
Estruturas de indexação métricas em operações distribuídas de agrupamento por similaridade em dados de alta dimensionalidade

Ana Paula Cassiano Alves da Silva



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Uberlândia
2024

Ana Paula Cassiano Alves da Silva

**Estruturas de indexação métricas em operações
distribuídas de agrupamento por similaridade
em dados de alta dimensionalidade**

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

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. Humberto Luiz Razente

Uberlândia
2024

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

S586
2024

Silva, Ana Paula Cassiano Alves da, 1977-
Estruturas de indexação métricas em operações
distribuídas de agrupamento por similaridade em dados de
alta dimensionalidade [recurso eletrônico] / Ana Paula
Cassiano Alves da Silva. - 2024.

Orientador: Humberto Luiz Razente.
Dissertação (Mestrado) - Universidade Federal de
Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: <http://doi.org/10.14393/ufu.di.2024.619>
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Razente, Humberto Luiz, 1977-,
(Orient.). II. Universidade Federal de Uberlândia. Pós-
graduação em Ciência da Computação. III. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Nunes de Ávila, 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br



ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Dissertação, 28/2024, PPGCO				
Data:	26 de agosto de 2024	Hora de início:	09:00	Hora de encerramento:	11:25
Matrícula do Discente:	12112CCP002				
Nome do Discente:	Ana Paula Cassiano Alves da Silva				
Título do Trabalho:	Estruturas de indexação métricas em operações distribuídas de agrupamento por similaridade em dados de alta dimensionalidade				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Ciência de Dados				
Projeto de Pesquisa de vinculação:	-----				

Reuniu-se por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Maurício Cunha Escarpinati - FACOM/UFU, Elaine Parros Machado de Sousa - ICMC/USP e Humberto Luiz Razente - FACOM/UFU, orientador do candidato.

Os examinadores participaram desde as seguintes localidades: Elaine Parros Machado de Sousa - São Carlos/SP. Os outros membros da banca e o aluno participaram da cidade de Uberlândia.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Humberto Luiz Razente, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir ao candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.



Documento assinado eletronicamente por **Humberto Luiz Razente, Professor(a) do Magistério Superior**, em 27/08/2024, às 09:36, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Elaine Parros Machado de Sousa, Usuário Externo**, em 27/08/2024, às 09:42, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Mauricio Cunha Escarpinati, Professor(a) do Magistério Superior**, em 29/08/2024, às 18:04, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5624811** e o código CRC **BDFF5D9B**.

Dedico este trabalho a Deus, Criador, Formador e Arquiteto de todas as coisas. A Ele, que é a fonte de toda existência e inspiração, minha eterna gratidão, pois sem Ele nada seria possível.

Agradecimentos

À Deus, pela força e sabedoria concedidas ao longo desta jornada. Sem Sua graça, essa vitória em minha vida não seria possível.

Ao meu marido, Raylson, cujo amor e paciência me sustentaram durante todo o processo. Por compreender muitas vezes a minha ausência e acreditar em mim, me incentivando muitas vezes quando já pensava em desistir.

Aos meus pais, Laerte e Neusa, por me amarem incondicionalmente e entenderem a minha ausência. Mesmo longe, sempre torcendo pela minha vitória.

Ao meu orientador, Prof. Humberto Razente, pela sabedoria, paciência e incentivo. Por acreditar que era possível, mesmo diante de muitas adversidades. Este trabalho é uma prova do seu impacto positivo em minha trajetória acadêmica.

À minha grande amiga e irmã Maria por, mesmo sem muitas vezes entender o tamanho da dificuldade, sempre me ajudar com palavras de incentivo e orações que foram essenciais.

*“Confia no Senhor de todo o teu coração, e não te estribes no teu próprio entendimento.
Reconhece-o em todos os teus caminhos, e ele endireitará as tuas veredas.”
(Provérbios 3:5-6)*

Resumo

A prevalência de *Big Data* apresenta desafios significativos para a extração de conhecimento de grandes volumes de dados complexos. A análise de agrupamentos, crucial para identificar padrões e similaridades, utiliza técnicas como a busca e junção por similaridade, essencial para consultas baseadas em relações intrínsecas dos dados. No entanto, a alta dimensionalidade e o volume massivo de dados tornam essas operações computacionalmente caras. Sistemas distribuídos, como Apache Hadoop e Spark, têm sido implementados para melhorar o desempenho dessas análises. Técnicas de particionamento e poda, juntamente com métodos baseados em dissimilaridade, como funções de distância, são fundamentais para otimizar a manipulação dos dados. Recentemente, o operador *Similarity Group-By* (SGB) e sua evolução, o *Distributed Similarity Grouping* (DSG), têm mostrado avanços significativos, permitindo a computação de agrupamentos por similaridade em plataformas distribuídas. Contudo, a crescente demanda por análises mais rápidas e precisas requer melhorias contínuas. Neste contexto, propõe-se o operador *Distributed Similarity Grouping with Vantage Point Tree* (DSG-VPTree), uma inovação que integra a estrutura de dados *Vantage Point Tree* (VP-Tree) ao operador DSG, visando um particionamento mais eficiente. Este trabalho detalha a implementação do DSG-VPTree no ambiente Spark, avaliando seu desempenho em termos de tempo de execução do algoritmo, comparativamente ao operador DSG, demonstrando sua eficiência na superação das limitações de escalabilidade das soluções anteriores. A proposta oferece uma solução eficiente para operações de busca por similaridade em grandes volumes de dados, contribuindo para a evolução das técnicas de análise em *Big Data*. Os experimentos demonstram que o DSG-VPTree supera o DSG em 40%, apresentando menor tempo de execução e melhor escalabilidade em dados de alta dimensionalidade.

Palavras-chave: Busca por similaridade. Particionamento de Dados. Alta Dimensionalidade. Avaliação de desempenho. Map/Reduce. Apache Spark. Vantage-Point Tree. Método de Acesso Métrico.

Abstract

The prevalence of *Big Data* presents significant challenges for extracting knowledge from large volumes of complex data. Cluster analysis, crucial for identifying patterns and similarities, uses techniques such as similarity search and join, essential for queries based on intrinsic data relationships. However, the high dimensionality and massive volume of data make these operations computationally expensive. Distributed systems, such as Apache Hadoop and Spark, have been implemented to improve the performance of these analyses. Partitioning and pruning techniques, along with dissimilarity-based methods such as distance functions, are key to optimizing data manipulation. Recently, the SGB operator and its evolution, the DSG, have shown significant advances, allowing clusters to be computed by similarity on distributed platforms. However, the growing demand for faster and more accurate analysis requires continuous improvements. In this context, we propose DSG-VPtree operator, an innovation that integrates the VP-Tree data structure with the DSG operator, aiming for a more efficient and balanced partitioning. This work details the implementation of the DSG-VPtree in the Spark environment, evaluating its performance in terms of execution time compared to the DSG operator, demonstrating its efficiency in overcoming the scalability limitations of previous solutions. The proposal offers an efficient solution for similarity search operations on large volumes of data, contributing to the evolution of analysis techniques in *Big Data*. The experiments show that the DSG-VPtree outperforms the DSG by 40%, with shorter execution times and better scalability on high-dimensional data.

Keywords: Similarity Search. Data Partitioning. High Dimensionality. Performance Evaluation. Map/Reduce. Apache Spark. Vantage-Point Tree. Metric Access Method.

Lista de ilustrações

Figura 1 – Consultas por similaridade em conjunto bidimensional onde o objeto O_q é o objeto de consulta. (a) Busca por abrangência com raio de busca r_q . (b) Busca por k-vizinhos mais próximos com $k=3$. Fonte: Figura do autor.	36
Figura 2 – Particionamentos para conjunto de dados em duas dimensões. (a) Hiperplano generalizado, onde p_1 e p_2 representam os pontos de referência (pivôs) que definem o hiperplano. (b) Bola, onde S_1 e S_2 representam os conjuntos de dados, O_q representa o objeto de consulta e r_q o raio de busca. Fonte: Adaptado de (SAMET, 2006)	38
Figura 3 – Estrutura de uma VP-Tree. Adaptado de (YIANILOS, 1993)	39
Figura 4 – Ilustração da escolha de pivôs aleatória, passo inicial do algoritmo para iniciar o particionamento de dados. Fonte: Figura do autor.	48
Figura 5 – Particionamento e agrupamento por similaridade pelo DSG usando dois pivôs. Fonte: (SILVA et al., 2019)	48
Figura 6 – Particionamento dos dados pelo pivô mais próximo e por janelamento (epsilon).(a) Conjunto de dados inicial onde alguns elementos foram destacados representando o elemento de busca para encontrar o pivô mais próximo. (b) Conjunto com os pivôs escolhidos de forma aleatória e o particionamento feito a partir da distância do elemento de busca para o elemento de pivô mais próximo. (c) Após a primeira busca pelo pivô mais próximo, é feita a verificação de distância para os demais elementos de pivô, buscando distância dentro de ϵ que representa elemento dentro da janela de outro pivô. Fonte: Figura do autor.	56
Figura 7 – Distribuição de dados em uma VP-Tree. Adaptado de (FU et al., 2000)	58
Figura 8 – Particionamento do conjunto de dados através da utilização da VP-Tree para encontrar o vizinho (pivô) mais próximo. Fonte: Figura do autor.	59

Figura 9 – Particionamento do conjunto de dados através da utilização da VP-Tree para identificar pontos que pertencem às janelas de outros pivôs. (a) Verificação dos pivôs dentro do range definido por Dring para reduzir as próximas verificações. (b) Verificação de pontos dentro da janela de outro pivô a partir de um número menor de pivôs. Fonte: Figura do autor.	59
Figura 10 – Cálculo de distância para poda de busca por abrangência. Fonte: Figura do autor.	60
Figura 11 – Tempo de execução com o aumento de tamanho do conjunto de dados. Fonte: Figura do autor.	69
Figura 12 – Tempo de execução com o aumento de número de dimensões. Fonte: Figura do autor.	70
Figura 13 – Tempo de execução com aumento de número de pivôs. Fonte: Figura do autor.	71

Lista de tabelas

Tabela 1	–	Identificação dos Grupos de Similaridade na Partição P0. Fonte: (SILVA et al., 2019)	49
Tabela 2	–	Identificação dos Grupos de Similaridade na Partição P1. Fonte: (SILVA et al., 2019)	49
Tabela 3	–	Resultados do Experimento: número de grupos identificados com 64 dimensões e com o aumento do fator de escala. Fonte: Tabela do autor.	67
Tabela 4	–	Resultados do Experimento: número de grupos identificados com SF10, com o aumento do número de dimensões. Fonte: Tabela do autor.	67
Tabela 5	–	Resultados do Experimento: número de grupos identificados com SF10 e 64 dimensões, com o aumento do número de pivôs. Fonte: Tabela do autor.	68
Tabela 6	–	Resultados do Experimento: número de grupos identificados com SF10 e 192 dimensões, com o aumento do número de pivôs. Fonte: Tabela do autor.	68

Lista de algoritmos

1	DistSimGrouping	57
2	DistSimGroupingVpTree	58
3	GetClosestPivotVpTree	60
4	Construção da VP-Tree	61
5	Busca na VP-Tree	62
6	Busca por abrangência definida na VP-Tree	64

Lista de siglas

DSG *Distributed Similarity Grouping*

DSG-VPTree *Distributed Similarity Grouping with Vantage Point Tree*

DSJ *Diversity Similarity Join*

GH-Tree *Generalized Hyperplane Tree*

HDFS *Hadoop Distributed File System*

HF *Hull of Foci*

KNN *K-Nearest Neighbors*

M-Tree *Metric Tree*

MAM *Metric Access Methods*

MVP-Tree *Multi Vantage-Point Tree*

RDD *Resilient Distributed Datasets*

SGB *Similarity Group-By*

VP-Tree *Vantage Point Tree*

YARN *Yet Another Resource Negotiator*

Sumário

1	INTRODUÇÃO	25
1.1	Motivação	27
1.2	Objetivos e Desafios da Pesquisa	28
1.3	Hipótese	29
1.4	Contribuições	29
1.5	Organização da Dissertação	29
2	FUNDAMENTAÇÃO TEÓRICA	31
2.1	Introdução à Big Data e Desafios de Análise de dados	31
2.2	Análise de Agrupamentos em Big Data	32
2.3	Sistemas Distribuídos para Processamento de Dados Massivos	33
2.3.1	Paradigma MapReduce	33
2.3.2	Plataformas Apache Hadoop e Spark	34
2.3.3	Vantagens do processamento distribuído para Big Data	34
2.4	Manipulação de dados por Similaridade	35
2.4.1	Buscas por Similaridade	35
2.5	Espaços Métricos e Funções de Distância	36
2.5.1	Funções de Distância	37
2.6	Métodos de Acesso Métricos	37
2.6.1	<i>Vantage-Point Tree</i>	39
2.6.2	<i>Multi Vantage-Point Tree</i>	39
2.6.3	M-Tree	40
2.6.4	<i>Generalized Hyperplane Tree</i>	40
2.6.5	Comparação entre as Estruturas de dados	41
2.6.6	Aplicações e Benefícios dos MAMs	43
2.7	Técnicas de Particionamento de Dados	44
2.8	Operadores de Agrupamento por Similaridade em <i>Big Data</i> . .	46
2.8.1	Similarity Group-By (SGB)	46

2.8.2	Distributed Similarity Grouping (DSG)	47
2.8.3	Benefícios e limitações do DSG	50
2.9	Trabalhos Correlatos	51
2.10	Considerações	52
3	ALGORITMO DE AGRUPAMENTO POR SIMILARIDADE EM OPERAÇÕES DISTRIBUÍDAS COM VANTAGE POINT TREE (DSG-VPTREE)	53
3.1	Integração da Vantage Point Tree ao Método DSG	54
3.1.1	Benefícios esperados da VP-Tree em comparação com outras estruturas	54
3.1.2	Metodologia do DSG-VPtree	55
3.1.3	Impactos na eficiência e desempenho das operações de consulta por si- milaridade	55
3.2	Detalhes da Solução Proposta	56
3.3	Construção da VP-Tree	61
3.4	Busca na VP-Tree pelo vizinho mais próximo	62
3.5	Busca por abrangência na VP-Tree	63
3.6	Considerações Finais	63
4	EXPERIMENTOS E ANÁLISE DOS RESULTADOS	65
4.1	Método para a Avaliação	65
4.2	Experimentos	66
4.2.1	Validação da formação de Grupos de Similaridade	67
4.2.2	Validação do Tempo de Processamento	69
4.3	Avaliação dos Resultados	72
4.4	Considerações Finais	72
5	CONCLUSÃO	73
5.1	Principais Contribuições	74
5.2	Trabalhos Futuros	75
5.3	Contribuições em Produção Bibliográfica	76
	REFERÊNCIAS	77

Introdução

No cenário atual, caracterizado pela prevalência de *Big Data*, organizações de diversas áreas enfrentam o desafio constante de extrair conhecimento relevante de conjuntos de dados de tamanho e complexidade sem precedentes. A análise eficaz desses dados não é apenas uma vantagem competitiva, mas uma necessidade para a inovação, a tomada de decisões estratégicas, a manutenção da relevância no mercado. Entre as várias técnicas empregadas para analisar esses dados, a análise de agrupamentos surge como um método fundamental para identificar padrões e grupos de similaridade que facilitam a interpretação de grandes volumes de informações (SILVA et al., 2019).

Buscas por similaridade retornam os dados baseados em suas relações de similaridade, que são derivadas dos vetores de características do conteúdo intrínseco aos dados. A busca por similaridade pode ser feita por abrangência, onde buscam-se os elementos mais próximos a partir de um limiar ou busca dos K vizinhos mais próximos, onde buscam-se os k-vizinhos mais próximos do elemento de pesquisa. Basicamente, essas consultas são feitas a partir de uma função de distância que calcula o quão similar um elemento (vetor de características) é em relação aos demais (ZEZULA et al., 2006).

Porém, quando tratamos de grandes volumes de dados complexos, busca por similaridade, apesar de ser extremamente útil, ainda é uma operação de alto custo computacional para análise de dados (CHEN et al., 2023; ZEZULA et al., 2006). Outro fator muito importante nessa avaliação é a dimensionalidade dos dados. Segundo ZEZULA et al. (2006), em volumes de dados massivos e de alta dimensionalidade, para tornar a busca por similaridade viável, é preciso utilizar técnicas de particionamento de dados e poda para garantir melhor desempenho nessas consultas.

Existem diversas técnicas de particionamento de dados em espaço métrico desenvolvidos nas últimas décadas (CHEN et al., 2023; ZEZULA et al., 2006) que permitem o processamento de buscas baseadas na similaridade entre elementos. Na maioria dos métodos, a utilização de pivôs como representantes, é um ponto chave para melhorar o desempenho dessas consultas.

O uso de sistemas distribuídos para processamento de dados massivos tem ganhado

popularidade em diversas aplicações, por exemplo, com o uso do paradigma Map/Reduce (DEAN; GHEMAWAT, 2004) implementado no *Apache Hadoop* e *Spark* (ZAHARIA et al., 2016). O sistema é composto de um serviço de processamento e sistema de arquivos distribuído em um cluster formado por computadores interligados em rede. São beneficiados os algoritmos para os quais os dados possam ser particionados para distribuição, onde o processamento é realizado nas partições de modo distribuído, e por fim, realiza-se a agregação dos resultados. O critério de particionamento é geralmente baseado na relação de ordem e a igualdade de chaves.

Para dados em que a relação de ordem e a igualdade entre elementos tem pouca utilidade, utiliza-se uma noção de dissimilaridade, normalmente modelada como uma função de distância, que quantifica o quão dissimilar dois elementos são. Por exemplo, se o usuário tem uma imagem, recuperar uma imagem idêntica tem pouca utilidade quando comparado com a possibilidade de recuperar imagens similares.

O particionamento de conjuntos de dados baseado em dissimilaridade é uma tarefa essencial para manipulação otimizada desses dados, por exemplo, com a criação de métodos de acesso métricos dinâmicos (TRAINA-JR. et al., 2002). Em (CHEN et al., 2023; SAMET, 2006; ZEZULA et al., 2006) são apresentadas extensas revisões sobre métodos de particionamento baseados em bolas e baseados em hiperplanos separadores. Esses métodos buscam a organização dos dados em agrupamentos baseados na similaridade, criando hierarquias balanceadas dinamicamente. Para o particionamento de dados estáticos são empregados elementos de referência selecionados previamente (pivôs).

Nos últimos anos, pesquisas nesse domínio têm avançado significativamente, com destaque para o operador SGB (SILVA; AREF; ALI, 2009; TANG et al., 2016), que permite a computação de agrupamentos por similaridade em consultas *ad hoc*; e para o operador DSG (SILVA et al., 2019), que opera sobre plataformas de computação distribuída *Apache Hadoop* e *Spark*. Através da utilização do paradigma *MapReduce*, o DSG consegue identificar grupos similares em conjuntos de dados massivos de maneira paralela, otimizando significativamente o tempo de processamento necessário para tal tarefa.

Contudo, apesar do progresso proporcionado pelo DSG, o crescimento incessante do volume de dados e a exigência por análises cada vez mais ágeis e precisas evidenciam a necessidade de contínuo aprimoramento das soluções existentes. Nesse contexto, este trabalho propõe o operador DSG-VPTree, uma inovação do método DSG que incorpora a estrutura de dados VP-Tree para otimizar o particionamento de dados. Esta abordagem visa oferecer um método mais eficiente para operações de busca por similaridade, mitigando assim as limitações de escalabilidade e desempenho no tempo de execução das soluções anteriores.

Este trabalho detalha a criação do DSG-VPTree no ambiente *Spark*, fornecendo diretrizes claras para sua aplicação e avalia seu desempenho em comparação com o operador DSG, que teve vantagem comprovada na comparação com outros métodos de agrupa-

mento de dados. A escolha do *Spark* justifica-se pelo desempenho superior na execução de operações em lote distribuídas, resultado da sua arquitetura baseada em *Resilient Distributed Datasets* (RDD), uma abstração do armazenamento distribuído, tolerante a falhas e que simplifica a manipulação dos dados em paralelo (STONEBRAKER; PAVLO, 2024).

1.1 Motivação

O crescimento constante na disponibilidade de dados e o aumento na complexidade desses conjuntos representam desafios para algoritmos de busca por similaridade, especialmente em contextos de Big Data (ABADI et al., 2022). Embora técnicas distribuídas, como o método DSG (SILVA et al., 2019), tenham avançado significativamente ao longo dos últimos anos, gargalos permanecem, comprometendo a escalabilidade e a eficiência das operações de agrupamento em grandes volumes de dados.

Um dos principais desafios enfrentados pelo DSG está no particionamento eficiente de dados e na necessidade de replicação de elementos entre as fronteiras das partições. Essa replicação, embora necessária para garantir a precisão das buscas por similaridade, introduz uma complexidade adicional que impacta diretamente o tempo de execução do operador. Além disso, a etapa de agrupamento entre os pivôs representa um ponto de alta sobrecarga, devido à alta quantidade de cálculos, limitando o potencial do operador para aplicações em cenários de dados massivos e de alta dimensionalidade.

Diante dessas limitações, torna-se essencial o desenvolvimento de soluções mais eficientes que sejam capazes de reduzir o custo computacional e melhorar o desempenho em termos de tempo de processamento do particionamento e das buscas por similaridade. A integração da estrutura de dados VP-Tree ao DSG surge como uma abordagem eficiente e inovadora, visto que a VP-Tree tem demonstrado grande eficiência em organizar e recuperar dados em espaços métricos de alta dimensionalidade (YIANILOS, 1993). Essa abordagem, além de melhorar o desempenho do particionamento de dados, visa mitigar os gargalos existentes e oferecer uma solução mais escalável para a análise de dados.

Com essa perspectiva, este trabalho busca não apenas aprimorar as operações de busca por similaridade no contexto do DSG, mas também contribuir com o desenvolvimento de uma metodologia que possa servir como referência para futuras pesquisas em algoritmos de particionamento e análise de dados em ambientes distribuídos.

A próxima seção deste trabalho detalha os objetivos específicos do desenvolvimento do DSG-VP-Tree, destacando como esta nova abordagem pretende resolver os problemas identificados e contribuir para o avanço da análise de dados em *Big Data*.

1.2 Objetivos e Desafios da Pesquisa

O **objetivo geral** deste trabalho é aprimorar o desempenho das operações de agrupamento por similaridade em conjuntos de dados massivos de alta dimensionalidade, por meio da incorporação da estrutura de dados VP-Tree ao método DSG.

Para alcançar o objetivo geral proposto, alguns **objetivos específicos** foram definidos:

1. Contribuir com um novo algoritmo para particionamento de dados complexos com a utilização da estrutura de indexação métrica VP-Tree.
2. Investigar a eficácia e a eficiência da VP-Tree na organização e busca de dados em espaços métricos complexos e de alta dimensionalidade, no operador DSG, visando reduzir a complexidade de tempo das funções críticas, como o agrupamento em pivôs.
3. Comparar o desempenho e a precisão do operador proposto DSG-VPTree com o operador DSG, em termos de tempo de execução e integridade dos grupos de similaridade formados em diferentes condições de dimensionalidade e tamanho do conjunto de dados.
4. Melhorar o particionamento de dados no operador DSG, utilizando a VP-Tree, buscando mitigar as limitações de escalabilidade e desempenho das soluções anteriores.
5. Fornecer uma abordagem detalhada para a implementação do DSG-VPTree no ambiente Apache Spark, incluindo o processo de construção e distribuição da VP-Tree nos nós do *cluster*, bem como avaliando seu impacto na latência e eficiência das operações de busca.
6. Avaliar o impacto da integração da VP-Tree ao operador DSG, no desempenho das operações de busca e agrupamento por similaridade em diferentes condições de dimensionalidade e tamanho do conjunto de dados.

Para abordar cada objetivo, a investigação foi dividida em três perguntas de pesquisa, conforme segue:

- ❑ Como a VP-Tree pode ser integrada ao método DSG para melhorar o tempo de processamento das operações de busca por similaridade?
- ❑ A utilização da VP-Tree realmente melhora o desempenho do DSG em termos de tempo de execução e uso de recursos computacionais?
- ❑ Como deve ser a implementação do DSG-VPTree no ambiente Apache Spark?

Ao abordar esses objetivos, este trabalho visa contribuir significativamente para o avanço do estado da arte na preparação dos dados para análises em *Big Data*, oferecendo uma solução eficiente para operações de agrupamento por similaridade.

1.3 Hipótese

A hipótese que motivou o desenvolvimento do trabalho apresentado nessa dissertação discorre sobre: "A inclusão da estrutura de indexação métrica VP-Tree no particionamento dos dados do operador DSG, por meio de um algoritmo de custo linear, pode melhorar significativamente o desempenho das operações de busca por similaridade, reduzindo o tempo de computação necessário para o processamento de operações de agrupamento por similaridade".

1.4 Contribuições

As principais contribuições trazidas pelo presente trabalho são:

- ❑ Oferecer uma estratégia mais eficiente de particionamento de dados complexos, que produza resultados mais rápidos para buscas por similaridade;
- ❑ Otimizar a análise das janelas de sobreposição das partições, melhorando a precisão e a eficiência das operações de busca;
- ❑ Reduzir a complexidade computacional das operações de agrupamento em pivôs;
- ❑ Fornecer abordagem detalhada para a implementação do operador no ambiente *Apache Spark*;
- ❑ Validar a eficiência do novo operador, por meio de uma análise de desempenho abrangente, que demonstra melhorias significativas em cenários de alta dimensionalidade.

1.5 Organização da Dissertação

O trabalho está organizado em cinco capítulos conforme descrito a seguir:

- ❑ **Capítulo 2.** Apresenta o arcabouço teórico do trabalho, trazendo fundamentação sobre Big Data, análise de agrupamentos, similaridade, espaços métricos e seus métodos de acesso, técnicas de particionamento de dados, sistemas distribuídos e operadores de agrupamento por similaridade. São apresentados os métodos de acessos métricos considerados durante a pesquisa, demonstrando o motivo teórico para

escolha da VP-Tree. Também é feita uma contextualização sobre o algoritmo principal e a integração da VP-Tree ao DSG. Nesse capítulo também são apresentados os trabalhos relacionados à pesquisa na literatura.

- ❑ **Capítulo 3.** Apresenta a nova metodologia proposta, contextualizando primeiramente a limitação existente no algoritmo principal. Nesse capítulo, são detalhados os ajustes necessários para a integração da VP-Tree ao DSG.
- ❑ **Capítulo 4.** Apresenta o detalhamento dos experimentos realizados com seus resultados e análise de cada situação proposta.
- ❑ **Capítulo 5.** Apresenta as principais contribuições alcançadas com a pesquisa, algumas limitações, sugestões para trabalhos futuros e contribuições em produção bibliográfica.

Fundamentação Teórica

Nesse capítulo são apresentados os fundamentos teóricos considerados para a pesquisa realizada. Além de uma breve revisão da literatura, são apresentados também os principais trabalhos correlatos que norteiam esta pesquisa. Esse capítulo está organizado como segue: (2.1) apresenta uma conceituação sobre Big Data e discorre sobre os desafios enfrentados em análise de dados massivos; (2.2) faz uma análise sobre agrupamentos em grandes conjuntos de dados; (2.3) apresenta fundamentos sobre processamento de dados massivos em sistemas distribuídos; (2.4) apresenta conceitos sobre similaridade e consultas por similaridade; (2.5) discorre sobre espaços métricos e funções de distância, priorizando a família de distância Minkowski; (2.6) expõe a definição de Métodos de Acesso Métricos, destacando os métodos alvos da pesquisa; (2.7) apresenta técnicas de particionamento de dados; (2.8) apresenta dois operadores de agrupamento por similaridade. Finalizando, (2.9) apresenta os principais trabalhos correlatos que baseiam a pesquisa e (2.10) traz um consolidado com as considerações finais sobre a fundamentação teórica utilizada para o trabalho.

2.1 Introdução à Big Data e Desafios de Análise de dados

Dados massivos ou *Big Data* refere-se ao grande volume de dados que é gerado continuamente a partir de diversas fontes, incluindo redes sociais, sensores, transações comerciais e mais. A característica essencial de *Big Data* não é apenas a quantidade de dados, mas também a variedade, velocidade e veracidade desses dados, frequentemente descritas como os “4Vs” do *Big Data* (*volume, variety, velocity e veracity*). O armazenamento, processamento e análise de tais volumes massivos de dados requerem abordagens e tecnologias avançadas que possam lidar com a complexidade e o dinamismo dos dados (CHEN; MAO; LIU, 2014).

A análise de grandes volumes de dados é crucial para extrair informações valiosas e

insights que podem apoiar tomada de decisões. As organizações podem usar *Big Data* para entender melhor o comportamento dos consumidores, otimizar operações, detectar fraudes, melhorar a saúde pública e muito mais. O valor de *Big Data* reside na capacidade de transformar dados brutos em conhecimento acionável, o que pode proporcionar vantagens competitivas e inovações significativas (CHEN; MAO; LIU, 2014).

A extração de conhecimento de dados massivos e complexos apresenta vários desafios significativos. Esses desafios incluem a gestão e integração de dados de diferentes fontes, a garantia da qualidade e precisão dos dados, e a necessidade de ferramentas e técnicas avançadas de análise que possam processar e interpretar dados em tempo real (DADVAR; GOLAB; SRIVASTAVA, 2022).

A escalabilidade dos algoritmos e sistemas também é um desafio crítico, pois eles devem ser capazes de lidar eficientemente com o crescimento contínuo dos dados. Além disso, questões de privacidade e segurança dos dados são de extrema importância, considerando a sensibilidade e o potencial de uso indevido dos dados (SILVA et al., 2023).

Os desafios técnicos são amplamente explorados por meio de abordagens como o processamento distribuído, que divide as tarefas de processamento de dados entre vários sistemas para aumentar a eficiência e reduzir o tempo de processamento. Ferramentas como *Apache Hadoop* e *Apache Spark* são exemplos de plataformas que facilitam o processamento distribuído e são amplamente utilizadas para gerenciar e analisar dados massivos (ZAHARIA et al., 2016).

2.2 Análise de Agrupamentos em Big Data

A análise de agrupamentos, também conhecida como *clustering*, é uma técnica crucial em *Big Data* que visa identificar e agrupar elementos semelhantes dentro de um conjunto de dados. Esta técnica é amplamente utilizada para descobrir padrões ocultos, segmentar dados e simplificar a análise de grandes volumes de informações. Segundo JAIN (2010), a análise de agrupamentos tem sido uma área de pesquisa ativa por mais de 50 anos, refletindo sua importância contínua em diversos campos, incluindo mineração de dados, reconhecimento de padrões e aprendizado de máquina.

Existem vários métodos de agrupamento utilizados para lidar com grandes volumes de dados. Entre os mais populares estão o K-Means, CURE e DBSCAN. Cada um desses métodos possui suas características específicas, vantagens e limitações. Eles tem sido amplamente estudados nas últimas décadas, segundo (JAIN, 2010).

O K-Means é um dos algoritmos de agrupamento mais conhecidos e utilizados. Ele funciona dividindo o conjunto de dados em k clusters, onde k é um parâmetro definido pelo usuário. O algoritmo tenta minimizar a soma das distâncias quadradas entre os pontos de dados e os centroides dos clusters (JAIN, 2010; IKOTUN et al., 2023; ALASALI; ORTAKCI, 2024). Apesar de sua simplicidade e eficiência computacional, o K-Means

apresenta limitações, como a necessidade de definir previamente o número de clusters e a sensibilidade a pontos de dados *outliers*.

O CURE é um método de agrupamento projetado para lidar com a presença de *outliers* e a variabilidade de formas e tamanhos dos *clusters*. Ao contrário do K-Means, que usa um único centroide para representar cada *cluster*, o CURE seleciona múltiplos pontos representativos para melhor capturar a estrutura do *cluster* (GUHA; RASTOGI; SHIM, 2001; JAIN, 2010). Este método é particularmente útil em cenários onde os dados apresentam uma grande variabilidade e não seguem uma distribuição uniforme.

O DBSCAN é um algoritmo de agrupamento baseado em densidade que identifica *clusters* como regiões de alta densidade de pontos separadas por regiões de baixa densidade. Este método é eficiente para descobrir *clusters* de formas arbitrárias e é robusto a *outliers* (JAIN, 2010; ALASALI; ORTAKCI, 2024). No entanto, a performance do DBSCAN pode ser sensível à escolha dos parâmetros de densidade mínima e raio de vizinhança.

2.3 Sistemas Distribuídos para Processamento de Dados Massivos

O processamento distribuído refere-se à utilização de múltiplos computadores trabalhando em conjunto para realizar tarefas de processamento de dados. Essa abordagem é particularmente eficaz no contexto de *Big Data*, onde a quantidade de dados a ser processada é extremamente grande e complexa para ser gerenciada por um único sistema. Sistemas distribuídos permitem que os dados sejam divididos em partes menores e processados simultaneamente por vários nós, o que resulta em uma maior eficiência e velocidade de processamento (DEAN; GHEMAWAT, 2004).

2.3.1 Paradigma MapReduce

O paradigma MapReduce é um modelo de programação que facilita o processamento e a geração de grandes conjuntos de dados. Desenvolvido pelo Google, MapReduce divide a tarefa de processamento em duas fases principais: a fase *Map*, onde os dados de entrada são divididos em partes menores e processados em paralelo, e a fase *Reduce*, onde os resultados intermediários são agregados para produzir o resultado final (DEAN; GHEMAWAT, 2004). Uma visão geral dos sistemas de programação distribuída que utilizam o paradigma MapReduce pode ser encontrada em (LESKOVEC; RAJARAMAN; ULLMAN, 2014; LEE et al., 2011; LU; FENG, 2014).

Na fase *Map*, a função *map* é aplicada a cada elemento de dados, produzindo pares chave-valor intermediários. Esses pares são então organizados e enviados para a fase *Reduce*, onde a função *reduce* processa todos os valores associados a cada chave única, agregando os resultados. Esse modelo de programação é altamente escalável e eficiente,

tornando-se uma escolha comum para muitas aplicações de *Big Data* (DEAN; GHEMAWAT, 2004).

2.3.2 Plataformas Apache Hadoop e Spark

Duas das plataformas mais utilizadas para o processamento distribuído de dados massivos são *Apache Hadoop* e *Apache Spark* (VERMA; MANSURI; JAIN, 2016).

Apache Hadoop: *Hadoop* é uma plataforma de código aberto que implementa o modelo *MapReduce*. Ele inclui o *Hadoop Distributed File System* (HDFS), que permite o armazenamento distribuído de dados, e o *Yet Another Resource Negotiator* (YARN), um sistema de gerenciamento de recursos que facilita a execução de tarefas de processamento distribuído (DEAN; GHEMAWAT, 2004). A arquitetura do *Hadoop* é projetada para escalar de um único servidor para milhares de máquinas, cada uma oferecendo armazenamento e computação local.

Apache Spark: *Spark* é uma plataforma de processamento de dados em tempo real que melhora o modelo *MapReduce*, proporcionando uma performance até 100 vezes mais rápida para certas aplicações, graças ao seu uso eficiente da memória (ZAHARIA et al., 2016). *Spark* suporta uma ampla gama de operações de processamento de dados, incluindo consultas SQL, *streaming* de dados, aprendizado de máquina e processamento de gráficos. Sua capacidade de processar dados em memória torna-o particularmente eficaz para tarefas iterativas e interativas.

2.3.3 Vantagens do processamento distribuído para Big Data

O processamento distribuído oferece várias vantagens significativas para o gerenciamento e análise de *Big Data* (ZAHARIA et al., 2016):

- ❑ **Escalabilidade:** Sistemas distribuídos podem facilmente escalar horizontalmente, adicionando mais nós para lidar com volumes de dados crescentes. Isso é essencial para aplicações que precisam processar *petabytes* ou *exabytes* de dados.
- ❑ **Resiliência e Tolerância a Falhas:** Sistemas como *Hadoop* e *Spark* são projetados para serem resilientes, com mecanismos integrados para detecção e recuperação de falhas de hardware ou software. Isso garante que as tarefas de processamento possam continuar sem interrupções significativas.
- ❑ **Eficiência de Tempo e Custo:** Ao distribuir as tarefas de processamento entre múltiplos nós, sistemas distribuídos reduzem o tempo total de processamento. Além disso, o uso de hardware *commodity* em vez de supercomputadores caros torna o processamento de *Big Data* mais acessível economicamente.

- **Flexibilidade e Versatilidade:** Plataformas como *Spark* oferecem suporte para uma ampla gama de operações e aplicações, incluindo processamento em lote, *streaming* de dados, aprendizado de máquina e análise de gráficos, proporcionando uma solução flexível para diversas necessidades de processamento de dados.

2.4 Manipulação de dados por Similaridade

Para compreender melhor a similaridade, é preciso entender os conceitos de dados escalares e dados complexos. Dados escalares são aqueles que correspondem a um único valor que não pode sofrer decomposição e são comparados diretamente, a exemplo de dados textuais ou numéricos. Já os dados complexos, como imagens, vídeos, séries temporais, áudio, dados geográficos, necessitam de uma transformação preliminar para facilitar a comparação.

Os dados escalares contemplam tanto as propriedades de Relação de Identidade quanto de Relação de Ordem, garantindo a comparação dos dados por igualdade ($=$ e $\neq$) e a consulta por comparadores ($<$, $\leq$, $\geq$ e $>$). Os dados complexos, geralmente não atendem às propriedades da Relação de Ordem, não podendo ser consultados usando os operadores $<$, $\leq$, $\geq$ e $>$. Já no caso da Relação de Identidade, pode ser utilizada a igualdade para comparação de dados complexos, porém não existe muita significância da igualdade para esse tipo de dado (ZEZULA et al., 2006; SAMET, 2006; CHEN et al., 2023).

Para a comparação de dados complexos, surge a necessidade da utilização do conceito da similaridade. Porém, normalmente os dados complexos não são utilizados em seu formato original. São utilizadas técnicas de extração de características, transformando o dado complexo em vetor de característica que descreve o dado em nível semântico. Para imagens, por exemplo, os principais extratores de características são baseados em cor, textura e forma (LEW et al., 2006).

Dessa forma, para efetuar a comparação de dados complexos, geralmente utiliza-se o conceito de similaridade para quantificar a proximidade entre os objetos (ZEZULA et al., 2006). O cálculo da similaridade dos objetos é feito a partir de uma função que compara um par de objetos complexos e retorna um valor numérico, que quantifica a similaridade entre eles. Essa função pode ser uma função de similaridade ou uma função de distância. Enquanto a função de distância retorna valor menor para objetos mais similares, a função de similaridade retorna valor maior para objetos mais similares (ZEZULA et al., 2006).

2.4.1 Buscas por Similaridade

A busca por similaridade é uma técnica de recuperação de informações que se concentra em encontrar itens que sejam semelhantes a um item de consulta dentro de um conjunto de dados, conforme demonstrado na Figura 1. Este conceito é fundamental em muitas

áreas, como recuperação de imagens, mineração de textos, e sistemas de recomendação. A eficiência na busca por similaridade é essencial para a análise de grandes volumes de dados, uma vez que permite a descoberta rápida de padrões e correlações úteis (ZEZULA et al., 2006). Dentre as consultas por similaridade, as mais utilizadas são (ZEZULA et al., 2006):

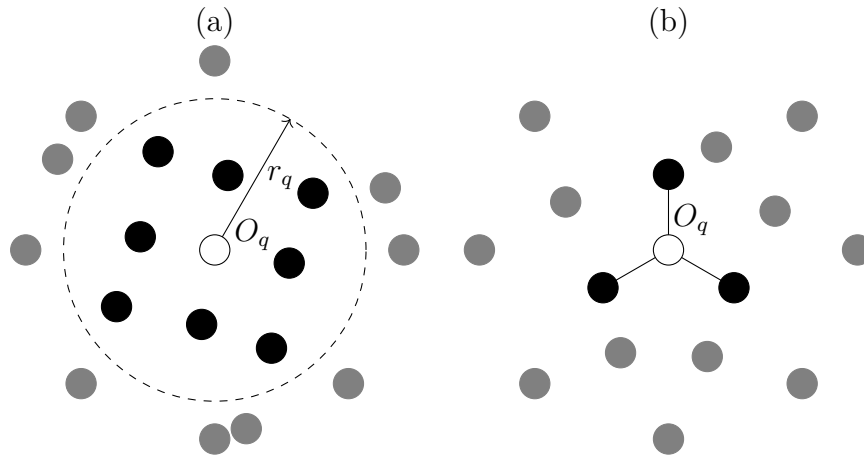


Figura 1 – Consultas por similaridade em conjunto bidimensional onde o objeto O_q é o objeto de consulta. (a) Busca por abrangência com raio de busca r_q . (b) Busca por k -vizinhos mais próximos com $k=3$. Fonte: Figura do autor.

- ❑ **Consulta por Abrangência:** a consulta por abrangência busca identificar todos os elementos similares ao elemento da consulta até no máximo um limiar pré-definido. Recupera todos os objetos que estão dentro de uma distância especificada r do objeto de consulta q . É particularmente útil em aplicações onde é necessário encontrar todos os elementos que estão próximos a um dado ponto, como na geolocalização.
- ❑ **Consulta aos k -vizinhos mais próximos:** a consulta aos k -vizinhos mais próximos identifica os k elementos mais similares ao elemento de consulta. O algoritmo *K-Nearest Neighbors* (KNN) é amplamente utilizado devido à sua simplicidade e eficácia em uma variedade de domínios.
- ❑ **Consulta do Vizinho Mais Próximo Reverso:** a consulta ao vizinho mais próximo reverso determina quais objetos na base de dados têm o objeto de consulta q como um de seus k vizinhos mais próximos.

2.5 Espaços Métricos e Funções de Distância

A relevância de utilizar espaços métricos é que eles permitem que regras de particionamento e poda sejam utilizadas. Essas regras são extremamente importantes no desenvolvimento de estruturas de índices (ZEZULA et al., 2006).

Formalmente, um espaço métrico M é definido como um par $M = \{S, d\}$, onde S é o domínio dos objetos e d é uma função de distância que indica a relação de similaridade entre os elementos de S . Essa função calcula o quão dissimilar um par de objetos é e retorna um valor real positivo, onde quanto mais próximo de zero, mais similar é o par de objetos (ZEZULA et al., 2006).

2.5.1 Funções de Distância

As funções de distância ou métricas podem ser definidas como uma função de cálculo de dissimilaridade entre pares de elementos do conjunto ao qual está associada. Neste modelo, quanto mais similares dois elementos entre si, menor o valor de retorno dessa função. As funções de distância são indispensáveis para a busca por similaridade, pois elas definem como a similaridade entre os itens é medida. A escolha da função de distância pode impactar significativamente a semântica, a precisão e a eficiência da busca por similaridade (SAMET, 2006).

Dado o conjunto dos objetos S e os elementos $s_1, s_2, s_3 \in S$, para que uma função de distância seja considerada uma métrica d , deve obedecer aos seguintes critérios (ZEZULA et al., 2006):

- **não-negatividade:** $0 < d(s_1, s_2) < \infty$, se $s_1 \neq s_2$
- **identidade:** $d(s_1, s_2) = 0$, se $s_1 = s_2$
- **simetria:** $d(s_1, s_2) = d(s_2, s_1)$
- **desigualdade triangular:** $d(s_1, s_2) \leq d(s_1, s_3) + d(s_3, s_2)$

Dentre as funções de distância mais utilizadas, estão as da família Minkowski, entre elas a Euclidiana e a Manhathan (ZEZULA et al., 2006). As funções Minkowski são específicas para vetores de números reais n-dimensionais. São chamadas métricas L_p pois suas normas dependem de um parâmetro numérico p (ZEZULA et al., 2006).

2.6 Métodos de Acesso Métricos

Os Metric Access Methods (MAM) são estruturas de dados especializadas para indexar dados complexos em espaços métricos. Eles exploram propriedades métricas como a desigualdade triangular para possibilitar acesso eficiente aos dados, efetuando a poda de subconjuntos do espaço de busca que certamente não tem elementos que farão parte do conjunto resposta. Eles são baseados nas distâncias entre elementos e podem ser aproveitados para reduzir o número de operações de comparação, podendo gerar um melhor desempenho na execução de consultas por similaridade.

Os índices utilizados nos MAM são geralmente estruturados em forma de árvores balanceadas e efetuam o particionamento dos conjuntos de dados em subconjuntos. O particionamento em subconjuntos pode ser feito por bolas ou por hiperplano generalizado (SAMET, 2006), conforme apresentado na Figura 2.

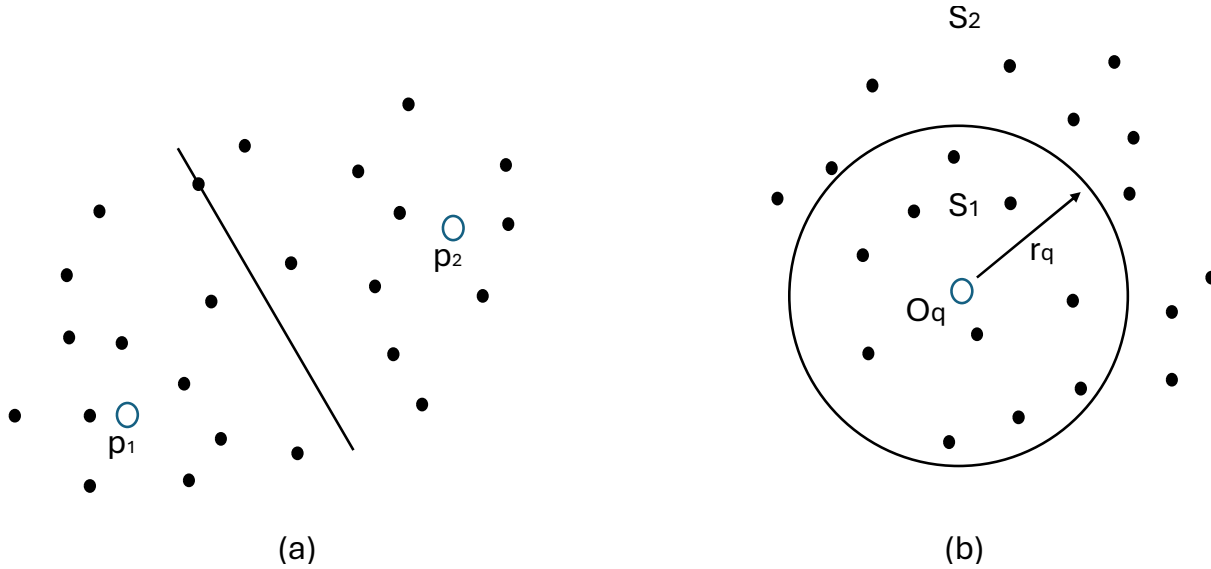


Figura 2 – Particionamentos para conjunto de dados em duas dimensões. (a) Hiperplano generalizado, onde p_1 e p_2 representam os pontos de referência (pivôs) que definem o hiperplano. (b) Bola, onde S_1 e S_2 representam os conjuntos de dados, O_q representa o objeto de consulta e r_q o raio de busca. Fonte: Adaptado de (SAMET, 2006)

Os métodos de acesso métricos também são divididos quanto a forma de persistência da estrutura (memória principal ou secundária). Dentre as estruturas baseadas em memória principal podem ser citadas a *Generalized Hyperplane Tree* (GH-Tree) discutida por SAMET (2006) e a VP-Tree proposta por YIANILOS (1993). As estruturas baseadas em memória secundária não foram estudadas por serem inadequadas para trabalhar com o Apache Spark, que possui um desempenho melhor justamente por trabalhar com memória principal.

Este trabalho explora estruturas de dados otimizadas para buscas em espaços métricos complexos, objetivo central do trabalho, buscando uma que se adeque principalmente aos requisitos de sistemas distribuídos. Dentre estas, a VP-Tree, introduzida por YIANILOS (1993), e a *Multi Vantage-Point Tree* (MVP-Tree), detalhada em estudos posteriores por BOZKAYA; OZSOYOGLU (1999), juntamente com a *Metric Tree* (M-Tree) e GH-Tree, conforme discutidos por SAMET (2006), formam o arcabouço teórico essencial. Estas estruturas, conhecidas como MAM, são fundamentais para melhorar a precisão e eficiência das operações de agrupamento por similaridade, permitindo consultas de proximidade sem a necessidade de examinar cada objeto do conjunto.

Nas próximas seções são apresentados detalhes sobre o funcionamento das estruturas estudadas na intenção de melhorar a precisão e eficiência do particionamento dos dados.

2.6.1 *Vantage-Point Tree*

A VP-Tree é uma estrutura de dados que utiliza um ponto de referência (pivô) para particionar o espaço de dados em regiões hierárquicas baseadas em distância (YIANILOS, 1993), conforme demonstrado na Figura 3. Com suas características de organização espacial baseadas em esferas concêntricas centradas em pontos de vantagem, demonstra uma adaptabilidade notável a diferentes distribuições de dados e uma eficácia superior em espaços de alta dimensionalidade (SAMET, 2006).

A construção da VP-Tree envolve selecionar um ponto de referência e dividir os dados em dois subconjuntos: um contendo os pontos dentro de uma certa distância do pivô e outro contendo os pontos fora dessa distância. Esse processo é repetido recursivamente para criar uma árvore binária que facilita a busca por similaridade (YIANILOS, 1993).

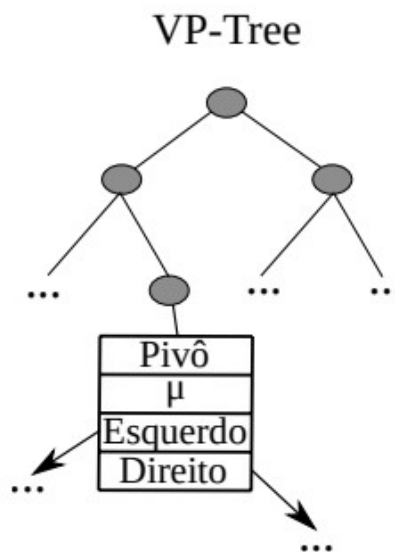


Figura 3 – Estrutura de uma VP-Tree. Adaptado de (YIANILOS, 1993)

A eficiência da VP-Tree decorre da sua capacidade de eliminar grandes porções do espaço de busca que não contêm os resultados da consulta, reduzindo significativamente o número de comparações necessárias (YIANILOS, 1993).

2.6.2 *Multi Vantage-Point Tree*

A MVP-Tree é uma extensão da VP-Tree que utiliza múltiplos pontos de referência em cada nível da árvore para particionar o espaço de dados de maneira mais refinada

(BOZKAYA; OZSOYOGLU, 1999). Esta abordagem melhora a eficiência da busca, especialmente em espaços de alta dimensão, onde a complexidade das consultas pode ser maior. A MVP-Tree permite a realização de consultas de vizinhança e abrangência com maior precisão e menor tempo de resposta. Embora a MVP-Tree ofereça uma abordagem com múltiplos pontos de vantagem que poderia diminuir o número de comparações necessárias, a utilização de múltiplos pontos de vantagem por nó pode dificultar a implementação e aumentar o custo computacional de manutenção e atualização da estrutura, como sugerido por BOZKAYA; OZSOYOGLU (1999). Outro ponto citado por BOZKAYA; OZSOYOGLU (1999) é que ela requer mais espaço em memória devido ao armazenamento de pontos de vantagens adicionais e à estrutura mais complexa necessária para manter múltiplas partições em cada nível.

2.6.3 M-Tree

A M-Tree é uma estrutura de dados hierárquica projetada para indexar e buscar dados em espaços métricos (CIACCIA; PATELLA; ZEZULA, 1997). A M-Tree organiza os dados em nós que representam regiões métricas definidas por um ponto central e um raio de cobertura. Cada nó pode conter vários sub-nós, formando uma estrutura em árvore balanceada que permite a execução eficiente de consultas de vizinhança e abrangência.

A M-Tree é particularmente útil em aplicações onde a precisão das consultas é crítica, como na recuperação de informações multimídia e na análise de dados biométricos (CIACCIA; PATELLA; ZEZULA, 1997).

A M-Tree, embora robusta para ambientes dinâmicos que requerem inserções e exclusões frequentes, apresenta desafios significativos em termos de manutenção do balanceamento, especialmente quando escalada para grandes volumes de dados, o que pode comprometer o desempenho em sistemas distribuídos como *Hadoop* e *Spark* (SAMET, 2006). Sistemas distribuídos dependem da comunicação em rede. Qualquer operação que exija múltiplas atualizações de estado entre nós do cluster, como o reequilíbrio da M-Tree, pode sofrer impacto por latências de rede, afetando seu desempenho geral.

Segundo CIACCIA; PATELLA; ZEZULA (1997), a M-Tree é uma estrutura de dados dinâmica projetada para organizar e buscar grandes conjuntos de dados em disco, particionando objetos de forma que cada nó possa ser armazenado em uma única página de disco, tornando-a particularmente eficiente para armazenamento secundário.

2.6.4 Generalized Hyperplane Tree

A GH-Tree é uma estrutura de dados que utiliza hiperplanos generalizados para particionar o espaço métrico (SAMET, 2006). A GH-Tree divide os dados em dois subconjuntos usando um hiperplano definido por dois pontos de referência. Esta abordagem é eficaz para reduzir a dimensionalidade do espaço de busca e melhorar a eficiência das consultas.

Apesar de ser eficaz na redução da dimensionalidade do espaço de busca, a GH-Tree tende a se tornar ineficiente à medida que a dimensionalidade dos dados aumenta. A complexidade dos cálculos envolvidos na manutenção de hiperplanos em altas dimensionalidades resulta em um *overhead* computacional que pode ser proibitivo (SAMET, 2006).

2.6.5 Comparação entre as Estruturas de dados

Para definição da estrutura de dados mais relevante para o objetivo do trabalho, foram comparadas as estruturas VP-Tree, M-Tree, GH-Tree e MVP-Tree, avaliando-as em termos de eficiência, escalabilidade e facilidade de implementação. A seguir, as vantagens e desvantagens de cada uma, em relação aos requisitos deste trabalho:

1. VP-Tree:

Vantagens: (1) Reduz significativamente o número de comparações necessárias durante as consultas, devido à organização espacial hierárquica (YIANILOS, 1993); (2) Eficiente para dados de alta dimensionalidade e distribuições heterogêneas (SAMET, 2006); (3) Fácil integração com sistemas de computação distribuída, como Spark, por trabalhar com memória principal (YIANILOS, 1993). (4) Estruturada para ser implementada em memória principal para desempenho otimizado (YIANILOS, 1993);

Desvantagens: (1) A construção inicial da árvore pode ser computacionalmente intensiva (YIANILOS, 1993); (2) Pode não ser a mais eficiente para dados com baixa dimensionalidade (SAMET, 2006).

2. M-Tree:

Vantagens: (1) Adequada para dados dinâmicos, suportando inserções e exclusões frequentes (CIACCIA; PATELLA; ZEZULA, 1997); (2) Eficiência garantida para consultas de vizinhança e abrangência (CIACCIA; PATELLA; ZEZULA, 1997).

Desvantagens: (1) Complexidade na manutenção do balanceamento da árvore (CIACCIA; PATELLA; ZEZULA, 1997); (2) Pode apresentar degradação de desempenho em espaços de alta dimensionalidade (SAMET, 2006). (3) Projetada para organizar e buscar grandes conjuntos de dados em disco, sendo eficiente para armazenamento secundário (CIACCIA; PATELLA; ZEZULA, 1997).

3. GH-Tree:

Vantagens: (1) Reduz a dimensionalidade do espaço de busca de forma eficiente (SAMET, 2006); (2) Pode ser aplicada a uma ampla gama de problemas métricos (SAMET, 2006). (3) Projetada como estrutura de dados para armazenamento em memória principal (SAMET, 2006).

Desvantagens: (1) A eficiência diminui significativamente com o aumento da dimensionalidade dos dados (SAMET, 2006); (2) Manutenção dos hiperplanos pode ser computacionalmente custosa (SAMET, 2006).

4. MVP-Tree:

Vantagens: (1) Utiliza múltiplos pontos de referência em cada nível da árvore para particionar o espaço de dados de maneira mais refinada (BOZKAYA; OZSOYOGLU, 1999); (2) Melhora a eficiência da busca, especialmente em espaços de alta dimensão (BOZKAYA; OZSOYOGLU, 1999). (3) Projetada para memória principal para suportar processamento e recuperação de consultas rápidas (BOZKAYA; OZSOYOGLU, 1999);

Desvantagens: (1) A complexidade da construção e manutenção da árvore pode ser alta devido ao uso de múltiplos pontos de referência (BOZKAYA; OZSOYOGLU, 1999); (2) Pode dificultar a implementação e aumentar o custo computacional de manutenção e atualização da estrutura (BOZKAYA; OZSOYOGLU, 1999). (3) Requer mais espaço em memória principal para o armazenamento dos múltiplos pontos de vantagem e à estrutura mais complexa para manter as múltiplas partições em cada nível da árvore (BOZKAYA; OZSOYOGLU, 1999).

A principal diferença entre a VP-Tree e a MVP-Tree reside no número de pontos de referência utilizados para particionar os dados. Enquanto a VP-Tree utiliza um único pivô para criar esferas concêntricas, a MVP-Tree emprega múltiplos pivôs, resultando em uma partição mais refinada (BOZKAYA; OZSOYOGLU, 1999). No entanto, esse refinamento vem ao custo de maior complexidade na construção, manutenção e armazenamento da árvore. Comparada à M-Tree, que é eficiente em dados dinâmicos (CIACCIA; PATELLA; ZEZULA, 1997) e projetada para memória secundária, a VP-Tree pode se destacar em ambientes distribuídos devido à sua simplicidade e eficácia na redução de comparações desnecessárias (YIANILOS, 1993), também por ser projetada para trabalhar em memória principal.

Além do estudo das estruturas para fundamentar a escolha da VP-Tree, foram considerados alguns requisitos da aplicação.

1. **Cenário estático:** A estrutura de dados será criada a partir de um conjunto fixo de pivôs (que representa uma pequena amostra do conjunto de dados), minimizando

a necessidade de atualizações frequentes, como inserções e deleções. Isso favorece a VP-Tree por sua eficiência em consultas com estrutura estática (YIANILOS, 1993).

2. **Armazenamento:** Considerando que os pivôs representam uma pequena amostra do conjunto de dados, optou-se por uma estrutura adequada para memória principal. A VP-Tree, com sua hierarquia eficiente, proporciona armazenamento eficaz e rápido acesso em memória principal (SAMET, 2006). Alternativamente, a MVP-Tree, que utiliza múltiplos pivôs, aumenta a eficiência da busca (BOZKAYA; OZSOYOGLU, 1999), porém demanda mais espaço em memória principal devido à sua estrutura mais complexa.
3. **Integração com Apache Spark:** Apesar da possibilidade de implementação do algoritmo em outros ambientes Map/Reduce, neste trabalho optou-se pelo *Apache Spark*, que tem seu processamento baseado em memória principal. O *Spark* permite o *broadcast* da estrutura criada em memória principal para execução pelos nós do *cluster* (ZAHARIA et al., 2016), e a VP-Tree se integra bem com este ambiente, aproveitando a capacidade do *Spark* de distribuir e processar dados em larga escala (ZAHARIA et al., 2016).

Dessa forma, a escolha da VP-Tree se justifica não apenas por suas características intrínsecas, como eficiência em consultas e compatibilidade com dados de alta dimensionalidade, mas também por sua adequação aos requisitos específicos da aplicação, como a necessidade de uma estrutura estática, armazenamento eficiente em memória principal e integração com o Apache Spark. Esta combinação de fatores torna a VP-Tree a opção mais indicada para o contexto do DSG-VPtree.

2.6.6 Aplicações e Benefícios dos MAMs

Os MAM são amplamente utilizados em diversas aplicações que requerem busca por similaridade eficiente, como recuperação de imagens, bioinformática, sistemas de recomendação e análise de dados geoespaciais. Os benefícios de utilização de MAM incluem (SAMET, 2006):

- ❑ **Eficiência de Consulta:** Redução significativa do número de comparações necessárias para encontrar itens semelhantes.
- ❑ **Escalabilidade:** Capacidade de lidar com grandes volumes de dados e espaços de alta dimensão.
- ❑ **Precisão:** Melhoria na precisão das consultas, resultando em resultados mais relevantes e úteis.

2.7 Técnicas de Particionamento de Dados

O particionamento de dados em espaços métricos é uma técnica essencial para melhorar a eficiência de busca e processamento de grandes volumes de dados. Em espaços métricos, os dados são organizados de acordo com uma métrica de distância, o que permite a realização de consultas por similaridade de forma mais eficaz. Sem técnicas de particionamento adequadas, a busca por similaridade em conjuntos massivos de dados pode se tornar impraticável devido ao alto custo computacional (SAMET, 2006).

A necessidade de particionamento é especialmente crítica em aplicações que envolvem grandes volumes de dados, como mineração de dados, recuperação de informação e análise de *Big Data*. Técnicas de particionamento permitem a decomposição do espaço de dados em regiões menores e mais manejáveis, facilitando o acesso e a consulta aos dados de maneira mais eficiente (SAMET, 2006).

Existem diversas técnicas de particionamento que são utilizadas para organizar dados em espaços métricos, cada uma com suas próprias características e aplicações (SAMET, 2006).

O particionamento com pivôs é uma técnica comum em que um ou mais pontos de referência (pivôs) são selecionados, e os dados são organizados com base em suas distâncias para esses pivôs. Essa abordagem pode reduzir o número de comparações necessárias durante uma consulta, melhorando a eficiência (FU et al., 2000).

SAMET (2006) detalha como os pivôs podem ser utilizados em diversas estruturas de dados métricas para otimizar a busca por similaridade. Estruturas como a VP-Tree e a MVP-Tree utilizam pivôs para criar partições hierárquicas do espaço de dados, permitindo que grandes porções do espaço de busca sejam descartadas rapidamente durante uma consulta. Na VP-Tree, por exemplo, um ponto central (pivô) é escolhido, e os dados são particionados em dois subconjuntos: um contendo os pontos dentro de uma certa distância do pivô e outro contendo os pontos fora dessa distância. Isso facilita a execução de consultas de vizinhança e abrangência de maneira eficiente.

Métodos de particionamento baseados em bolas envolvem a criação de regiões esféricas ao redor de pontos de referência, dividindo o espaço em diferentes subespaços. Esta técnica é particularmente útil para lidar com dados de alta dimensão, pois permite a construção de regiões de busca que podem ser facilmente verificadas quanto à presença de pontos de dados relevantes (YIANILOS, 1993; CIACCIA; PATELLA; ZEZULA, 1997; TRAINA-JR. et al., 2002).

Por outro lado, métodos baseados em hiperplanos utilizam hiperplanos para dividir o espaço em subespaços menores. Uma estrutura comum que utiliza essa abordagem é a GH-Tree, que organiza os dados em uma estrutura de árvore onde cada nó representa um subespaço definido por um hiperplano (SAMET, 2006). A GH-Tree é altamente eficiente para consultas de vizinhança e abrangência devido à sua capacidade de reduzir significativamente o número de comparações necessárias.

SAMET (2006) discute como as técnicas baseadas em bolas e hiperplanos são implementadas em estruturas de dados avançadas para suportar consultas complexas em volumes massivos de dados. A combinação dessas técnicas com a otimização de índices permite um processamento eficiente e escalável, essencial em aplicações de *Big Data*.

Encontrar um conjunto ótimo de pivôs que maximize o somatório de distâncias entre eles é uma tarefa NP-completa, ou seja, é proibitivo encontrar uma solução exata mesmo para um pequeno número de pivôs. Em vez disso, a escolha de elementos aleatórios pode resultar em uma boa distribuição de pivôs (JACOX; SAMET, 2008; HJALTASON; SAMET, 1995).

Quando a quantidade de pivôs é pequena (até poucas dezenas), é possível selecionar pivôs espacialmente distantes entre si com heurísticas lineares. O algoritmo de seleção de pivôs mais simples (*naive pivot selection*) seleciona aleatoriamente n elementos como pivôs (JACOX; SAMET, 2008; HJALTASON; SAMET, 1995). Esta seleção aleatória pode ser executada várias vezes, e o conjunto que maximiza uma função objetivo pode ser escolhido. Funções como o Máximo de Distâncias Mínimas (MMD) e o Máximo de Soma de Distâncias (MSD) (SOCORRO; MICO; ONCINA, 2011; MICO; ONCINA; VIDAL, 1994) selecionam pivôs de forma incremental, começando com um objeto aleatoriamente e escolhendo os pivôs subsequentes com base na maximização das respectivas funções em relação aos já selecionados.

A técnica *Hull of Foci* (HF) (TRAINA-JR et al., 2007) também seleciona pivôs de forma incremental, começando com um objeto aleatoriamente e buscando o objeto mais distante dele (o primeiro pivô). O objeto mais distante do primeiro pivô é definido como o segundo pivô, e a distância entre eles é definida como a borda (*edge*). Os pivôs subsequentes são selecionados como os elementos que resultam no menor erro.

Para o particionamento de dados em um grande número de processos que serão executados em vários nós de um cluster, a seleção incremental de pivôs não gera resultados expressivos em relação à seleção aleatória, tornando-se um custo de processamento desnecessário. Estudos mostram que a utilização de múltiplos pivôs selecionados aleatoriamente é mais eficiente para o particionamento do espaço (JACOX; SAMET, 2008; HJALTASON; SAMET, 1995).

Portanto, no contexto de grandes volumes de dados e a necessidade de particionamento eficiente, a escolha aleatória de pivôs é prática e eficaz. Esta técnica simplifica o processo de seleção de pivôs, podendo ser facilmente implementada e escalada em ambientes de processamento distribuído, como *clusters* de computação utilizando *Apache Spark* (ZAHARIA et al., 2016). A simplicidade e eficiência da seleção aleatória de pivôs, aliada à sua eficácia em cenários de grande escala, justificam sua escolha como método preferencial para particionamento de dados em espaços métricos (JACOX; SAMET, 2008).

Apesar das técnicas de particionamento de dados baseadas em bolas e hiperplanos serem úteis para lidar com dados de alta dimensionalidade e permitirem um processamento

eficiente (SAMET, 2006), o particionamento por pivôs torna-se mais eficiente no *Apache Hadoop* e no *Apache Spark*, pois é feito por igualdade de chaves (ZAHARIA et al., 2016). Nesse caso, para efetuar um particionamento métrico ou espacial por hiperplano separador, com a utilização de pivôs, cada pivô recebe uma chave e os dados dentro da partição recebem a mesma chave, tornando assim o *Spark* capaz de particionar por similaridade, com o custo computacional de encontrar o pivô mais próximo para cada elemento do conjunto.

2.8 Operadores de Agrupamento por Similaridade em *Big Data*

Os operadores de agrupamento por similaridade são fundamentais em diversas áreas que lidam com volumes de dados massivos. Em um ambiente de Big Data, a eficiência e a escalabilidade desses operadores são cruciais para a análise rápida e precisa dos dados. Este capítulo aborda dois operadores que fazem parte do referencial teórico em que esse trabalho baseia-se: SGB e DSG. O SGB agrupa elementos que são semelhantes entre si, enquanto o DSG estende esse conceito para ambientes distribuídos, garantindo escalabilidade e eficiência em volumes de dados massivos (TANG et al., 2016; SILVA; AREF; ALI, 2009; SILVA et al., 2019; OLIVEIRA et al., 2023).

2.8.1 Similarity Group-By (SGB)

O operador SGB foi introduzido por SILVA; AREF; ALI (2009) como uma extensão ao operador tradicional *GROUP BY* encontrado em sistemas de gerenciamento de bancos de dados relacionais. Seu objetivo é permitir o agrupamento de elementos semelhantes com base em uma função de distância definida pelo usuário, superando a limitação do agrupamento por igualdade. O SGB forma grupos de elementos que estão dentro de um limite de similaridade (ϵ) especificado, sendo particularmente útil em cenários onde pequenas variações nos dados são relevantes, como em análise de redes sociais, mineração de dados e bioinformática (SILVA; AREF; ALI, 2009; TANG et al., 2016).

SILVA; AREF; ALI (2009) propuseram estratégias para lidar com sobreposições nos grupos gerados pelo SGB, incluindo a atribuição de elementos ao primeiro grupo compatível (*Join-Any*) ou a criação de novos grupos para elementos que não se encaixam perfeitamente (*Form-New-Group*). TANG et al. (2016) ampliaram essa ideia, aplicando o SGB em contextos multidimensionais e propondo versões como *SGB-All* e *SGB-Any*, que otimizam a definição de grupos em espaços métricos, aumentando a aplicabilidade do operador em volumes de dados massivos (TANG et al., 2016).

Vale também ressaltar que OLIVEIRA et al. (2023) propõem uma versão otimizada do SGB para aplicações industriais, abordando desafios específicos de Big Data, como

a variabilidade dos dados sensoriais e a necessidade de processamento em tempo real. A função de distância adaptativa desenvolvida pelos autores permite uma identificação mais precisa de padrões relevantes, demonstrando melhorias significativas em termos de precisão e tempo de processamento em comparação com métodos tradicionais. Esses avanços são particularmente relevantes em cenários de manutenção preditiva e otimização de processos, onde o agrupamento eficiente de dados pode prevenir falhas e melhorar a eficiência operacional. A abordagem descrita por OLIVEIRA et al. (2023) reforça a importância da eficiência e escalabilidade do SGB, alinhando-se com as discussões teóricas e práticas apresentadas neste trabalho.

2.8.2 Distributed Similarity Grouping (DSG)

O DSG é uma extensão do operador SGB para ambientes de *Big Data*. O DSG foi desenvolvido para lidar com a escalabilidade e a eficiência necessárias ao processar grandes volumes de dados distribuídos em múltiplos nós de computação. Essa abordagem divide o conjunto de dados em partições menores que podem ser processadas paralelamente em diferentes nós, combinando os resultados posteriormente para formar os grupos de similaridade finais (SILVA et al., 2019).

A arquitetura distribuída do DSG permite que ele processe conjuntos de dados massivos de maneira eficiente, utilizando o poder computacional de *clusters* de servidores (SILVA et al., 2019). Isso é particularmente importante em aplicações de *Big Data*, onde a quantidade de dados pode facilmente exceder a capacidade de processamento de um único nó. A eficiência do DSG em ambientes distribuídos o torna uma ferramenta valiosa para análise de grandes volumes de dados rapidamente e com precisão.

Para melhor entendimento sobre o algoritmo proposto nesse trabalho, é necessário um detalhamento da abordagem algorítmica do DSG, de acordo com SILVA et al. (2019), para estabelecer uma base sobre a qual as melhorias são introduzidas.

Conforme demonstrado na Figura 4, o algoritmo DSG inicia com a seleção de pivôs a partir dos dados de entrada, que são utilizados para particionar o conjunto de dados. A próxima fase é o particionamento dos dados entre os pivôs escolhidos. Cada elemento de entrada é atribuído à partição associada ao pivô mais próximo e aqueles localizados na região de fronteira entre as partições são replicados, conforme demonstrado na Figura 5, para garantir a correta detecção dos grupos de similaridade nessas regiões. O particionamento é realizado utilizando uma distância euclidiana generalizada para calcular a distância de um elemento ao hiperplano que separa duas partições, assegurando uma distribuição eficiente dos dados.

Na fase de formação dos grupos de similaridade, cada partição é processada individualmente. Se a partição for pequena o suficiente, ou seja, menor ou igual ao *threshold* definido pelo usuário na chamada do programa, um algoritmo de nó único é usado para identificar os grupos de similaridade; caso contrário, a partição é armazenada para re-

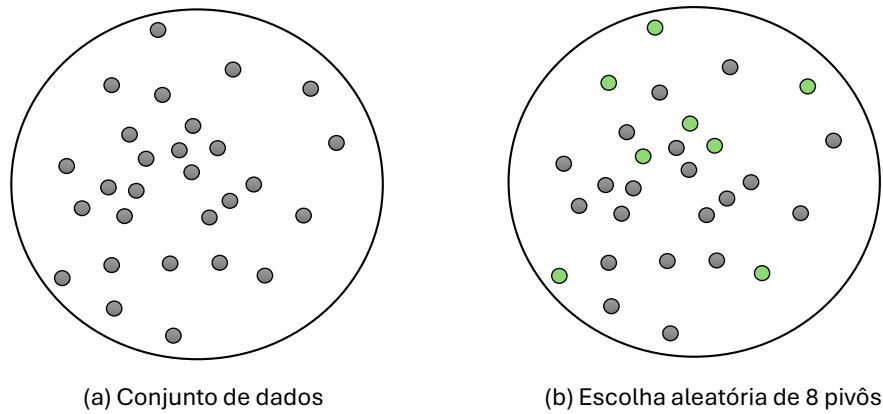


Figura 4 – Ilustração da escolha de pivôs aleatória, passo inicial do algoritmo para iniciar o particionamento de dados. Fonte: Figura do autor.

processamento em rodadas subsequentes utilizando o mesmo algoritmo DSG, até que a partição seja pequena o suficiente. Durante a verificação de pertinência aos grupos, cada elemento é comparado com todos os elementos do grupo para verificar se satisfaz a condição de similaridade, e se assim for, é adicionado ao grupo. Finalmente, cada grupo é marcado com informações de *flag* (onde cada elemento é anotado com informações sobre a sua partição base e a partição à qual foi atribuído) para garantir que os grupos sejam gerados sem duplicação em diferentes partições, permitindo uma organização eficiente e precisa dos dados distribuídos.

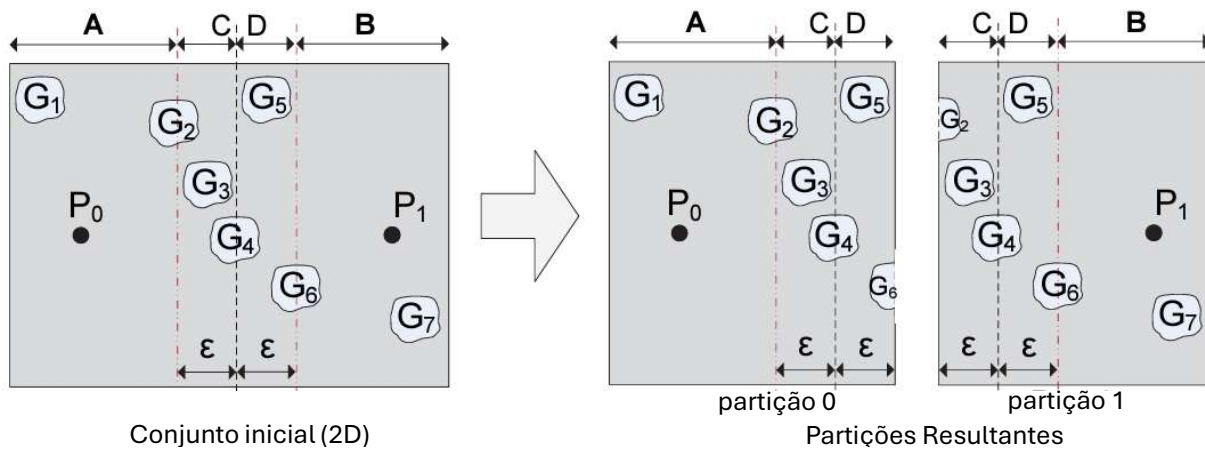


Figura 5 – Particionamento e agrupamento por similaridade pelo DSG usando dois pivôs. Fonte: (SILVA et al., 2019)

Para exemplificar o funcionamento do algoritmo DSG, conforme demonstrado na Figura 5, observa-se o particionamento de um conjunto de dados inicial em duas partições de modo a identificar todos os grupos de similaridade (G_1 a G_7). No exemplo, cada grupo de similaridade deve ser gerado apenas em uma partição. O algoritmo fará o particiona-

mento dos dados de entrada utilizando dois pivôs (P0 e P1), de modo que cada ponto pertença à partição do seu pivô mais próximo. Adicionalmente, o algoritmo duplicará os pontos nas janelas ε (C e D), de modo que a partição 0 será dividida em A+C+D, e a partição 1 será dividida em C+D+B. Para identificar os grupos de similaridade em cada partição, o algoritmo utilizará a lógica demonstrada na Tabela 1 para a partição P0 e na Tabela 2 para a partição P1. Dessa forma, após a formação de grupos e a validação de janelamento, os grupos de similaridade G1, G2, G3 e G4 serão gerados na partição P0, enquanto os grupos de similaridade G5, G6 e G7 serão gerados na partição P1. Nesse caso, o grupo G4 permaneceu na partição P0 devido a conter mais pontos próximos a esta partição.

Tabela 1 – Identificação dos Grupos de Similaridade na Partição P0. Fonte: (SILVA et al., 2019)

Se o Grupo	Ação
Somente em A	Gerar grupo em P0
Em A e C	Gerar grupo em P0
Somente em C	Gerar grupo em P0
Em C e D	Gerar grupo em P0
Somente em D	Ignorar

Tabela 2 – Identificação dos Grupos de Similaridade na Partição P1. Fonte: (SILVA et al., 2019)

Se o Grupo	Ação
Somente em C	Ignorar
Em C e D	Ignorar
Somente em D	Gerar grupo em P1
Em D e B	Gerar grupo em P1
Somente em B	Gerar grupo em P1

A utilização do *Apache Spark* e sua *API* de RDD para a implementação do DSG demonstra um aumento significativo na concisão e eficiência do código, conforme ilustrado por SILVA et al. (2019). O processo de seleção de pivôs, particionamento de dados, e a consolidação dos elementos em grupos de similaridade são facilitados por funções específicas da plataforma *Spark*, como *mapPartitionsToPair* e *groupByKey* (ZAHARIA et al., 2016).

Importante também citar que IQBAL; LISSANDRINI; PEDERSEN (2022) propõem técnicas avançadas para melhorar a eficiência do agrupamento por similaridade, introduzindo algoritmos híbridos e abordagens multi-nível que podem ser integradas ao DSG. Esses algoritmos combinam técnicas de particionamento e métodos de agrupamento baseados em densidade, reduzindo a complexidade computacional e acelerando o processamento. A avaliação empírica apresentada pelos autores demonstra que essas técnicas

superam métodos tradicionais em precisão e tempo de processamento, validando a eficácia das estratégias aplicadas em grandes conjuntos de dados.

2.8.3 Benefícios e limitações do DSG

Dentre os benefícios do DSG destacados por (SILVA et al., 2019) estão:

- ❑ **Escalabilidade:** A capacidade de distribuir o processamento de dados em múltiplos nós permite que o DSG lide com conjuntos de dados de grande escala de maneira eficiente.
- ❑ **Eficiência:** O processamento paralelo e a capacidade de particionar dados reduzem o tempo necessário para agrupar elementos semelhantes, mesmo em grandes volumes de dados.
- ❑ **Flexibilidade:** O DSG pode ser adaptado para diferentes métricas de similaridade e ajustado para atender a várias necessidades específicas de aplicação.

No entanto, o DSG também apresenta algumas limitações (SILVA et al., 2019):

- ❑ **Complexidade de Implementação:** A configuração e o gerenciamento de um ambiente distribuído podem ser complexos, exigindo conhecimento especializado em sistemas distribuídos e processamento paralelo.
- ❑ **Comunicação entre Nós:** A necessidade de comunicação entre os nós durante o processamento pode introduzir latência, especialmente em redes de grande escala.
- ❑ **Equilíbrio de Carga:** Garantir que a carga de trabalho seja distribuída uniformemente entre os nós é um desafio, podendo afetar a eficiência geral do sistema.
- ❑ **Escalabilidade:** Embora o DSG melhore a escalabilidade em relação a operadores tradicionais, ele ainda enfrenta desafios quando lida com conjuntos de dados massivos e heterogêneos.

Estudos de caso em bioinformática demonstraram como o DSG pode ser utilizado para agrupar sequências genéticas semelhantes, facilitando a identificação de padrões genéticos e a descoberta de novas relações biológicas (SILVA et al., 2023). Na análise de redes sociais, o DSG foi aplicado para detectar comunidades de usuários com interesses similares, melhorando a segmentação de mercado e a personalização de conteúdos.

2.9 Trabalhos Correlatos

O operador DSG, introduzido como uma evolução do SGB, é uma abordagem fundamental para lidar com agrupamentos baseados em similaridade em ambientes distribuídos (SILVA et al., 2019). Estudos recentes demonstram que o tema permanece em destaque, com contribuições significativas tanto para o aprimoramento do SGB quanto para áreas correlatas de agrupamento e análise de conjunto de dados massivos.

O trabalho de OLIVEIRA et al. (2023) introduz um novo enfoque sobre o operador SGB ao explorar a diversificação de resultados nos agrupamentos por similaridade, aplicável em cenários como sistemas de recomendação e mineração de dados. Este estudo destaca que a diversificação é uma característica fundamental para maximizar a representatividade dos grupos formados, especialmente em aplicações onde a redundância pode comprometer os resultados, e que este avanço oferece ganhos significativos em escalabilidade e precisão para análise de Big Data.

SILVA et al. (2023), por sua vez, ampliaram os conceitos do operador SGB ao propor o *Diversity Similarity Join* (DSJ) para grandes volumes de dados. Esse trabalho se concentra na eficiência computacional de junções baseadas em similaridade, integrando técnicas de otimização para reduzir o tempo de execução em plataformas distribuídas, como *Apache Spark*. Os autores ressaltam que a abordagem é particularmente útil para lidar com dados de alta dimensionalidade e demonstram, através de experimentos, uma melhoria expressiva na execução de consultas.

Já os avanços em algoritmos de agrupamento continuam sendo investigados, como ilustrado por IKOTUN et al. (2023) em sua revisão abrangente sobre variações do algoritmo *K-Means* no contexto de *Big Data*. Embora este estudo não se concentre diretamente no SGB ou no DSG, ele fornece *insights* sobre técnicas de agrupamento eficientes e escaláveis que podem ser comparadas ou integradas às abordagens baseadas em similaridade. As contribuições incluem melhorias específicas para ambientes de *Big Data*, destacando o impacto da alta dimensionalidade e dos volumes crescentes de dados.

Por fim, ALASALI; ORTAKCI (2024) conduz uma análise detalhada de técnicas de agrupamento em mineração de dados, discutindo métodos e desafios atuais. Este estudo aborda diretamente a aplicabilidade do agrupamento em domínios como bioinformática e análise de redes sociais, corroborando a importância de operadores como o SGB e DSG para auxiliar no processamento dos dados e revelar padrões em conjuntos de dados massivos. A análise ressalta a necessidade de soluções cada vez mais otimizadas, como as oferecidas pelas evoluções do SGB e do DSG.

Esses estudos reforçam que o tema tratado neste trabalho está alinhado às tendências e necessidades atuais, demonstrando sua relevância tanto em contextos acadêmicos quanto práticos. Além disso, as soluções propostas no presente trabalho contribuem diretamente para o avanço da área ao incorporar inovações como o DSG-VPTree, respondendo aos desafios atuais de eficiência e escalabilidade em ambientes de *Big Data*.

2.10 Considerações

Neste capítulo, foram discutidos diversos conceitos e metodologias fundamentais para a análise e processamento de grandes volumes de dados, abordando desde o conceito de *Big Data* até as técnicas avançadas de particionamento de dados e métodos de acesso métrico. Inicialmente, foram introduzidos os conceitos básicos de *Big Data*, destacando a importância da análise de grandes volumes de dados e os desafios associados à extração de conhecimento de dados massivos e complexos (DEAN; GHEMAWAT, 2004; ZAHARIA et al., 2016).

Subsequente à introdução, a análise de agrupamentos em *Big Data* foi explorada, com destaque para métodos como K-Means, CURE e DBSCAN, que são essenciais para a segmentação e descoberta de padrões em grandes conjuntos de dados (JAIN, 2010). A busca por similaridade, uma técnica crucial para a recuperação eficiente de informações, também foi discutida, incluindo consultas por abrangência, k vizinhos mais próximos e a importância das funções de distância (ZEZULA et al., 2006).

Algumas técnicas de particionamento de dados em espaços métricos foram detalhadas, mostrando como o particionamento com pivôs e métodos baseados em bolas e hiperplanos podem otimizar a busca por similaridade (SAMET, 2006). Além disso, a eficácia dos sistemas distribuídos para o processamento de dados massivos foi abordada, destacando paradigmas como *MapReduce* e plataformas como *Apache Hadoop* e *Spark* (DEAN; GHEMAWAT, 2004; ZAHARIA et al., 2016).

Os métodos de acesso métrico, incluindo VP-Tree, MVP-Tree, M-Tree e GH-Tree, foram analisados em termos de sua capacidade de otimizar buscas métricas em grandes volumes de dados (FU et al., 2000; SAMET, 2006). O capítulo também discutiu os operadores de agrupamento por similaridade em *Big Data*, como SGB e DSG, ressaltando seus benefícios e limitações (SILVA; AREF; ALI, 2009; TANG et al., 2016; SILVA et al., 2019).

A relevância desta pesquisa é justificada pela crescente necessidade de ferramentas e métodos eficazes para a análise de grandes volumes de dados em diversos domínios, incluindo negócios, ciência, saúde e tecnologia. Com a explosão da quantidade de dados gerados diariamente, técnicas avançadas de processamento e análise são essenciais para extrair valor e *insights* significativos desses dados. A combinação de métodos de acesso métrico com sistemas distribuídos oferece uma solução robusta para os desafios de escalabilidade e eficiência em *Big Data*.

Este capítulo estabelece a base teórica necessária para os capítulos subsequentes, que detalharão a implementação e experimentação das técnicas discutidas. A integração da VP-Tree ao método DSG será explorada em profundidade, com uma análise prática de seu desempenho e eficácia em ambientes de *Big Data*. Além disso, estudos de caso e experimentos empíricos serão apresentados para validar a abordagem proposta, proporcionando uma compreensão completa de sua aplicabilidade e benefícios em contextos reais.

Algoritmo de agrupamento por similaridade em operações distribuídas com Vantage Point Tree (DSG-VPTree)

O presente capítulo apresenta a proposta do algoritmo DSG-VPTree, desenvolvido com o intuito de melhorar a eficiência e a escalabilidade de operações de agrupamento em volumes de dados massivos. A proposta baseia-se na integração da estrutura de dados VP-Tree ao método de agrupamento distribuído DSG, resultando em uma solução que melhora significativamente o desempenho das consultas de similaridade em ambientes distribuídos.

A motivação da escolha da VP-Tree como componente central da solução é detalhada, destacando as características que a tornam especialmente adequada para ambientes de dados massivos. São explorados os benefícios esperados da integração da VP-Tree ao método DSG, incluindo a redução do número de comparações necessárias, a escalabilidade aprimorada, a eficiência nas consultas, a robustez em relação a dados de alta dimensionalidade e a eficiência no uso de memória principal.

Além disso, a metodologia do algoritmo DSG-VPTree também é apresentada, incluindo a construção da VP-Tree e o processo de busca para o particionamento, a partir dessa árvore. Também são descritos os impactos dessa integração na eficiência e no desempenho das operações de consulta por similaridade, demonstrando como a estrutura hierárquica da VP-Tree contribui para acelerar as operações e reduzir a sobrecarga computacional.

Ao longo deste capítulo, são introduzidos e formalizados os conceitos e métodos desenvolvidos, com o objetivo de fornecer uma compreensão abrangente do algoritmo DSG-VPTree e suas contribuições para o campo de agrupamento de dados em sistemas distribuídos.

3.1 Integração da Vantage Point Tree ao Método DSG

A VP-Tree é uma estrutura de dados eficiente para busca por similaridade em espaços métricos. Sua capacidade de particionar o espaço de dados de maneira que grandes porções do espaço de busca possam ser descartadas rapidamente, torna-a ideal para comprovar a hipótese deste trabalho. Isso reduz significativamente o número de comparações necessárias durante uma consulta, melhorando a eficiência do processo de agrupamento (YIANILOS, 1993).

Além disso, a VP-Tree é particularmente eficaz em ambientes distribuídos, onde a eficiência e a escalabilidade são essenciais. A sua estrutura hierárquica baseada em pivôs permite que os dados sejam organizados de forma a minimizar a necessidade de comunicação entre os nós de processamento, o que é crucial para manter a performance em sistemas de *Big Data* (FU et al., 2000).

A escolha da VP-Tree se justifica não apenas pelas suas características intrínsecas, como eficiência em consultas e compatibilidade com dados de alta dimensionalidade, mas principalmente pela sua adequação aos requisitos específicos da aplicação, como a necessidade de uma estrutura estática, armazenamento eficiente em memória principal e integração com o Apache Spark a partir do *broadcast* para memória principal. Esta combinação de fatores torna a VP-Tree a opção mais indicada para o contexto do DSG-VPTree.

3.1.1 Benefícios esperados da VP-Tree em comparação com outras estruturas

A integração da VP-Tree ao método DSG traz diversos benefícios em comparação com outras estruturas de dados:

- ❑ **Redução de Comparações:** A capacidade da VP-Tree de particionar o espaço métrico de maneira eficiente (YIANILOS, 1993) reduz o número de comparações necessárias para encontrar itens semelhantes, o que é fundamental para lidar com volumes de dados massivos.
- ❑ **Eficiência de Consulta:** A estrutura hierárquica da VP-Tree permite a execução rápida de consultas de vizinhança e abrangência (SAMET, 2006), melhorando a eficiência das operações de busca por similaridade.
- ❑ **Robustez a Dimensionalidade:** A VP-Tree é eficaz em lidar com dados de alta dimensão (FU et al., 2000), uma característica importante para muitas aplicações modernas que envolvem conjuntos de dados complexos e multifacetados.
- ❑ **Eficiência em Memória Principal:** A VP-Tree é projetada para ser eficiente em termos de armazenamento e acesso em memória principal (SAMET, 2006), o que

a torna ideal para aplicações que requerem processamento rápido e eficiente sem depender de armazenamento em disco.

3.1.2 Metodologia do DSG-VPTree

O operador DSG-VPTree combina os princípios do DSG com a estrutura da VP-Tree para otimizar o agrupamento de dados semelhantes em ambientes distribuídos. A seguir estão os principais componentes e processos do DSG-VPTree:

- ❑ **Construção da VP-Tree:** Inicialmente, uma árvore única e estática é criada a partir do conjunto de pivôs, que são organizados em uma estrutura hierárquica baseada em distâncias aos pivôs (YIANILOS, 1993), e distribuída entre os nós do cluster. Cada nó possui uma cópia local da mesma VP-Tree.
- ❑ **Consulta Distribuída:** Durante a execução de uma consulta, cada nó utiliza sua cópia local da VP-Tree para identificar rapidamente os dados relevantes, descartando grandes porções do espaço de busca que não contêm resultados pertinentes (SAMET, 2006).
- ❑ **Agregação de Resultados:** Os resultados das consultas individuais de cada nó são agregados para formar o resultado final da busca por similaridade. Esta etapa envolve a comunicação mínima entre os nós, mantendo a eficiência do sistema distribuído (SILVA et al., 2019).

3.1.3 Impactos na eficiência e desempenho das operações de consulta por similaridade

A integração da VP-Tree ao método DSG tem um impacto significativo na eficiência e desempenho das operações de consulta por similaridade:

- ❑ **Desempenho Acelerado:** A estrutura eficiente da VP-Tree permite que as operações de consulta sejam realizadas de maneira mais rápida (YIANILOS, 1993), reduzindo o tempo total de processamento.
- ❑ **Escalabilidade Aprimorada:** A capacidade da VP-Tree de funcionar bem em ambientes distribuídos, trabalhando em memória principal, permite que o operador DSG-VPTree escale horizontalmente, lidando eficazmente com o aumento do volume de dados (DEAN; GHEMAWAT, 2004; SAMET, 2006).
- ❑ **Redução de Sobrecarga Computacional:** A hierarquia baseada em pivôs da VP-Tree minimiza a necessidade de comparações desnecessárias, reduzindo a sobrecarga computacional e melhorando a utilização dos recursos (SAMET, 2006).

3.2 Detalhes da Solução Proposta

O algoritmo DSG, apesar de eficiente em diversos cenários, apresenta limitações em termos de velocidade de execução quando aplicado a grandes volumes de dados. A função *getClosestPivot*, responsável por determinar o pivô mais próximo para cada elemento, representa um gargalo significativo, especialmente porque é executada múltiplas vezes para cada elemento, conforme demonstrado na Figura 6. Primeiro, para identificar o pivô mais próximo, conforme ilustrado por (a) e (b), e novamente para confirmar esta proximidade dentro de um limite definido (ϵ), identificando pontos que fazem parte da janela e portanto também estão próximos de outros pivôs, como mostrado em (c). O Algoritmo 1 apresenta a parte do algoritmo principal que é foco do presente trabalho, destacando os pontos de gargalo nas linhas 3 e 5. Na linha 3 é chamada a função *getClosestPivot* para identificar o pivô mais próximo ao elemento. Já da linha 5 à linha 9, é feita a identificação de elementos dentro da janela definida por ϵ .

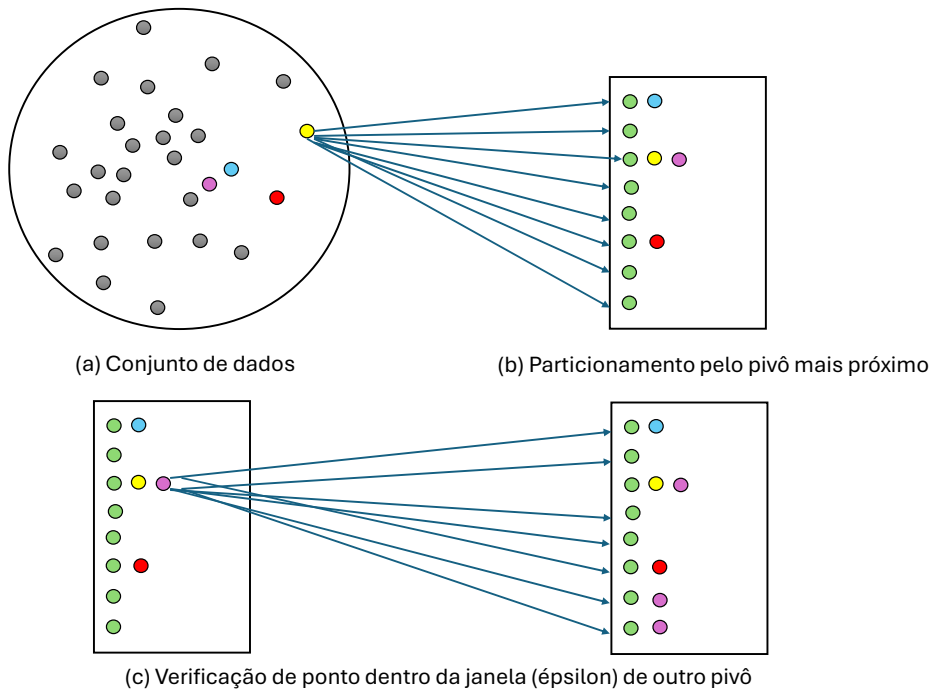


Figura 6 – Particionamento dos dados pelo pivô mais próximo e por janelamento (épsilon). (a) Conjunto de dados inicial onde alguns elementos foram destacados representando o elemento de busca para encontrar o pivô mais próximo. (b) Conjunto com os pivôs escolhidos de forma aleatória e o particionamento feito a partir da distância do elemento de busca para o elemento de pivô mais próximo. (c) Após a primeira busca pelo pivô mais próximo, é feita a verificação de distância para os demais elementos de pivô, buscando distância dentro de épsilon que representa elemento dentro da janela de outro pivô. Fonte: Figura do autor.

Algoritmo 1 DistSimGrouping**Require:** *inputData*, *eps*, *numPivots*, *memT***Ensure:** similarity groups in *inputData*

```

1: pivots  $\leftarrow$  selectPivots(numPivots, inputData) ▷ Partitioning
2: for each record r in a chunk of inputData do
3:   Pc  $\leftarrow$  getClosestPivot(r, pivots)
4:   output  $\langle P_c, r \rangle$  ▷ intermediate output
5:   for each pivot p in  $\{pivots - P_c\}$  do
6:     if  $(dist(r, p) - dist(r, P_c))/2 \leq eps$  then
7:       output  $\langle p, r \rangle$  ▷ intermediate output
8:     end if
9:   end for
10: end for ▷ Shuffle: records with same key  $\Rightarrow$  partition
11: for each partition Pi do ▷ Group Formation
12:   .....
13:   .....
14: end for

```

Para superar essas limitações, propomos a integração da VP-Tree na estrutura do DSG. A VP-Tree é uma estrutura de dados que permite consultas por similaridade eficientes e pode ser particularmente útil para acelerar a identificação de pivôs próximos. Após a escolha aleatória do conjunto de pivôs, uma VP-Tree estática é construída a partir desses elementos e distribuída aos nós do *cluster* via mecanismo de *broadcast* do *Apache Spark*. Esse processo garante que cada nó receba uma cópia única e imutável dessa árvore, que fica armazenada em memória principal, permitindo consultas locais sem a necessidade de comunicação constante entre os nós, o que reduz drasticamente a latência nas operações de busca, conforme descrito por ZAHARIA et al. (2016).

Apesar de apresentar um custo associado à sua construção, a VP-Tree é extremamente útil durante as consultas, permitindo a poda rápida de um número significativo de elementos sem a necessidade de recalcular todas as distâncias. Esse benefício deriva da metodologia de estruturação dos elementos na VP-Tree: inicialmente, escolhe-se um elemento como ponto de vantagem. Em seguida, calcula-se a distância de todos os demais elementos até este ponto de vantagem (o).

O próximo passo consiste em calcular a mediana dessas distâncias. Utilizando essa mediana μ , os elementos são bipartidos. Conforme demonstrado na Figura 7, aqueles elementos cuja distância é igual ou menor que μ são posicionados no nó esquerdo, enquanto aqueles elementos com distância maior são alocados no nó direito. Essa divisão estratégica permite consultas eficientes, pois durante uma busca, parte dos elementos pode ser excluída com base na comparação de distâncias relativas a μ .

A construção da VP-Tree é completada por meio de um processo recursivo, onde este método de divisão é aplicado sucessivamente aos nós filhos, até que todos os elementos estejam adequadamente atribuídos.

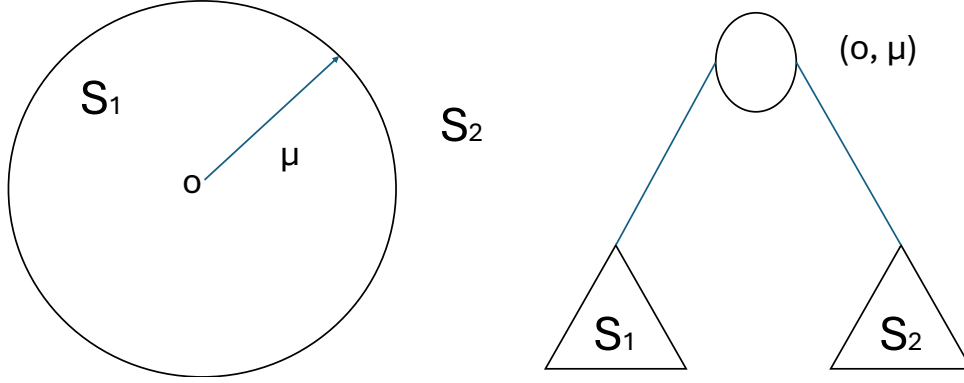


Figura 7 – Distribuição de dados em uma VP-Tree. Adaptado de (FU et al., 2000)

Algoritmo 2 DistSimGroupingVpTree

Require: *inputData*, *eps*, *numPivots*, *memT*

Ensure: similarity groups in *inputData*

- 1: $pivots \leftarrow selectPivots(numPivots, inputData)$
 - 2: $vptree \leftarrow createVpTree(pivots, euclidianDist)$
 - 3: **for** each record r in a chunk of *inputData* **do** ▷ Partitioning
 - 4: $P_c \leftarrow getClosestPivotVpTree(r, pivots, vptree, eps)$
 - 5: **end for** ▷ Shuffle: records with same key $\Rightarrow$ partition
 - 6: **for** each partition P_i **do** ▷ Group Formation
 - 7:
 - 8:
 - 9: **end for**
-

A incorporação da VP-Tree altera tanto o algoritmo principal da DSG quanto a função *getClosestPivot*. Para resolver o problema proposto, era necessário uma estrutura estática, sem exclusões ou atualizações, eficiente em memória principal e capaz de ser distribuída para todos os nós do *cluster* Spark. A VP-Tree foi escolhida por cumprir esses requisitos, possibilitando a criação de uma única árvore distribuída via *broadcast* para todos os nós do *cluster*. A construção da árvore é baseada no conjunto de pivôs selecionados aleatoriamente, que corresponde a uma amostra aleatória do conjunto, cujo tamanho é determinado pelo usuário no momento da chamada do algoritmo.

O Algoritmo 2 demonstra o algoritmo principal ajustado, onde foi incluído na linha 2 a criação da VP-Tree a partir da lista de pivôs e de uma função euclidiana. Também é possível observar, na linha 4, a chamada da função que trará o pivô mais próximo e também fará o tratamento dos elementos contidos na janela definida por ϵ . Essa função utiliza o elemento sendo tratado, a árvore de vantage point criada em memória principal e o valor de ϵ . O tratamento de elementos contidos em janela passa a ser efetuado dentro da função *getClosestPivotVpTree*, no mesmo laço que o pivô mais próximo, já garantindo um ganho de desempenho inicial.

No Algoritmo 3 é possível verificar que inicialmente é feita a busca pelo vizinho mais próximo, na linha 1. Essa função, retorna o pivô mais próximo ao ponto de pesquisa e a

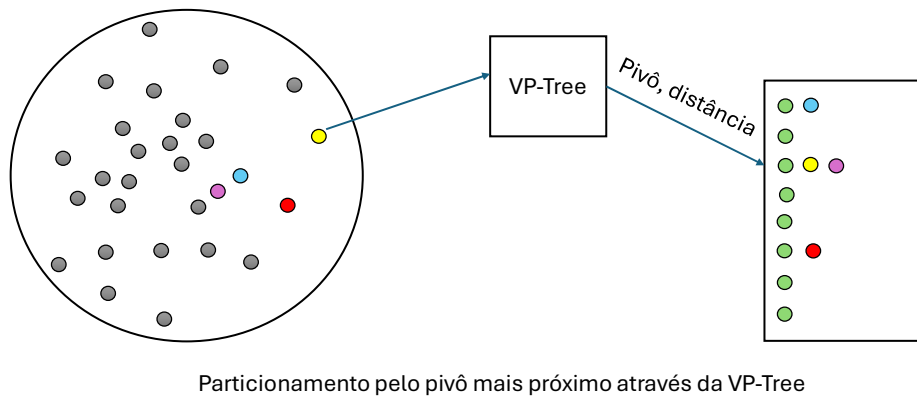


Figura 8 – Particionamento do conjunto de dados através da utilização da VP-Tree para encontrar o vizinho (pivô) mais próximo. Fonte: Figura do autor.

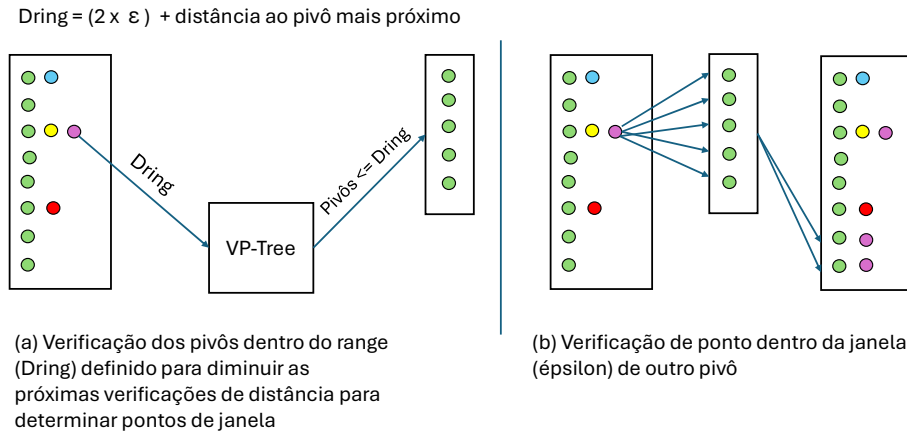


Figura 9 – Particionamento do conjunto de dados através da utilização da VP-Tree para identificar pontos que pertencem às janelas de outros pivôs. (a) Verificação dos pivôs dentro do range definido por $Dring$ para reduzir as próximas verificações. (b) Verificação de pontos dentro da janela de outro pivô a partir de um número menor de pivôs. Fonte: Figura do autor.

distância entre os dois, conforme demonstrado na Figura 8. Na Figura 9, é demonstrado um dos pontos essenciais do trabalho proposto, que inclui o cálculo da variável $Dring$, para efetuar a poda na próxima busca para determinar pontos que são próximos a mais de um pivô, que pode ser vista na linha 3. Nessa busca, a VP-Tree também é utilizada, recebendo além do ponto de pesquisa, um valor de *range* definido por $Dring$, conforme pode ser visto em (a). Para melhorar o desempenho dessa verificação de pontos presentes em mais de uma partição, a variável $Dring$ representa o valor de 2 vezes ϵ mais o valor da distância entre o ponto e seu vizinho mais próximo, identificado no passo anterior. Como pode ser observado na Figura 10, através dessa variável, é possível definir um limite de poda de pontos para a próxima busca por janelamento. Dessa forma, na linha 4, onde

é feito a busca pelos vizinhos pertencentes à janela, também demonstrando na Figura 9 em (b), a quantidade de pivôs é determinada pela busca por abrangência limitada pelo raio determinado por $Dring$, ou seja, apenas os pivôs dentro da distância definida por $Dring$ terão sua distância verificada para entrar no agrupamento. Essa poda remove da busca, aqueles pivôs muito distantes do ponto de busca, gerando um ganho muito grande no desempenho do algoritmo.

Algoritmo 3 GetClosestPivotVpTree

Require: $r, vptree, eps$

Ensure: closest pivots

- 1: $P_c, D_c \leftarrow getVPtreeNearestNeighbor(r, vptree)$
 - 2: output $\langle P_c, r \rangle$ ▷ intermediate output
 - 3: $Dring \leftarrow (2 \times \epsilon) + D_c$
 - 4: $pivotsRing \leftarrow getAllInRange(vptree, Dring)$
 - 5: **for** each pivot p in $\{pivotsRing - P_c\}$ **do**
 - 6: **if** $(dist(r, p) - dist(r, P_c))/2 \leq \epsilon$ **then**
 - 7: output $\langle p, r \rangle$ ▷ intermediate output
 - 8: **end if**
 - 9: **end for**
-

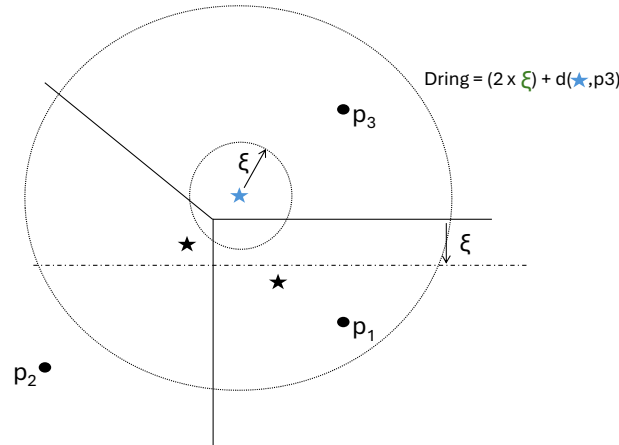


Figura 10 – Cálculo de distância para poda de busca por abrangência. Fonte: Figura do autor.

Espera-se que a integração da VP-Tree reduza a complexidade de tempo da função *getClosestPivot* de $\mathcal{O}(n)$ para $\mathcal{O}(\log n)$ em média, considerando a eficiência típica das árvores de *vantage point* em cenários de busca dimensional. Esta melhoria pode resultar em uma redução significativa no tempo total de processamento, especialmente em conjuntos de dados massivos onde o número de elementos e pivôs é grande. Adicionalmente, a redução do tempo de busca contribui para uma menor utilização de recursos computacionais, como CPU e memória, otimizando o uso da infraestrutura de computação distribuída.

3.3 Construção da VP-Tree

A construção da VP-Tree começa após a seleção do conjunto de pivôs, que são escolhidos aleatoriamente a partir do conjunto de dados inicial. O usuário do algoritmo define o número de pivôs com os quais deseja particionar o conjunto inicial. Esses pivôs servirão como pontos de referência para a criação da VP-Tree, facilitando a busca pelos vizinhos mais próximos. Após a seleção dos pivôs, o processo de construção da VP-Tree é iniciado, conforme descrito a seguir.

A construção da VP-Tree envolve a seleção de um ponto de referência (pivô) e a divisão dos dados em subpartições com base nas distâncias calculadas a partir desse pivô. Esta abordagem hierárquica permite a redução significativa do espaço de busca, melhorando a eficiência das operações de consulta.

Os passos para Construção da VP-Tree são apresentados abaixo e demonstrados pelo Algoritmo 4:

1. Seleção do Pivô: Selecionar um ponto de referência (pivô) a partir do conjunto de dados.
2. Cálculo das Distâncias: Calcular as distâncias de todos os outros pontos em relação ao pivô.
3. Determinação da Mediana: Determinar a mediana das distâncias calculadas.
4. Divisão dos Dados: Dividir os dados em duas subpartições: pontos dentro da mediana e pontos fora da mediana.
5. Recursão: Aplicar recursivamente os passos 1 a 4 para as subpartições até que cada partição contenha um único ponto ou esteja vazia.

Algoritmo 4 Construção da VP-Tree

Require: *points, distFn*

Ensure: vantage point tree

```

1: function BUILDVP TREE(points, distFn)
2:   vp = SELECTVANTAGEPOINT(points)
3:   pivotIndex = vp.index
4:   pivot = vp.coordinates
5:   distances = CALCULATEDISTANCES(points, pivot, distFn)
6:   median = FINDMEDIAN(distances)
7:   {leftPoints, rightPoints} = PARTITIONPOINTS(points, distances, median)
8:   leftTree = BUILDVP TREE(leftPoints, distFn)
9:   rightTree = BUILDVP TREE(rightPoints, distFn)
10:  return new VpTreeNode(pivotIndex, pivot, median, leftTree, rightTree)
11: end function

```

3.4 Busca na VP-Tree pelo vizinho mais próximo

A busca na VP-Tree pelo vizinho mais próximo utiliza a propriedade hierárquica da árvore para localizar rapidamente os pontos mais próximos do ponto de consulta. O algoritmo de busca compara a distância do ponto de consulta ao pivô em cada nó e decide recursivamente qual subárvore explorar com base na distância calculada e na mediana do nó.

Algoritmo 5 Busca na VP-Tree

```

1: function NEARESTNEIGHBORSEARCH(query, tree, distFn)
2:   return NNRECURSIVE(query, tree, distFn, Double.MAX_VALUE)
3: end function
4: function NNRECURSIVE(query, node, distFn, bestDistance)
5:   distanceToPivot = CALCULATEDISTANCE(query, node.pivot, distFn)
6:   bestResult = UPDATEBESTRESULT(node.pivotIndex, node.pivot, distanceToPi-
    vot, bestDistance)
7:   if distanceToPivot < bestDistance then
8:     bestDistance = distanceToPivot
9:   end if
10:  if distanceToPivot < node.median then
11:    closerNode = node.left
12:    fartherNode = node.right
13:  else
14:    closerNode = node.right
15:    fartherNode = node.left
16:  end if
17:  closerResult = NNRECURSIVE(query, closerNode, distFn, bestDistance)
18:  if closerResult is not null and closerResult.distance < bestDistance then
19:    bestResult = closerResult
20:    bestDistance = closerResult.distance
21:  end if
22:  if distanceToPivot - bestDistance <= node.median then
23:    fartherResult = NNRECURSIVE(query, fartherNode, distFn, bestDistance)
24:    if fartherResult is not null and fartherResult.distance < bestDistance then
25:      bestResult = fartherResult
26:    end if
27:  end if
28:  return bestResult
29: end function

```

Os passos para busca na VP-Tree são apresentados abaixo e demonstrados pelo Algoritmo 5:

1. Calcular a Distância ao Pivô: Calcular a distância d entre o ponto de consulta e o pivô do nó atual.

2. Atualizar o Melhor Resultado: Atualizar o melhor resultado se a distância d for menor que a melhor distância conhecida.
3. Decisão de Recursão: Decidir qual subárvore explorar (esquerda ou direita) com base na comparação de d com a mediana.
4. Busca na Subárvore Mais Próxima: Buscar recursivamente na subárvore mais próxima.
5. Verificação da Subárvore Mais Distante: Se a busca na subárvore mais próxima não for suficiente, verificar a subárvore mais distante.

3.5 Busca por abrangência na VP-Tree

A busca por abrangência (*range query*) na VP-Tree visa encontrar todos os elementos que estão dentro de uma determinada distância (raio) a partir de um elemento de consulta. Esse tipo de consulta é especialmente útil para identificar todos os elementos em um espaço métrico que satisfazem um critério de proximidade específico.

Os passos para busca por abrangência na VP-Tree são apresentados no Algoritmo 6:

1. Calcular a Distância ao Pivô: Calcular a distância d entre o ponto de consulta e o pivô do nó atual.
2. Verificar Inclusão no Range: Incluir o pivô no resultado se a distância d for menor ou igual ao raio especificado.
3. Decisão de Recursão: Determinar qual subárvore explorar primeiro (esquerda ou direita) com base na comparação de d com a mediana do nó.
4. Busca na Subárvore Mais Próxima: Recursivamente buscar na subárvore mais próxima.
5. Verificação da Subárvore Mais Distante: Se a distância do pivô menos o raio especificado for menor ou igual à mediana, verificar a subárvore mais distante, pois ela pode conter pontos dentro do range.

3.6 Considerações Finais

Neste capítulo, foi detalhado o processo de integração da estrutura de dados VP-Tree ao operador DSG, resultando na proposta do DSG-VP-Tree. A abordagem explorou as vantagens da VP-Tree, como sua eficiência em particionamento de dados em espaços

Algoritmo 6 Busca por abrangência definida na VP-Tree

```

1: function RANGESEARCH(query, tree, maxDistance, distFn)
2:   neighbors = NEW(List)
3:   RANGERECURSIVE(query, tree, maxDistance, neighbors, distFn)
4:   return neighbors
5: end function
6: function RANGERECURSIVE(query, node, maxDistance, neighbors, distFn)
7:   distanceToPivot = CALCULATEDISTANCE(query, node.pivot, distFn)
8:   if distanceToPivot  $\leq$  maxDistance then
9:     neighbors.add((distanceToPivot, node.pivot))
10:  end if
11:  if distanceToPivot < node.median then
12:    RANGERECURSIVE(query, node.left, maxDistance, neighbors, distFn)
13:    if distanceToPivot + maxDistance  $\geq$  node.median then
14:      RANGERECURSIVE(query, node.right, maxDistance, neighbors, distFn)
15:    end if
16:  else
17:    RANGERECURSIVE(query, node.right, maxDistance, neighbors, distFn)
18:    if distanceToPivot - maxDistance  $\leq$  node.median then
19:      RANGERECURSIVE(query, node.left, maxDistance, neighbors, distFn)
20:    end if
21:  end if
22: end function

```

métricos de alta dimensionalidade e sua capacidade de reduzir a quantidade de cálculos em consultas por similaridade.

Além disso, a metodologia apresentada indica que a utilização da VP-Tree pode otimizar operações críticas, como a busca por vizinhos mais próximos e o particionamento eficiente de volumes de dados massivos, contribuindo para a escalabilidade e melhor desempenho do operador DSG. Também foram fornecidos detalhes sobre a construção e a utilização da estrutura no ambiente distribuído *Apache Spark*.

Esses avanços teóricos e metodológicos fornecem a base para a realização dos experimentos apresentados no próximo capítulo, nos quais será avaliado o impacto prático do DSG-VP-Tree em cenários com diferentes dimensões e volumes de dados. Assim, este capítulo consolida a proposta metodológica do trabalho, destacando sua contribuição para a melhoria nas técnicas de agrupamento por similaridade em sistemas distribuídos.

Experimentos e Análise dos Resultados

Neste capítulo, são apresentados os experimentos realizados e a análise dos resultados obtidos com o objetivo de validar a eficácia da incorporação da VP-Tree no algoritmo DSG. Inicialmente, descrevemos o método utilizado para a avaliação, incluindo as configurações do ambiente de teste e os parâmetros experimentais. Em seguida, detalhamos os resultados dos experimentos conduzidos, comparando o desempenho do DSG tradicional com a versão aprimorada DSG-VPTree em diferentes cenários. Por fim, apresentamos uma avaliação crítica dos resultados, destacando os benefícios e as limitações da abordagem proposta.

4.1 Método para a Avaliação

Os experimentos foram conduzidos utilizando *Hadoop* versão 3.3.6 e *Spark* versão 3.5.1, e Linux kernel versão 5.4 virtualizado com KVM. O *cluster* consiste de um nó mestre e cinco nós de trabalho. Todos os nós estão equipados com dois processadores *Intel Xeon Quad-Core* com frequência de 2.4 GHz (8 cores). O nó mestre tem 48 GB de memória RAM e 8 TB de espaço em disco, enquanto os nós de trabalho têm 24 GB de memória RAM e 1 TB de espaço em disco.

Embora esse ambiente tenha sido adequado para testar conjuntos de dados de tamanho moderado, ele apresentou limitações significativas quando se tratou de processar conjuntos de dados maiores. Para esses testes mais exigentes, era necessário um ambiente mais robusto com recursos computacionais mais avançados, como nós com mais núcleos de processamento, maior quantidade de memória RAM e maior capacidade de armazenamento.

A falta de disponibilidade de um ambiente com essas especificações mais elevadas limitou a extensão dos testes que puderam ser realizados. Essa limitação impactou diretamente a capacidade de executar testes em escala maior e avaliar completamente o desempenho do DSG-VPTree sob condições de alta carga e grande volume de dados. No

entanto, os testes realizados foram suficientes para comprovar grande parte dos objetivos almejados.

Apesar das restrições do ambiente, foi possível observar melhorias significativas no desempenho do DSG-VPtree em comparação com o DSG, especialmente em termos de tempo de execução e uso de recursos computacionais para os conjuntos de dados testados. Essas melhorias são indicativas do potencial do DSG-VPtree para ambientes de produção, onde a disponibilidade de recursos não é uma limitação tão severa.

Para a avaliação, utilizou-se um gerador de dados sintéticos configurado para simular diversas condições de teste, o mesmo utilizado para os testes do DSG no artigo publicado anteriormente. A intenção era reproduzir exatamente os mesmos experimentos documentados no estudo anterior, garantindo assim a confiabilidade e a comparabilidade dos resultados. Os dados gerados representavam grupos de similaridade em espaços multidimensionais (de 64D a 256D), com a separação entre grupos definida por $2 * \epsilon$. Essa abordagem permitiu verificar a consistência dos resultados entre o DSG original e o DSG-VPtree sob condições controladas e bem documentadas, assegurando que qualquer melhoria de desempenho observada fosse devida às otimizações introduzidas, e não a variações no ambiente de teste.

Para a realização dos experimentos, os parâmetros gerais do algoritmo foram:

- ❑ Função de Distância: Euclidiana
- ❑ Fator de escala: SFN onde cada conjunto de dados continha aproximadamente 200.000 registros multiplicados por N.
- ❑ Número de pivôs: 40 x N (SFN)
- ❑ Limiar de partição (threshold): 50.000
- ❑ Limiar de janela (Epsilon): 100
- ❑ Dimensões: 64 a 256

Os testes foram executados a partir do *spark-submit*, no modo *deploy client*. Cada registro foi replicado de uma a três vezes para testar a robustez dos algoritmos frente a duplicações.

4.2 Experimentos

Os experimentos focaram em avaliar a eficiência do processamento e a acurácia na formação de grupos de similaridade. Comparou-se o desempenho do DSG e do DSG-VPtree em termos de tempo de execução e utilização de recursos, além de verificar a integridade dos grupos formados em presença de variações de dimensionalidade e tamanho do conjunto de dados.

4.2.1 Validação da formação de Grupos de Similaridade

Os resultados dos experimentos são apresentados nas tabelas a seguir. As tabelas mostram o número de grupos de similaridade identificados e o tempo total de execução para os algoritmos DSG e DSG-VPTree, incluindo a construção da árvore VP-Tree, em diferentes cenários.

Tabela 3 – Resultados do Experimento: número de grupos identificados com 64 dimensões e com o aumento do fator de escala. Fonte: Tabela do autor.

SFN	Número de Pivôs	DSG		DSG-VPTree	
		Grupos Identificados	Tempo de Execução (s)	Grupos Identificados	Tempo de Execução (s)
1	40	2696	24	2696	22
2	80	5392	28	5392	28
3	120	8078	39	8078	35
4	160	10762	44	10762	37
5	200	13455	60	13455	49
6	240	16133	68	16133	50
7	280	18817	83	18817	60
8	320	21484	93	21484	61
9	360	24163	112	24163	73
10	400	26862	122	26862	74

Na Tabela 3, observa-se os resultados do experimento com o aumento do fator de escala (SFN). Os dados mostram que o tempo de execução do DSG-VPTree é consistentemente menor do que o do DSG para a grande maioria dos valores de SFN. Além disso, o número de grupos de similaridade identificados é idêntico para ambos os algoritmos, demonstrando que a integridade dos resultados foi preservada. Isso indica que a otimização introduzida pela VP-Tree melhora o tempo de processamento sem comprometer a qualidade dos resultados.

Tabela 4 – Resultados do Experimento: número de grupos identificados com SF10, com o aumento do número de dimensões. Fonte: Tabela do autor.

Dimensões	Número de Pivôs	DSG		DSG-VPTree	
		Grupos Identificados	Tempo de Execução (s)	Grupos Identificados	Tempo de Execução (s)
64	400	26862	122	26862	75
96	400	26829	170	26829	100
128	400	26835	223	26835	125
160	400	26841	272	26841	151
192	400	26876	317	26876	173
224	400	26854	364	26854	197
256	400	26871	419	26871	231

Na Tabela 4 estão os resultados com o aumento do número de dimensões. Novamente, o DSG-VPTree demonstra um tempo de execução significativamente menor em comparação ao DSG, enquanto o número de grupos de similaridade identificados permanece igual para ambos os algoritmos. Esse resultado reforça que a incorporação da VP-Tree otimiza o desempenho do algoritmo sem alterar a precisão dos resultados, mesmo em espaços de alta dimensionalidade.

Tabela 5 – Resultados do Experimento: número de grupos identificados com SF10 e 64 dimensões, com o aumento do número de pivôs. Fonte: Tabela do autor.

Número de Pivôs	DSG		DSG-VPTree	
	Grupos Identificados	Tempo de Execução (s)	Grupos Identificados	Tempo de Execução (s)
400	26862	126	26862	79
500	26862	137	26862	81
600	26862	149	26862	80
700	26862	161	26862	78
800	26862	172	26862	79
900	26862	187	26862	80
1000	26862	198	26862	80
1100	26862	209	26862	82
1200	26862	222	26862	83
1300	26862	236	26862	84
1400	26862	246	26862	83
1500	26862	259	26862	86
1600	26862	271	26862	85
1700	26862	284	26862	83
1800	26862	297	26862	87
1900	26862	309	26862	85
2000	26862	317	26862	86

Na Tabela 5 são demonstrados os resultados com o aumento do número de pivôs para 64 dimensões. Observa-se que o DSG-VPTree continua a apresentar um tempo de execução menor do que o DSG, independentemente do número de pivôs. O número de grupos de similaridade identificados permanece o mesmo, evidenciando que a precisão do algoritmo não foi afetada pelas otimizações introduzidas.

Tabela 6 – Resultados do Experimento: número de grupos identificados com SF10 e 192 dimensões, com o aumento do número de pivôs. Fonte: Tabela do autor.

Número de Pivôs	DSG		DSG-VPTree	
	Grupos Identificados	Tempo de Execução (s)	Grupos Identificados	Tempo de Execução (s)
100	26876	259	26876	205
200	26876	242	26876	176
300	26876	282	26876	174
400	26876	315	26876	175
500	26876	354	26876	176
600	26876	394	26876	177
700	26876	431	26876	179
800	26876	470	26876	179
900	26876	512	26876	180
1000	26876	548	26876	178
1100	26876	587	26876	181
1200	26876	626	26876	184
1300	26876	671	26876	179
1400	26876	709	26876	182
1500	26876	753	26876	182
1600	26876	800	26876	183
1700	26876	847	26876	184
1800	26876	900	26876	189
1900	26876	953	26876	188
2000	26876	1004	26876	188

Por fim, na Tabela 6 exibe-se os resultados do experimento com o aumento do número

de pivôs para 192 dimensões. Mais uma vez, o DSG-VPTree supera o DSG em termos de tempo de execução, enquanto mantém a consistência no número de grupos de similaridade identificados. Isso reafirma a eficiência da VP-Tree na otimização do algoritmo DSG, sem comprometer a integridade dos resultados.

Os resultados apresentados nas tabelas acima indicam claramente que a incorporação da VP-Tree ao algoritmo DSG resulta em uma melhoria significativa no tempo de execução, sem impactar a acurácia na formação de grupos de similaridade. O fato de o número de grupos de similaridade identificados ser idêntico em ambos os algoritmos demonstra que a otimização foi bem-sucedida em manter a integridade dos resultados. Essa otimização é atribuída à eficiência da VP-Tree em reduzir o número de comparações necessárias durante as buscas por similaridade, especialmente em cenários de alta dimensionalidade e grande volume de dados.

Além disso, a preservação do número de grupos de similaridade identificados indica que o algoritmo principal foi mantido inalterado, com as melhorias focadas nos pontos de otimização para reduzir o tempo de processamento. Com isso, podemos efetuar as comparações entre os dois algoritmos com a certeza de estar reproduzindo praticamente o mesmo código, exceto pelas melhorias implementadas. Essas melhorias fazem do DSG-VPTree uma abordagem promissora para aplicações em ambientes de produção, onde a eficiência e a escalabilidade são cruciais.

4.2.2 Validação do Tempo de Processamento

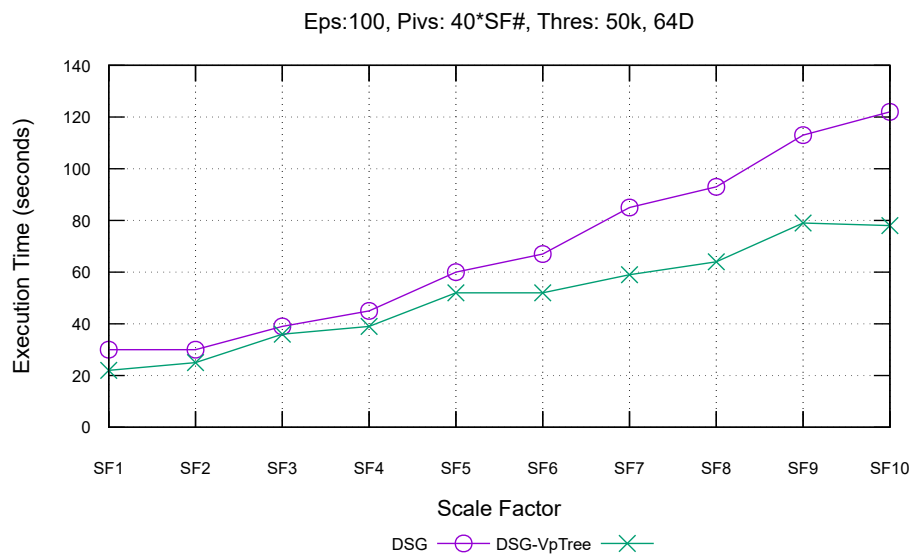


Figura 11 – Tempo de execução com o aumento de tamanho do conjunto de dados. Fonte: Figura do autor.

Aumento do fator de escala. Observa-se, na Figura 11, o comportamento dos dois algoritmos à medida que o fator de escala aumenta. Nesse experimento, o valor de

epsilon foi fixado em 100 e o número de dimensões em 64D. Até 1.000.000 de registros (SF5), o tempo de execução para ambos os algoritmos permanece comparativamente próximo, mostrando um aumento gradual. Entretanto, a partir de 1.200.000 registros (SF6), observa-se que o tempo de execução para o DSG começa a crescer mais rapidamente, alcançando um tempo 40% maior que o do DSG-VPtree quando expandido para SF10. Avaliando as funções do gráfico e suas derivadas, é possível notar que conforme o número de registros (SF) aumenta, o tempo de execução da DSG aumenta linearmente em função de uma constante maior que do DSG-VPtree.

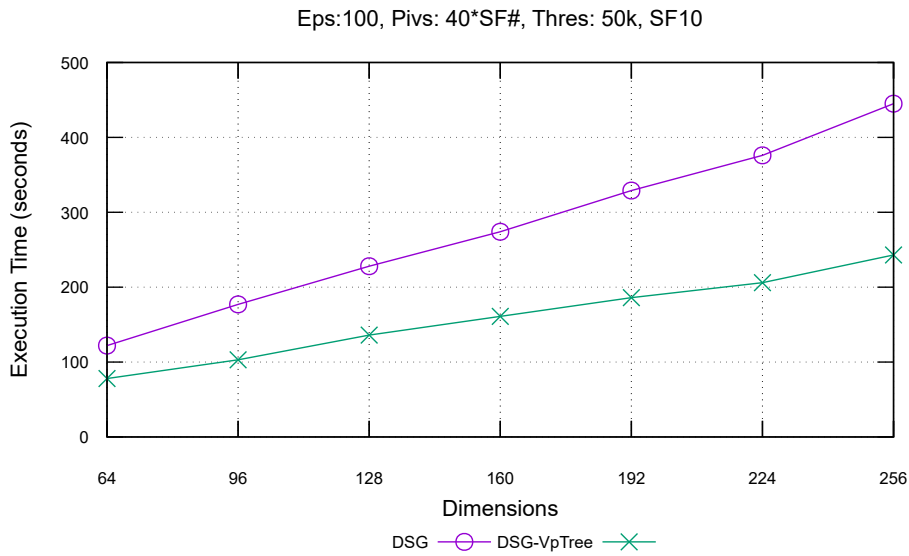
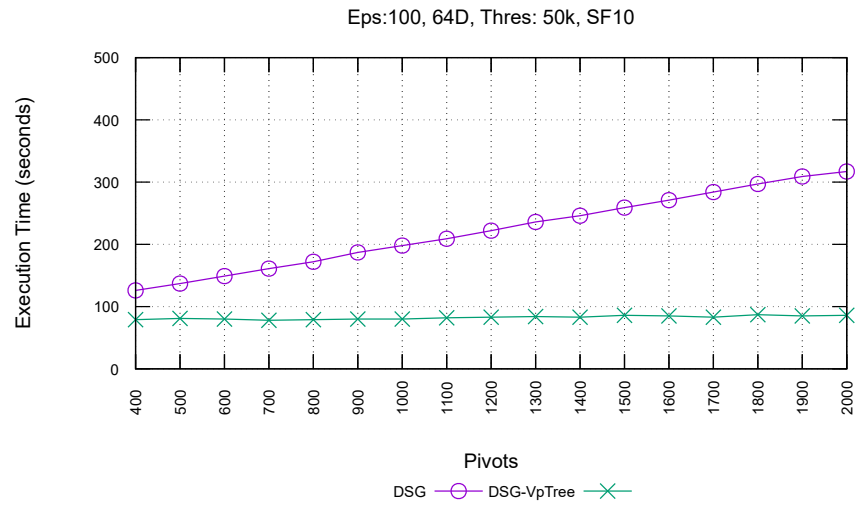


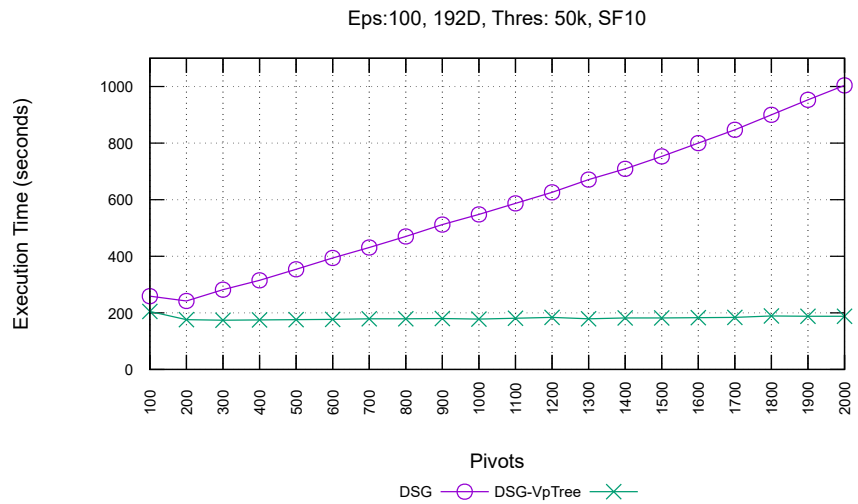
Figura 12 – Tempo de execução com o aumento de número de dimensões. Fonte: Figura do autor.

Aumento do número de dimensões. Os experimentos foram realizados fixando o fator de escala em SF10, *epsilon* em 100, enquanto o número de dimensões foi variado de 64D a 256D. Conforme demonstrado na Figura 12, o tempo de execução para ambos os algoritmos aumenta com o número de dimensões. No entanto, o aumento no tempo de execução do DSG-VPtree é mais moderado em comparação com o DSG tradicional. Notavelmente, com 256 dimensões, o tempo de execução do DSG é aproximadamente o dobro do tempo necessário para o DSG-VPtree, sugerindo que o DSG-VPtree está mais preparado para lidar com contextos de alta dimensionalidade. Com a análise das funções do gráfico e suas derivadas, é possível notar que o tempo de execução da DSG possui um crescimento mais rápido comparado ao tempo da DSG-VPtree. Mostra ainda que a DSG-VPtree cresce de maneira mais controlada e linear.

Aumento do número de pivôs. Realizaram-se experimentos variando o número de pivôs em duas configurações distintas: 64 dimensões e 192 dimensões. Em ambos os casos, os valores de *epsilon* foram mantidos constantes em 100, o limiar de partição em 50000 e o fator de escala em SF10. Conforme ilustrado na Figura 13, observa-se que, para as duas configurações dimensionais testadas, o DSG-VPtree apresenta um desempenho mais



(a) 64 Dimensões.



(b) 192 Dimensões.

Figura 13 – Tempo de execução com aumento de número de pivôs. Fonte: Figura do autor.

estável e uma melhor escalabilidade. Nas configurações de 192 dimensões, à medida que o número de pivôs aumenta, o DSG-VPTree tem tempo de execução constante, enquanto o DSG tem um aumento de tempo que cresce linearmente com o número de pivôs. Ao analisar as funções dos gráficos e suas derivadas, verifica-se que o DSG tem um ponto de inflexão a partir do qual o aumento no número de partições resulta em aumento no tempo de processamento, enquanto o DSG-VPTree exibe um comportamento constante. Esse experimento evidencia os benefícios proporcionados pela utilização de uma árvore *vantage-point*. Mesmo com o aumento da quantidade de pivôs, estes são eficientemente indexados na árvore, o que melhora significativamente a eficiência das buscas. Isso ocorre principalmente devido à poda de elementos não candidatos durante a busca por pontos na

janela, o que reduz o número de comparações necessárias e acelera o processo de busca.

4.3 Avaliação dos Resultados

A avaliação dos resultados dos experimentos confirma a hipótese de que a incorporação da VP-Tree no algoritmo DSG oferece vantagens significativas em termos de eficiência e escalabilidade. Os testes demonstraram que o DSG-VPTree não apenas mantém a integridade dos grupos de similaridade formados, considerando sua quantidade, mas também opera de maneira mais eficiente em termos de tempo de execução, especialmente em cenários de alta dimensionalidade e volume de dados massivos. Contudo, é importante notar que, embora o DSG-VPTree tenha mostrado melhorias consideráveis, há limitações a serem consideradas, como a dependência da escolha do número de pivôs e o impacto do tamanho da amostra na construção da árvore. Futuras pesquisas podem explorar técnicas avançadas de seleção de pivôs e estratégias de balanceamento de carga para otimizar ainda mais o desempenho do algoritmo em diferentes contextos.

4.4 Considerações Finais

O capítulo apresentou a avaliação dos experimentos do operador DSG-VPTree, destacando sua eficiência em operações de agrupamento por similaridade em ambientes distribuídos. Os experimentos permitiram validar aspectos importantes do operador, como o desempenho em termos de tempo de processamento, formação de grupos de similaridade e escalabilidade em cenários de alta dimensionalidade.

Os resultados demonstraram que o DSG-VPTree superou o operador DSG em até 40% no tempo de execução, mantendo a integridade dos grupos de similaridade formados, considerando sua quantidade. Essa melhoria deve-se à utilização eficiente da estrutura VP-Tree, que reduz a sobrecarga computacional ao minimizar a necessidade de comparações redundantes durante o particionamento dos dados e a execução das buscas por similaridade.

Esses resultados corroboram a hipótese inicial do trabalho, reafirmando o potencial do DSG-VPTree como uma solução escalável para particionamento de dados com pivôs no *Apache Spark* para busca por similaridade em Big Data. A partir desse resultado, o próximo capítulo sintetizará as contribuições gerais do trabalho, destacando sua relevância acadêmica e prática, além de apontar perspectivas para pesquisas futuras.

Conclusão

Este trabalho introduziu o DSG-VPTree, uma evolução significativa do algoritmo DSG que incorpora a estrutura de dados VP-Tree para melhorar o desempenho do particionamento de dados e aprimorar as operações de busca por similaridade em conjuntos de dados de grande escala. A implementação proposta provou ser notavelmente superior em termos de velocidade quando comparada ao DSG, demonstrando um desempenho até 40% melhor em contextos de alta dimensionalidade e grandes volumes de dados, conforme evidenciado pelos testes de desempenho extensivos.

Avaliando todos os experimentos e os resultados obtidos, é possível notar que os seguintes objetivos foram atingidos:

1. O desenvolvimento do DSG-VPTree conseguiu melhorar o particionamento dos dados, utilizando a VP-Tree para criar partições de forma mais otimizada e eficiente.
2. A pesquisa demonstrou que a adição da VP-Tree ao operador DSG, o tornou mais eficiente na organização de dados em espaços métricos, proporcionando uma redução significativa no tempo de busca devido à sua capacidade de eliminar rapidamente grandes porções do espaço de busca.
3. Os resultados experimentais mostraram que o DSG-VPTree supera o DSG em termos de tempo de execução, especialmente em conjuntos de dados de alta dimensionalidade. A integridade dos grupos de similaridade também foi mantida, ao verificar sua quantidade, confirmando a eficácia do novo método.
4. Foi fornecida uma abordagem detalhada e clara para a implementação do DSG-VPTree no *Apache Spark*, mostrando cada etapa do processo. A avaliação demonstrou que a implementação no *Spark* resultou em uma latência menor e maior eficiência nas operações de busca.
5. A avaliação do impacto da VP-Tree mostrou melhorias significativas no desempenho das operações de busca por similaridade, independentemente das condições de dimensionalidade e tamanho do conjunto de dados.

Também é possível verificar que as perguntas norteadoras da pesquisa foram respondidas: (1) A VP-Tree foi integrada ao DSG criando uma estrutura hierárquica que permite o particionamento eficiente do espaço de dados, reduzindo o número de comparações necessárias; (2) Os experimentos confirmaram que a VP-Tree melhora o desempenho do DSG em termos de tempo de execução por diminuir a quantidade de cálculos necessários; (3) O detalhamento da implementação incluem a construção e distribuição da VP-Tree entre os nós do cluster, otimizando a distribuição de carga e minimizando a latência das operações de busca.

A presente pesquisa atingiu os objetivos propostos, demonstrando melhorias importantes no particionamento de dados e nas operações de busca por similaridade em conjuntos de dados massivos. A integração da VP-Tree ao DSG proporcionou um método mais eficiente, contribuindo para a evolução das técnicas de particionamento de dados com pivôs para análise em *Big Data*. As perguntas norteadoras da pesquisa também foram respondidas de maneira satisfatória, evidenciando a eficácia da abordagem proposta.

Em conclusão, o DSG-VPTree representa um avanço significativo na análise de *Big Data*, proporcionando uma ferramenta robusta para o agrupamento de dados por similaridade. Este trabalho não apenas demonstra a viabilidade de usar a VP-Tree para melhorar o desempenho das operações de agrupamento, mas também abre novas perspectivas para a pesquisa e aplicação prática em ambientes de computação distribuída.

5.1 Principais Contribuições

A hipótese que guiou este trabalho postulava que a inclusão de estruturas de indexação métricas no particionamento dos dados, por meio de um algoritmo de custo linear, poderia melhorar significativamente o desempenho das operações de busca por similaridade, reduzindo o tempo de computação necessário para o processamento de operações de agrupamento por similaridade. Os resultados experimentais confirmaram essa hipótese, demonstrando que a integração da VP-Tree ao método DSG resultou em ganhos substanciais de eficiência e escalabilidade.

As principais contribuições trazidas pelo estudo estão listadas abaixo:

1. **Novo Algoritmo para Particionamento de Dados Complexos:** O novo algoritmo demonstrou ser eficiente no particionamento de dados com pivôs e pode ser utilizado em outros operadores que necessitam melhorar seu desempenho.
2. **Otimização do Particionamento de Dados:** O uso da VP-Tree permitiu a otimização do particionamento de dados com pivôs, mitigando as limitações de escalabilidade e melhorando o desempenho das operações de busca por similaridade, com a redução da quantidade de cálculos de distância entre elementos.

3. **Implementação no Apache Spark:** Foram fornecidas instruções para a implementação do DSG-VPTree no ambiente *Apache Spark*. Os detalhes fornecidos incluíram o processo de construção e distribuição da VP-Tree entre os nós do *cluster*.
4. **Disponibilização de Código:** O código fonte e o conjunto de dados foi disponibilizado no GitHub ¹ para que a pesquisa tenha maior transparência e reprodutibilidade. O compartilhamento do código permite que outros pesquisadores e profissionais possam verificar os resultados, replicar os experimentos e utilizar as metodologias desenvolvidas em seus próprios trabalhos. Além disso, facilita a colaboração e o avanço contínuo da pesquisa na área, permitindo que melhorias e adaptações possam ser implementadas por uma comunidade mais ampla.

5.2 Trabalhos Futuros

Os experimentos realizados mostraram que o DSG-VPTree apresenta uma performance superior em diversas métricas quando comparado ao DSG. No entanto, algumas áreas ainda apresentam oportunidades para melhorias:

1. **Otimização do Processo de Seleção de Pivôs:** A seleção de pivôs é um componente crítico para a eficácia da VP-Tree. Em trabalhos futuros, podem ser investigadas técnicas mais avançadas de seleção de pivôs que minimizem o custo computacional e maximizem a eficiência da estrutura de dados.
2. **Redução da Latência de Comunicação:** A comunicação entre nós durante as operações de busca pode introduzir latências significativas. Futuras pesquisas podem focar em técnicas de redução de latência, como compressão de dados, protocolos de comunicação mais eficientes ou até mesmo métodos que minimizem a necessidade de comunicação.
3. **Escalabilidade em Ambientes Heterogêneos:** Testar e adaptar o DSG-VPTree para funcionar eficientemente em ambientes de computação heterogêneos, onde os nós do *cluster* possuem capacidades e recursos variados, pode ser um caminho importante para garantir a aplicabilidade universal do método.
4. **Exploração de Técnicas de Aprendizado de Máquina:** Integrar técnicas de aprendizado de máquina para melhorar a seleção de pivôs e a organização da VP-Tree, utilizando algoritmos que aprendam com os dados históricos e ajustem automaticamente os parâmetros do sistema.
5. **Avaliação em Diferentes Plataformas de Processamento:** Ampliar a avaliação do DSG-VPTree em diversas plataformas de processamento distribuído, como

¹ <https://github.com/asilva-dados/DSG-VPTree>

Apache Flink e *Google Dataflow*, para identificar vantagens e desvantagens em cada ambiente e adaptar o método às especificidades de cada plataforma.

6. **Exploração do DuckDB ² para Otimização:** Avaliar a utilização do DuckDB para otimizar a seleção de pivôs, reduzir a latência de comunicação através de pré-processamento local de dados e testar a escalabilidade em ambientes heterogêneos. Além disso, avaliar o desempenho do DSG-VPTree com DuckDB em conjunto com outras plataformas de processamento distribuído pode fornecer *insights* valiosos sobre a adaptabilidade e eficiência do método proposto.
7. **Desenvolvimento de Ferramentas de Visualização:** Criar ferramentas de visualização para analisar os resultados dos agrupamentos de similaridade, permitindo uma compreensão mais intuitiva dos dados e facilitando a tomada de decisões baseada em insights visuais.

5.3 Contribuições em Produção Bibliográfica

O operador desenvolvido nessa dissertação foi publicado e apresentando no 39º Simpósio Brasileiro de Banco de Dados (SBBD 2024) (SILVA; RAZENTE, 2024) como artigo completo, intitulado "Desempenho de operações distribuídas de agrupamento por similaridade em dados de alta dimensionalidade por meio da VP-Tree". O artigo foi apresentado em Florianópolis, na sessão *Best Papers* e recebeu Menção Honrosa.

² O DuckDB é um sistema de gerenciamento de banco de dados orientado a colunas, projetado para executar consultas analíticas de forma eficiente em sistemas distribuídos. Ele oferece suporte completo à linguagem SQL e é otimizado para processamento em memória, permitindo análises rápidas e escaláveis. Mais informações estão disponíveis em: <<https://duckdb.org>>.

Referências

- ABADI, D. et al. The seattle report on database research. **Commun. ACM**, v. 65, n. 8, p. 72–79, 2022. Disponível em: <<https://doi.org/10.1145/3524284>>.
- ALASALI, T.; ORTAKCI, Y. Clustering techniques in data mining: A survey of methods, challenges, and applications. **Computer Science**, v. 9, p. 32–50, 06 2024. Disponível em: <<https://doi.org/10.53070/bbd.1421527>>.
- BOZKAYA, T.; OZSOYOGLU, M. Indexing large metric spaces for similarity search queries. **ACM Transactions on Database Systems (TODS)**, ACM New York, NY, USA, v. 24, n. 3, p. 361–404, 1999. Disponível em: <<https://doi.org/10.1145/328939.328959>>.
- CHEN, L. et al. Indexing metric spaces for exact similarity search. **ACM Comput. Surv.**, v. 55, n. 6, p. 128:1–128:39, 2023. Disponível em: <<https://doi.org/10.1145/3534963>>.
- CHEN, M.; MAO, S.; LIU, Y. Big data: A survey. **Mob. Networks Appl.**, v. 19, n. 2, p. 171–209, 2014. Disponível em: <<https://doi.org/10.1007/s11036-013-0489-0>>.
- CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree: An efficient access method for similarity search in metric spaces. In: **International Conference on Very Large Data Bases (VLDB)**. [s.n.], 1997. p. 426–435. Disponível em: <<https://www.vldb.org/conf/1997/P426.PDF>>.
- DADVAR, V.; GOLAB, L.; SRIVASTAVA, D. Exploring data using patterns: A survey. **Inf. Syst.**, v. 108, p. 101985, 2022. Disponível em: <<https://doi.org/10.1016/j.is.2022.101985>>.
- DEAN, J.; GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. In: BREWER, E. A.; CHEN, P. (Ed.). **Symposium on Operating System Design and Implementation (OSDI)**. USENIX, 2004. p. 137–150. Disponível em: <<http://www.usenix.org/events/osdi04/tech/dean.html>>.
- FU, A. W.-C. et al. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. **The VLDB Journal**, v. 9, p. 154–173, 2000. Disponível em: <<https://doi.org/10.1007/PL00010672>>.

- GUHA, S.; RASTOGI, R.; SHIM, K. Cure: an efficient clustering algorithm for large databases. **Information Systems**, v. 26, n. 1, p. 35–58, 2001. ISSN 0306-4379. Disponível em: <[https://doi.org/10.1016/S0306-4379\(01\)00008-4](https://doi.org/10.1016/S0306-4379(01)00008-4)>.
- HJALTASON, G. R.; SAMET, H. Ranking in spatial databases. In: **International Symposium on Advances in Spatial Databases (SSD)**. Springer, 1995. (LNCS, v. 951), p. 83–95. Disponível em: <https://doi.org/10.1007/3-540-60159-7_6>.
- IKOTUN, A. M. et al. K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. **Inf. Sci.**, v. 622, p. 178–210, 2023. Disponível em: <<https://doi.org/10.1016/j.ins.2022.11.139>>.
- IQBAL, M.; LISSANDRINI, M.; PEDERSEN, T. B. A foundation for spatio-textual-temporal cube analytics. **Inf. Syst.**, v. 108, p. 102009, 2022. ISSN 0306-4379. Disponível em: <<https://doi.org/10.1016/j.is.2022.102009>>.
- JACOX, E. H.; SAMET, H. Metric space similarity joins. **ACM Trans. Database Syst.**, v. 33, n. 2, p. 7:1–7:38, 2008. Disponível em: <<https://doi.org/10.1145/1366102.1366104>>.
- JAIN, A. K. Data clustering: 50 years beyond k-means. **Pattern Recognit. Lett.**, v. 31, n. 8, p. 651–666, 2010. Disponível em: <<https://doi.org/10.1016/j.patrec.2009.09.011>>.
- LEE, K. et al. Parallel data processing with mapreduce: a survey. **SIGMOD Rec.**, v. 40, n. 4, p. 11–20, 2011. Disponível em: <<https://doi.org/10.1145/2094114.2094118>>.
- LESKOVEC, J.; RAJARAMAN, A.; ULLMAN, J. D. **Mining of Massive Datasets, 2nd Ed.** Cambridge University Press, 2014. ISBN 978-1107077232. Disponível em: <<https://doi.org/10.1017/CBO9781139924801>>.
- LEW, M. et al. Content-based multimedia information retrieval: State of the art and challenges. **TOMCCAP**, v. 2, p. 1–19, 02 2006. Disponível em: <<https://doi.org/10.1145/1126004.1126005>>.
- LU, J.; FENG, J. A survey of mapreduce based parallel processing technologies. **China Communications**, v. 11, n. 14, p. 146–155, 2014. Disponível em: <<http://doi.org/10.1109/CC.2014.7085615>>.
- MICO, M. L.; ONCINA, J.; VIDAL, E. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESAs) with linear preprocessing time and memory requirements. **Pattern Recognition Letters**, v. 15, n. 1, p. 9–17, 1994. ISSN 0167-8655. Disponível em: <[http://dx.doi.org/10.1016/0167-8655\(94\)90095-7](http://dx.doi.org/10.1016/0167-8655(94)90095-7)>.
- OLIVEIRA, W. D. et al. Similarity grouping by influence: Exploring result diversification in similarity group-by operators. In: **Simpósio Brasileiro de Banco de Dados (SBBd)**. SBC, 2023. p. 402–407. Disponível em: <<https://doi.org/10.5753/sbbd.2023.233430>>.
- SAMET, H. **Foundations of Multidimensional and Metric Data Structures**. San Francisco, CA: Morgan Kaufmann, 2006. Disponível em: <<https://shop.elsevier.com/books/foundations-of-multidimensional-and-metric-data-structures/samet/978-0-12-369446-1>>.

SILVA, A. P.; RAZENTE, H. Desempenho de operações distribuídas de agrupamento por similaridade em dados de alta dimensionalidade por meio da vp-tree. In: **Proceedings of the 39th Brazilian Symposium on Databases (SBBD 2024)**. Florianópolis, SC, Brazil, October 14-17, 2024: SBC, 2024. p. 195–206. ISSN 2763-8979. Disponível em: <<https://doi.org/10.5753/sbbd.2024.240864>>.

SILVA, Y. N.; AREF, W. G.; ALI, M. H. Similarity group-by. In: **International Conference on Data Engineering (ICDE)**. IEEE, 2009. p. 904–915. Disponível em: <<https://doi.org/10.1109/ICDE.2009.113>>.

SILVA, Y. N. et al. Diversity similarity join for big data. In: **International Conference on Similarity Search and Applications (SISAP)**. Springer, 2023. (LNCS, v. 14289), p. 238–252. Disponível em: <https://doi.org/10.1007/978-3-031-46994-7_20>.

_____. Similarity grouping in big data systems. In: **International Conference on Similarity Search and Applications (SISAP)**. Springer, 2019. (LNCS, v. 11807), p. 212–220. Disponível em: <https://doi.org/10.1007/978-3-030-32047-8_19>.

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

STONEBRAKER, M.; PAVLO, A. What goes around comes around... and around... **SIGMOD Rec.**, v. 53, n. 2, p. 21–37, 2024. Disponível em: <<https://doi.org/10.1145/3685980.3685984>>.

TANG, M. et al. Similarity group-by operators for multi-dimensional relational data. **IEEE Trans. Knowl. Data Eng.**, v. 28, n. 2, p. 510–523, 2016. Disponível em: <<https://doi.org/10.1109/TKDE.2015.2480400>>.

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

TRAINA-JR., C. et al. 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>>.

VERMA, A.; MANSURI, A. H.; JAIN, N. Big data management processing with hadoop mapreduce and spark technology: A comparison. In: **Symposium on Colossal Data Analysis and Networking (CDAN)**. [s.n.], 2016. p. 1–4. Disponível em: <<https://doi.org/10.1109/CDAN.2016.7570891>>.

YIANILOS, P. N. Data structures and algorithms for nearest neighbor search in general metric spaces. In: **ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA)**. ACM/SIAM, 1993. p. 311–321. Disponível em: <<http://dl.acm.org/citation.cfm?id=313559.313789>>.

ZAHARIA, M. et al. Apache spark: a unified engine for big data processing. **Commun. ACM**, v. 59, n. 11, p. 56–65, 2016. Disponível em: <<https://doi.org/10.1145/2934664>>.

ZEZULA, P. et al. **Similarity Search: The Metric Space Approach (Advances in Database Systems)**. Springer, 2006. Disponível em: <<https://doi.org/10.1007/0-387-29151-2>>.