters, 5 - Dogs

Fonte: Adaptada de [Moriyama et al. \(2021\)](#).

Como o Bloat-Factor penaliza o uso de mais nós, as árvores que apresentaram os maiores valores de Bloat-Factor foram as que necessitaram de mais nós para manter as reorganizações baseadas em Diagrama de Voronoi. Isso ocorreu principalmente em árvores construídas utilizando o método de extração de características CL, que gera estruturas muito pequenas. Nesses casos, a Slim-Tree obteve uma melhor performance, devido à estratégia de forçar uma ocupação mínima nos nós na Slim-Tree, que reduz a quantidade de nós utilizados. Outras características que contribuí para esses casos são as reorganizações baseadas no Diagrama de Voronoi, que exigem que os objetos estejam no nó do representante mais próximo, o que aumenta a possibilidade de ser necessário executar uma quebra de nós.

Apesar do VD-Tree utilizar mais nós e ser mais custoso para ser construído, foram obtidas melhores organizações de elementos, o que possibilitou reduzir a sobreposição de nós e acelerar as consultas por similaridade. Essa característica é refletida nos valores de Fat-Factor e Bloat-Factor que, por penalizar o uso de mais nós, resultou em valores de Bloat-Factor maiores que os do Slim-Tree, em parte dos experimentos. Outra observação importante é que, conforme a árvore cresce, ocorrem menos quebras de nós ou trocas de representantes, o que reduz a quantidade de reorganizações. Dessa maneira, com a árvore suficientemente grande, poucas inserções exigem que reorganizações sejam realizadas, o que reduz o crescimento do custo de construir a árvore.

Tabela 6 – Tempo de construção, número de nós folhas e índices, altura, Fat-Factor e Bloat-Factor das árvores construídas pelo Slim-Tree e o VD-Tree utilizando a função de distância Manhattan

ID		VD-Tree						Slim-Tree					
ds	fem	Const(s)	Folhas	Índices	Altura	Fat	Bloat	Const(s)	Folhas	Índices	Altura	Fat	Bloat
1	CL	23,777	53	1	2	0,599350	1,018910	3,396	38	1	2	0,639872	0,767859
1	CS	47,482	317	18	3	0,083893	0,138843	5,177	251	11	3	0,211508	0,272847
1	EH	44,532	490	53	4	0,114955	0,262954	5,939	302	15	3	0,199254	0,260948
1	LBP	41,207	445	34	4	0,069150	0,118933	6,542	359	20	3	0,198782	0,262477
2	CL	718,340	188	1	2	0,476202	1,018090	14,805	97	1	2	0,602036	0,657396
2	CS	126,073	1494	148	4	0,266300	0,636001	24,413	997	62	4	0,366257	0,563369
2	EH	149,421	1984	257	4	0,134464	0,378190	24,891	1086	60	4	0,229043	0,328728
2	LBP	134,719	1825	202	4	0,083390	0,179951	25,764	1257	73	4	0,234846	0,332094
3	CL	27,654	60	1	2	0,619951	1,383000	3,015	28	1	2	0,649596	0,649609
3	CS	40,790	354	27	3	0,222412	0,448393	5,159	243	13	3	0,288755	0,389127
3	EH	40,660	470	46	4	0,201392	0,463891	5,747	284	17	3	0,296337	0,392912
3	LBP	34,594	453	34	4	0,097117	0,179738	6,402	366	21	3	0,220910	0,318077
4	CL	320,505	122	1	2	0,158450	0,297095	10,958	76	1	2	0,312324	0,361126
4	CS	134,571	1282	187	4	0,163376	0,469106	16,491	634	31	3	0,314378	0,405868
4	EH	113,821	1112	86	4	0,119101	0,241702	17,193	775	40	4	0,251448	0,345732
4	LBP	103,801	1234	109	4	0,065121	0,124652	17,995	973	56	4	0,192143	0,281480
5	CL	9,263	40	1	2	0,441796	0,987595	1,927	20	1	2	0,439013	0,464861
5	CS	27,725	184	14	3	0,128826	0,213611	2,892	154	7	3	0,161896	0,217247
5	EH	21,679	292	34	4	0,213792	0,497341	3,280	175	10	3	0,274255	0,354575
5	LBP	24,045	272	26	4	0,089227	0,162537	3,505	223	13	3	0,129626	0,180080

Conjuntos de Dados (DS): 1 - Mammaset, 2 - Flickr-FireSmoke, 3 - Covid-19, 4 - Letters, 5 - Dogs

Fonte: Elaborada pelo autor.

5.6 Considerações Finais

Neste capítulo foram apresentados os experimentos realizados comparando o VD-Tree com o MAM mais semelhante, o Slim-Tree, e as conclusões sobre os resultados obtidos. Foram coletadas as principais medidas para avaliar os custos da execução de consultas por similaridade e as construções das árvores. Para realizar as comparações, foram utilizados 5 conjuntos de dados de diversas aplicações, com cardinalidades diferentes, para analisar o impacto de variar a cardinalidade e 4 métodos de extração de características para analisar o impacto da variação de dimensionalidade.

Por meio dos experimentos, foi observado que, apesar do método ser capaz de reduzir a sobreposição dos nós na maioria dos casos, há um aumento do custo de construção das árvores e do número necessário de nós para indexar os dados. Foi observado que as reduções de tempo são realizadas por meio da redução da sobreposição de nós combinadas com a melhor organização dos elementos nos nós. Essas duas características possibilitaram reduções significativas no número de cálculos de distância, de maneira que, mesmo com o aumento nos acessos a disco, o tempo gasto para realizar as consultas fica reduzido.

CONCLUSÃO

6.1 Considerações iniciais

Para realizar consultas em grandes conjuntos de dados, o uso de índices torna-se indispensável. Índices são ainda mais necessários em consultas por similaridade em dados complexos, onde são utilizadas as funções de distância para realizar as comparações entre os dados.

Entre as diversas abordagens existentes na literatura, uma estratégia comumente utilizada são as estruturas hierárquicas em formato de árvores, que possibilitam acelerar as consultas ao custo de aumentar o tempo gasto na construção da estrutura. Entre as diversas estratégias baseadas em árvore, uma das que se destacam são as estratégias baseadas em *raios de cobertura*, que equilibram o custo da construção da estrutura com o custo da execução de consultas. Porém, um dos principais problemas das estratégias baseadas em *raios de cobertura*, é a sobreposição de nós. A sobreposição de nós aumenta o custo de realizar consultas na estrutura, devido à necessidade de explorar todos os nós que se qualificam para garantir que a resposta esteja correta. Porém, a sobreposição de nós possibilita reduzir o custo da construção da estrutura por manter sua flexibilidade no posicionamento de objetos.

Com o objetivo de reduzir a sobreposição de nós, aumentando o mínimo possível o custo de construção, e manter a estrutura flexível, foi proposto neste trabalho o VD-Tree. O VD-Tree combina a estrutura flexível dos *raios de cobertura* com uma estratégia rígida de particionamento de espaço baseado no Diagrama de Voronoi.

Neste capítulo, são apresentadas as conclusões obtidas neste trabalho de Mestrado, assim como as principais contribuições, publicações e os trabalhos futuros. O capítulo está organizado na seguinte forma:

- Na [Seção 6.2](#) são listadas as contribuições obtidas neste trabalho de Mestrado e as publicações resultantes.

- Na [Seção 6.3](#) são apresentadas oportunidades de trabalhos futuros, que foram identificados ao longo do desenvolvimento do presente trabalho.

6.2 Contribuições

A principal contribuição deste trabalho de Mestrado foi o MAM VD-Tree, que reduz a sobreposição entre os nós e acelera as consultas por similaridade por meio da redução dos cálculos de distância. As reduções são obtidas por meio de reorganizações dos objetos nos nós baseadas no Diagrama de Voronoi, de maneira que todo objeto que esteja em sobreposição de nós, esteja no nó do representante mais próximo.

Este método é implementado em duas componentes: A primeira é o algoritmo VD-fit, que reorganiza os objetos que estão em regiões com sobreposição de nós, movendo-os para o nó do representante mais próximo. A segunda é um conjunto de alterações nos algoritmos de inserção e atualização de dados, para executar o algoritmo VD-fit e atualizar as informações corretamente, mantendo assim a consistência das reorganizações baseadas no Diagrama de Voronoi.

Combinar a estratégia baseada em *raios de cobertura* com o Diagrama de Voronoi possibilita reduzir a sobreposição, com o custo de utilizar mais nós para armazenar os dados. Porém, resultados experimentais mostraram que o aumento no tamanho da estrutura é compensado com vantagem pela maior capacidade de podar sub-árvores ao realizar as consultas por similaridade. Também foi observado o problema da técnica com conjuntos de dados com muitos elementos com vetores de características iguais. Devido às reorganizações sempre moverem os elementos para o representante mais próximo, muitos elementos acabam sendo movidos para apenas um nó, forçando mais quebras e o uso de mais nós.

Os MAMs baseados em *raios de cobertura* são rápidos de construir devido à flexibilidade dos posicionamento dos elementos. Porém, devido à sobreposição de nós, há um custo extra ao realizar as buscas por similaridade exatas. Ao contrário dos MAMs baseados em *raios de cobertura*, MAMs baseados no Diagrama de Voronoi possibilitam realizar consultas rapidamente, porém com a etapa de construção muito custosa. A estratégia proposta busca equilibrar o custo da construção da estrutura e o custo de se realizar consultas por similaridade. Para reduzir o custo excessivo de manter as regras do Diagrama de Voronoi, as reorganizações são realizadas apenas quando ocorrem quebra de nós ou troca de representantes, considerando apenas o nó que deve ser reorganizado e os seus filhos do nível seguinte. Dessa maneira, a sobreposição de nós é reduzida e as consultas por similaridade são aceleradas, sem que seja necessário um grande aumento no custo de construir a estrutura. Essa compensação de custos faz com que o método proposto VD-Tree obtenha resultados melhores que Slim-Tree, conforme mostrado experimentalmente no Capítulo 5.

6.2.1 Publicações

As duas publicações resultantes deste trabalho de Mestrado são:

- **Principal: SAC 2021 - Qualis-CC A2** ([MORIYAMA et al., 2021](#)) - MORIYAMA, A.; RODRIGUES, L. S.; SCABORA, L. C.; CAZZOLATO, M. T.; TRAINA, A. J. M.; TRAINA-JR, C. Vd-tree: how to build an efficient and fit metric access method using voronoi diagrams. In: HUNG, C.; HONG, J.; BECHINI, A.; SONG, E. (Ed.). SAC '21: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, Republic of Korea, March 22-26, 2021. ACM, 2021. p. 327–335.
- **Complementar: SAC 2019 - Qualis-CC A2** ([TRAINA-JR et al., 2019](#)) - TRAINA-JR, C.; MORIYAMA, A.; ROCHA, G. M. da; CORDEIRO, R. L. F.; CIFERRI, C. D. de A.; TRAINA, A. J. M. The similarql framework: similarity queries in plain SQL. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019. ACM, 2019. p. 468–471.

6.3 Trabalhos futuros

Como trabalhos futuros resultantes deste trabalho de Mestrado, foram identificadas pelo menos as seguintes oportunidades:

- *Utilização de outras estratégias baseadas em agrupamento para realizar as quebras de nós:* Observou-se que a MST apresentou bons resultados na redução do número de objetos que necessitam ser movidos. Isso ocorreu porque a técnica separa os elementos em grupos, e escolhe elementos representantes no centro desses grupos, o que reduz a quantidade de nós que precisam ser movidos para manter as organizações baseadas no Diagrama de Voronoi. Dessa forma, heurísticas parecidas podem favorecer a indexação gerada pelo VD-Tree.
- *Adaptar e analisar o impacto de utilizar as reorganizações como um algoritmo de pós-processamento após a estrutura estar construída:* Apesar de ser uma operação custosa por ter que buscar cada nó folha e propagar a reorganização até o topo da árvore, esta estratégia pode reduzir o número de nós necessários para indexar os dados e consequentemente a altura da estrutura.
- *Considerar o nó pai inteiro para realizar as quebras de nós:* Na estratégia atual, a quebra de nós considera apenas o representante do nó filho e, após a quebra de nós, seleciona dois novos objetos para serem os novos representantes a serem promovidos para o nó pai. Em seguida, o algoritmo distribui os elementos nos dois novos nós filhos. Porém, considerar o nó pai inteiro para realizar a quebra de nós torna possível escolher representantes melhores,

de modo a balancear a ocupação dos nós e organizar os objetos nos filhos baseando-se no Diagrama de Voronoi. As reorganizações são realizadas durante o processo de distribuição dos elementos nos nós filhos.

- Considerar utilização de nós de *overflow*: Na estratégia atual, sempre que um nó precisa ser quebrado, a quebra ocorre imediatamente, e pode gerar nós com relativamente poucos elementos. Usar um nó extra para postergar a quebra, seguindo uma estratégia semelhante à estrutura EGNAT, pode reduzir a necessidade de expansão da árvore e produzir nós mais "densos", revertendo o aumento da quantidade de nós do VD-Tree.

6.4 Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, do Conselho Nacional de Pesquisa Científica (CNPq), e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

Agradeço principalmente aos meus pais Cláudia e Carlos por sempre me apoiarem e acreditarem em mim, independente de todas as dificuldades. Também agradeço muito ao meu irmão Daniel que sempre esteve disposto a me ajudar quando eu precisei.

Tenho muito a agradecer ao Prof. Dr. Caetano Traina Júnior, principalmente pela paciência para lidar com tantos problemas que tivemos durante esse trabalho de mestrado, mas também pelas discussões produtivas e as lições. Agradeço também à sua esposa Profa. Dra. Agma Juci Machado Traina que também me ensinou valiosas lições.

Relembrando todos problemas e dificuldades que ocorreram, eu não posso deixar de agradecer aos meus colegas de laboratório, em especial a Mirela, o Lucas Santiago e o Lucas Scabora, pelas valiosas discussões que sempre foram muito produtivas e pela ajuda que me deram.

Agradeço também a todos os professores e professoras pelos ensinamentos necessários para realizar este trabalho de mestrado.

Por fim, eu gostaria de agradecer minha falecida avó Regina, que foi a pessoa que mais me motivou a encarar o desafio de fazer um mestrado.

REFERÊNCIAS

- BERCHTOLD, S.; BÖHM, C.; KEIM, D. A.; KRIEGEL, H. A cost model for nearest neighbor search in high-dimensional data space. In: MENDELZON, A. O.; ÖZSOYOGLU, Z. M. (Ed.). **Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona, USA**. ACM Press, 1997. p. 78–86. Disponível em: <<https://doi.org/10.1145/263661.263671>>. Citado na página 38.
- BOZKAYA, T.; ÖZSOYOGLU, Z. M. Indexing large metric spaces for similarity search queries. **ACM Trans. Database Syst.**, v. 24, n. 3, p. 361–404, 1999. Disponível em: <<https://doi.org/10.1145/328939.328959>>. Citado nas páginas 17 e 35.
- BRIN, S. Near neighbor search in large metric spaces. In: DAYAL, U.; GRAY, P. M. D.; NISHIO, S. (Ed.). **VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland**. Morgan Kaufmann, 1995. p. 574–584. Disponível em: <<http://www.vldb.org/conf/1995/P574.PDF>>. Citado nas páginas 19, 36 e 42.
- CAZZOLATO, M. T.; AVALHAIS, L.; CHINO, D.; RAMOS, J. S.; SOUZA, J. A. de; RODRIGUES-JR, J. F.; TRAINA, A. Fismo: A compilation of datasets from emergency situations for fire and smoke analysis. In: **Proceedings of Satellite Events of the 32nd Brazilian Symposium on Databases SBBD - DSW (Dataset Showcase Workshop)**. Uberlândia, Brazil: SBC, 2017. p. 213–223. ISBN 978-85-7669-399-4. Citado na página 56.
- CHEN, L.; GAO, Y.; SONG, X.; LI, Z.; MIAO, X.; JENSEN, C. S. Indexing metric spaces for exact similarity search. **CoRR**, abs/2005.03468, 2020. Disponível em: <<https://arxiv.org/abs/2005.03468>>. Citado na página 17.
- CHEN, L.; GAO, Y.; ZHENG, B.; JENSEN, C. S.; YANG, H.; YANG, K. Pivot-based metric indexing. **Proc. VLDB Endow.**, v. 10, n. 10, p. 1058–1069, 2017. Disponível em: <<http://www.vldb.org/pvldb/vol10/p1058-gao.pdf>>. Citado na página 17.
- CIACCIA, P.; PATELLA, M. Searching in metric spaces with user-defined and approximate distances. **ACM Trans. Database Syst.**, v. 27, n. 4, p. 398–437, 2002. Disponível em: <<https://doi.org/10.1145/582410.582412>>. Citado na página 18.
- CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree: An efficient access method for similarity search in metric spaces. In: JARKE, M.; CAREY, M. J.; DITTRICH, K. R.; LOCHOVSKY, F. H.; LOUCOPOULOS, P.; JEUSFELD, M. A. (Ed.). **VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece**. Morgan Kaufmann, 1997. p. 426–435. Disponível em: <<http://www.vldb.org/conf/1997/P426.PDF>>. Citado nas páginas 18, 35 e 36.
- CODD, E. F. **The Relational Model for Database Management, Version 2.** : Addison-Wesley, 1990. ISBN 0-201-14192-2. Citado na página 17.
- COHEN, J. P. *et al.* Covid-19 image data collection: Prospective predictions are the future. **CoRR**, abs/2006.11988, 2020. Disponível em: <<https://arxiv.org/abs/2006.11988>>. Citado na página 56.

- DOHNAL, V.; GENNARO, C.; SAVINO, P.; ZEZULA, P. D-index: Distance searching index for metric data sets. **Multim. Tools Appl.**, v. 21, n. 1, p. 9–33, 2003. Disponível em: <<https://doi.org/10.1023/A:1025026030880>>. Citado na página 33.
- FALOUTSOS, C. **Searching Multimedia Databases by Content**. Boston, MA: Kluwer Academic Publishers, 1996. Citado na página 22.
- GRUHNE, M.; DITTMAR, C.; GÄRTNER, D. Improving rhythmic similarity computation by beat histogram transformations. In: **Proceedings of the 10th International Society for Music Information Retrieval Conference, ISMIR 2009, Kobe International Conference Center, Kobe, Japan, October 26-30, 2009**. International Society for Music Information Retrieval, 2009. p. 177–182. Disponível em: <<http://ismir2009.ismir.net/proceedings/OS2-1.pdf>>. Citado na página 23.
- HAFNER, J. L.; SAWHNEY, H. S.; EQUITZ, W.; FLICKNER, M.; NIBLACK, W. Efficient color histogram indexing for quadratic form distance functions. **IEEE Trans. Pattern Anal. Mach. Intell.**, v. 17, n. 7, p. 729–736, 1995. Disponível em: <<https://doi.org/10.1109/34.391417>>. Citado na página 23.
- HAJDER, S. **Letters organized by typefaces - Standard Windows fonts with each letters organized in classes by typeface**. 2020. Disponível em: <<https://www.kaggle.com/killen/bw-font-typefaces>>. Citado na página 56.
- HANYF, Y.; SILKAN, H. A queries-based structure for similarity searching in static and dynamic metric spaces. **J. King Saud Univ. Comput. Inf. Sci.**, v. 32, n. 2, p. 188–196, 2020. Disponível em: <<https://doi.org/10.1016/j.jksuci.2018.05.004>>. Citado nas páginas 18 e 32.
- HARALICK, R. M.; SHANMUGAM, K. S.; DINSTEIN, I. Textural features for image classification. **IEEE Trans. Syst. Man Cybern.**, Ieee, v. 3, n. 6, p. 610–621, 1973. Disponível em: <<https://doi.org/10.1109/TSMC.1973.4309314>>. Citado na página 23.
- KASUTANI, E.; YAMADA, A. The mpeg-7 color layout descriptor: a compact image feature description for high-speed image/video segment retrieval. In: **Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)**. IEEE, 2001. v. 1, p. 674–677. Disponível em: <<https://doi.org/10.1109/ICIP.2001.959135>>. Citado na página 23.
- KHOSLA, A. *et al.* Novel dataset for fine-grained image categorization: Stanford dogs. In: **Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)**. Colorado Springs, CO: , 2011. v. 2, n. 1. Citado na página 56.
- KOUAHLA, Z.; ANJUM, A.; AKRAM, S.; SABA, T.; MARTINEZ, J. Xm-tree: data driven computational model by using metric extended nodes with non-overlapping in high-dimensional metric spaces. **Comput. Math. Organ. Theory**, v. 25, n. 2, p. 196–223, 2019. Disponível em: <<https://doi.org/10.1007/s10588-018-9272-x>>. Citado nas páginas 17, 18, 31 e 44.
- KOUAHLA, Z.; MARTINEZ, J. A new intersection tree for content-based image retrieval. In: LAMBERT, P. (Ed.). **10th International Workshop on Content-Based Multimedia Indexing, CBMI 2012, Annecy, France, June 27-29, 2012**. IEEE, 2012. p. 1–6. Disponível em: <<https://doi.org/10.1109/CBMI.2012.6269793>>. Citado na página 43.
- LOGAN, B. Mel frequency cepstral coefficients for music modeling. In: **ISMIR 2000, 1st International Symposium on Music Information Retrieval, Plymouth, Massachusetts, USA, October 23-25, 2000, Proceedings**. International Society for Music Information Retrieval, 2000.

v. 270, p. 1–11. Disponível em: <http://ismir2000.ismir.net/papers/logan_paper.pdf>. Citado na página 23.

MORIYAMA, A.; RODRIGUES, L. S.; SCABORA, L. C.; CAZZOLATO, M. T.; TRAINA, A. J. M.; TRAINA-JR, C. Vd-tree: how to build an efficient and fit metric access method using voronoi diagrams. In: HUNG, C.; HONG, J.; BECHINI, A.; SONG, E. (Ed.). **SAC '21: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, Republic of Korea, March 22-26, 2021**. ACM, 2021. p. 327–335. Disponível em: <<https://doi.org/10.1145/3412841.3441915>>. Citado nas páginas 50, 56, 58, 62, 63, 64, 65, 66, 67, 68, 69, 84 e 89.

OJALA, T.; PIETIKÄINEN, M.; HARWOOD, D. A comparative study of texture measures with classification based on featured distributions. **Pattern recognition**, Elsevier, v. 29, n. 1, p. 51–59, 1996. Disponível em: <[https://doi.org/10.1016/0031-3203\(95\)00067-4](https://doi.org/10.1016/0031-3203(95)00067-4)>. Citado na página 23.

OLIVEIRA, P. H.; SCABORA, L. C.; CAZZOLATO, M. T.; BEDO, M. V.; TRAINA, A. J.; TRAINA-JR, C. MAMMOSET: An Enhanced Dataset of Mammograms. In: **Proceedings of Satellite Events of the 32nd Brazilian Symposium on Databases SBBD - DSW (Dataset Showcase Workshop)**. Uberlândia, Brazil: SBC, 2017. p. 256–266. Citado na página 56.

PAREDES, R. U.; NAVARRO, G. EGNAT: A fully dynamic metric access method for secondary memory. In: SKOPAL, T.; ZEZULA, P. (Ed.). **Second International Workshop on Similarity Search and Applications, SISAP 2009, 29-30 August 2009, Prague, Czech Republic**. IEEE Computer Society, 2009. p. 57–64. Disponível em: <<https://doi.org/10.1109/SISAP.2009.20>>. Citado na página 42.

POLA, I. R. V.; TRAINA-JR, C.; TRAINA, A. J. M. The nobh-tree: Improving in-memory metric access methods by using metric hyperplanes with non-overlapping nodes. **Data Knowl. Eng.**, v. 94, p. 65–88, 2014. Disponível em: <<https://doi.org/10.1016/j.datak.2014.09.001>>. Citado nas páginas 17, 18, 32, 36, 43 e 44.

POPOVA, A. A.; NESHOV, N. N. Combining features evaluation approach in content-based image search for medical applications. In: KOUNTCHEV, R.; IANTOVICS, B. (Ed.). **Advances in Intelligent Analysis of Medical Data and Decision Support Systems**. Springer, 2013, (Studies in Computational Intelligence, v. 473). p. 113–123. Disponível em: <https://doi.org/10.1007/978-3-319-00029-9_10>. Citado na página 24.

SHAH, D. M.; DESAI, U. A survey on combine approach of low level features extraction in cbir. In: IEEE. **2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)**. 2017. p. 284–289. Disponível em: <<https://doi.org/10.1109/ICIMIA.2017.7975620>>. Citado na página 23.

SKOPAL, T.; POKORNÝ, J.; SNÁSEL, V. Pm-tree: Pivoting metric tree for similarity search in multimedia databases. In: **Advances in Databases and Information Systems, 8th East European Conference, ADBIS 2004, Budapest, Hungary, September 22-25, 2004, Local Proceeding**. Springer Berlin Heidelberg, 2004. Disponível em: <<http://www.sztaki.hu/conferences/ADBIS/9-Skopal.pdf>>. Citado nas páginas 36 e 41.

SMEULDERS, A. W. M.; WORRING, M.; SANTINI, S.; GUPTA, A.; JAIN, R. C. Content-based image retrieval at the end of the early years. **IEEE Trans. Pattern Anal. Mach. Intell.**, v. 22, n. 12, p. 1349–1380, 2000. Disponível em: <<https://doi.org/10.1109/34.895972>>. Citado na página 24.

TAO, Y.; PAPADIAS, D.; LIAN, X. Reverse knn search in arbitrary dimensionality. In: NASCIMENTO, M. A.; ÖZSU, M. T.; KOSSMANN, D.; MILLER, R. J.; BLAKELEY, J. A.; SCHIEFER, K. B. (Ed.). **(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004**. Morgan Kaufmann, 2004. p. 744–755. Disponível em: <<http://www.vldb.org/conf/2004/RS20P1.PDF>>. Citado na página 28.

TORRES, R. da S.; FALCÃO, A. X.; GONÇALVES, M. A.; PAPA, J. P.; ZHANG, B.; FAN, W.; FOX, E. A. A genetic programming framework for content-based image retrieval. **Pattern Recognition**, Elsevier, v. 42, n. 2, p. 283–292, 2009. Disponível em: <<https://doi.org/10.1016/j.patcog.2008.04.010>>. Citado na página 25.

TRAINA-JR, C.; MORIYAMA, A.; ROCHA, G. M. da; CORDEIRO, R. L. F.; CIFERRI, C. D. de A.; TRAINA, A. J. M. The similarql framework: similarity queries in plain SQL. In: **Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019**. ACM, 2019. p. 468–471. Disponível em: <<https://doi.org/10.1145/3297280.3299736>>. Citado nas páginas 24, 28 e 89.

TRAINA-JR, C.; TRAINA, A. J. M.; FALOUTSOS, C.; SEEGER, B. Fast indexing and visualization of metric data sets using slim-trees. **IEEE Trans. Knowl. Data Eng.**, v. 14, n. 2, p. 244–260, 2002. Disponível em: <<https://doi.org/10.1109/69.991715>>. Citado nas páginas 19, 36, 37, 38, 39 e 55.

TRAINA-JR, C.; TRAINA, A. J. M.; FILHO, R. F. S.; FALOUTSOS, C. How to improve the pruning ability of dynamic metric access methods. In: **Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 4-9, 2002**. ACM, 2002. p. 219–226. Disponível em: <<https://doi.org/10.1145/584792.584831>>. Citado na página 19.

TRAN, Q. T.; TANIAR, D.; SAFAR, M. Reverse k nearest neighbor and reverse farthest neighbor search on spatial networks. **Trans. Large Scale Data Knowl. Centered Syst.**, v. 1, p. 353–372, 2009. Disponível em: <https://doi.org/10.1007/978-3-642-03722-1_14>. Citado na página 28.

URIBE, R.; NAVARRO, G.; BARRIENTOS, R. J.; MARÍN, M. An index data structure for searching in metric space databases. In: ALEXANDROV, V. N.; ALBADA, G. D. van; SLOOT, P. M. A.; DONGARRA, J. J. (Ed.). **Computational Science - ICCS 2006, 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part I**. Springer, 2006. (Lecture Notes in Computer Science, v. 3991), p. 611–617. Disponível em: <https://doi.org/10.1007/11758501_82>. Citado nas páginas 19 e 36.

VIEIRA, M. R.; TRAINA-JR., C.; CHINO, F. J. T.; TRAINA, A. J. M. Dbm-tree: A dynamic metric access method sensitive to local density data. **J. Inf. Data Manag.**, v. 1, n. 1, p. 111–128, 2010. Disponível em: <<http://seer.lcc.ufmg.br/index.php/jidm/article/view/22>>. Citado na página 32.

YIANILOS, P. N. Data structures and algorithms for nearest neighbor search in general metric spaces. In: RAMACHANDRAN, V. (Ed.). **Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA**. ACM-SIAM, 1993. p. 311–321. Disponível em: <<http://dl.acm.org/citation.cfm?id=313559.313789>>. Citado na página 33.

YU, M.; LI, G.; DENG, D.; FENG, J. String similarity search and join: a survey. **Frontiers of Computer Science**, Springer, v. 10, n. 3, p. 399–417, 2016. Disponível em: <<https://doi.org/10.1007/s11704-015-5900-5>>. Citado na página 25.

ZEZULA, P.; AMATO, G.; DOHNAL, V.; BATKO, M. **Similarity search: the metric space approach**. Kluwer, 2006. v. 32. ISBN 978-0-387-29146-8. Disponível em: <<https://doi.org/10.1007/0-387-29151-2>>. Citado nas páginas 17, 24, 28, 32 e 35.

ZHOU, X.; WANG, G.; YU, J. X.; YU, G. M+-tree : A new dynamical multidimensional index for metric spaces. In: SCHEWE, K.; ZHOU, X. (Ed.). **Database Technologies 2003, Proceedings of the 14th Australasian Database Conference, ADC 2003, Adelaide, South Australia, February 2003**. Australian Computer Society, 2003. (CRPIT, v. 17), p. 161–168. Disponível em: <<http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV17Zhou.html>>. Citado nas páginas 19, 36, 39 e 40.

ZHOU, X.; WANG, G.; ZHOU, X.; YU, G. Bm⁺-tree: A hyperplane-based index method for high-dimensional metric spaces. In: ZHOU, L.; OOI, B. C.; MENG, X. (Ed.). **Database Systems for Advanced Applications, 10th International Conference, DASFAA 2005, Beijing, China, April 17-20, 2005, Proceedings**. Springer, 2005. (Lecture Notes in Computer Science, v. 3453), p. 398–409. Disponível em: <https://doi.org/10.1007/11408079_36>. Citado nas páginas 19, 33, 36 e 41.