Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Paralelização e Aceleração em GPU de Algoritmos de Inteligência Artificial: Classificação e Agrupamento

Diego Sanches Nere dos Santos

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Luiz Carlos Bambirra Torres

Março, 2026
João Monlevade–MG

Diego Sanches Nere dos Santos

Paralelização e Aceleração em GPU de Algoritmos de Inteligência Artificial: Classificação e Agrupamento

Orientador: Luiz Carlos Bambirra Torres

Monografia apresentada ao curso de Engenharia de Computação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Março de 2026

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

S237p Santos, Diego Sanches Nere dos.
Paralelização e aceleração em GPU de algoritmos de inteligência artificial [manuscrito]: classificação e agrupamento. / Diego Sanches Nere dos Santos. - 2026.
62 f.: il.: color., gráf., tab..

Orientador: Prof. Dr. Luiz Carlos Bambirra Torres.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Engenharia de Computação .

1. Análise por agrupamento. 2. Aprendizado do computador. 3. Classificação. 4. Computação de alto desempenho. 5. Inteligência artificial. 6. Processamento paralelo (Computadores). 7. Teoria dos grafos. I. Torres, Luiz Carlos Bambirra. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004.272:004.85

Bibliotecário(a) Responsável: Flavia Reis - CRB6/2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Diego Sanches Nere dos Santos

Paralelização e Aceleração em GPU de Algoritmos de Inteligência Artificial: Classificação e Agrupamento

Monografia apresentada ao Curso de Engenharia de Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação

Aprovada em 05 de março de 2026

Membros da banca

Doutor Luiz Carlos Bambirra Torres - Orientador (Universidade Federal de Ouro Preto)
Doutor Eduardo da Silva Ribeiro - (Universidade Federal de Ouro Preto)
Mestre Murilo Vale Ferreira Menezes - (QuintoAndar)
Mestre Vítor Mourão Hanriot - (Technium)

Luiz Carlos Bambirra Torres, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 17/03/2026

Documento assinado eletronicamente por **Luiz Carlos Bambirra Torres**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 17/03/2026, às 08:05, 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 http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **1076160** e o código CRC **01EB6B71**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

A minha família e aos meus amigos
por todo o incentivo e companheirismo durante essa jornada.

Agradecimentos

A Deus, primeiramente, por ser minha força nos dias difíceis, minha luz nos momentos de incerteza e minha base para seguir adiante. Sem Sua presença e sabedoria, nada disso teria sido possível.

Aos meus pais, que nunca soltaram minha mão, mesmo quando os caminhos pareciam incertos. Foram eles meu alicerce, meu abrigo e minha maior fonte de incentivo, sempre presentes com apoio incondicional, compreensão e dedicação, permitindo que eu pudesse me dedicar integralmente a este sonho.

Aos meus irmãos, pela parceria constante, pelos conselhos sinceros e pelo incentivo em cada etapa dessa jornada. A presença, o apoio e as orientações compartilhadas fizeram toda a diferença, fortalecendo minha caminhada e me mostrando que nunca estive sozinho.

Aos amigos, que dividiram comigo não apenas conquistas, mas também dúvidas, cansaços e esperanças. Obrigado por cada palavra de apoio, por cada gesto de parceria e por tornarem essa jornada mais humana e verdadeira.

Aos professores, que com dedicação e generosidade compartilharam não apenas conhecimento, mas experiências, valores e inspiração. Cada ensinamento deixou marcas que levarei comigo muito além da vida acadêmica.

E, por fim, a todos que fizeram parte dessa trajetória de maneira especial. Há pessoas que passam por nossa vida e, mesmo que não permaneçam, deixam marcas profundas, aprendizados silenciosos e lembranças que o tempo não apaga. Por tudo o que foi vivido, por tudo o que foi compartilhado e por tudo o que, de alguma forma, contribuiu para que eu me tornasse quem sou hoje, deixo aqui minha sincera gratidão.

"Science is more than a body of knowledge; it is a way of thinking."

— Carl Sagan (1934 – 1996),
in: The Demon-Haunted World: Science as a Candle in the Dark.

Resumo

Este projeto tem como objetivo investigar como estratégias de paralelização podem acelerar algoritmos de Inteligência Artificial (IA) baseados em geometria computacional, por meio do uso de Graphics Processing Unit (GPU) - Unidades de Processamento Gráfico. O trabalho foca em duas aplicações principais que utilizam o Grafo de Gabriel como estrutura: a aceleração de classificadores de margem larga como o NN-clas e a proposição de um novo algoritmo de clusterização. As GPUs oferecem uma arquitetura massivamente paralela, adequada para a custosa etapa de construção de Grafos de Gabriel ($O(N^3)$), o que possibilita ganhos significativos de desempenho. Essa pesquisa avalia o impacto de diferentes granularidades de *threads* na construção do grafo e demonstra como a mesma estrutura pode ser utilizada tanto para algoritmos supervisionados quanto para os não supervisionadas. A aceleração pode ser proveitosa em áreas críticas como saúde e engenharia, onde a agilidade no processamento de grandes volumes de dados é essencial.

Palavras-chaves: paralelismo. GPU. aceleração. inteligência artificial. grafos. aprendizado de máquina. computação de alto desempenho.

Abstract

This project aims to investigate how parallelization strategies can accelerate Artificial Intelligence (AI) algorithms based on computational geometry through the use of Graphics Processing Unit (GPU). The work focuses on two main applications that employ the Gabriel Graph as an underlying structure: the acceleration of large-margin classifiers, such as NN-clas, and the proposal of a novel clustering algorithm. GPUs provide a massively parallel architecture that is well suited to the computationally expensive Gabriel Graph construction step, which has a time complexity of ($O(N^3)$), enabling significant performance gains. This research evaluates the impact of different thread granularities on graph construction and demonstrates how the same structure can be leveraged for both supervised and unsupervised learning algorithms. Such acceleration is particularly beneficial in critical areas such as healthcare and engineering, where fast processing of large volumes of data is essential.

Key-words: parallelism. GPU. acceleration. artificial intelligence. graphs. machine learning. high-performance computing.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

GPU Graphics Processing Unit

SM Streaming Multiprocessors

SIMT Single Instruction, Multiple Threads

CUDA Compute Unified Device Architecture

IA Inteligência Artificial

SVM Máquinas de Vetores de Suporte

KNN K-Nearest Neighbors

AS Arestas de Suporte

IVIs Índices de Validação Interna

Lista de símbolos

Γ	Letra grega Gama
Λ	Lambda
ζ	Letra grega minúscula zeta
$\in$	Pertence
$\forall$	Para todo (quantificador universal)
$\setminus$	Diferença de conjuntos (exceto)
R^n	Espaço Euclidiano de dimensão n
$\|\| \cdot \|\|$	Norma ou distância Euclidiana
$O(\cdot)$	Ordem de complexidade assintótica (Big-O)
S	Conjunto de dados de treinamento
N	Número total de amostras ou vértices
x_i	i-ésima amostra do conjunto de dados (vetor de características)
y_i	Rótulo da classe associado à amostra x_i
G	Grafo definido pelo par (V, A)
V	Conjunto de vértices do grafo
A	Conjunto de arestas do grafo
AS	Conjunto de Arestas de Suporte
$Gr(x_i)$	Vizinhança de Gabriel do vértice x_i
$q(x_i)$	Medida de qualidade do vértice x_i para filtragem de ruído
t^+, t^-	Limiares de corte para filtragem de classes positivas e negativas
k	Número de classes ou número de clusters (dependendo do contexto)
$s(i)$	Coeficiente de Silhouette para o ponto i
σ_i	Dispersão média (desvio padrão) do cluster i

$Tr(\cdot)$	Traço de uma matriz
$\Delta(C_l)$	Diâmetro máximo do cluster C_l
v_i	Centróide do cluster i
T_{corte}	Limiar de corte de distância para o algoritmo de clusterização proposto

Sumário

1 Introdução

1.1 Motivação

O crescimento exponencial na geração de dados nos diferentes setores tem impulsionado o uso de algoritmos de Inteligência Artificial (IA) para classificação, análise e tomada de decisão. Porém, muitas dessas ferramentas são implementadas de forma sequencial, o que podem ser ineficientes em tempo de execução, especialmente quando aplicadas a grandes volumes de dados.

Nas últimas décadas, o avanço de tecnologias digitais, sensores inteligentes, redes sociais e sistemas de monitoramento tem produzido conjuntos de dados cada vez maiores e mais complexos. Esse fenômeno, frequentemente associado ao conceito de *Big Data*, impõe desafios significativos no processamento, armazenamento e análise eficiente das informações (CHEN; MAO; LIU, 2014; HAN; PEI; TONG, 2022). Técnicas tradicionais de processamento sequencial tornam-se limitadas diante da necessidade de respostas rápidas e da análise em larga escala, especialmente em cenários que demandam processamento em tempo real (GANDOMI; HAIDER, 2015).

Esse fator se torna crítico tanto para a classificação de novos dados quanto para a análise não supervisionada, onde o objetivo é descobrir padrões inerentes aos dados sem rótulos prévios. Em ambos os casos, a construção da estrutura de vizinhança é a etapa mais custosa. Em aplicações de tempo real, como diagnósticos médicos e sistemas de engenharia, atrasos nesse processamento podem comprometer a eficácia do sistema (ZHANG et al., 2023; ÇOLHAK et al., 2025).

A construção de estruturas de vizinhança desempenha papel fundamental em diversos algoritmos de aprendizado de máquina, especialmente aqueles baseados em grafos e métodos geométricos. Entre essas estruturas, o Grafo de Gabriel destaca-se por preservar propriedades geométricas relevantes, permitindo representar relações espaciais entre pontos de forma eficiente e mantendo conexões que refletem a proximidade entre amostras. Essa característica torna o grafo adequado para aplicações em classificação de padrões, análise de agrupamentos e processamento de imagens.

Entretanto, a construção do Grafo de Gabriel apresenta elevada complexidade computacional, principalmente quando aplicada a conjuntos de dados de grande dimensionalidade. O cálculo das relações entre pares de pontos exige elevado custo computacional, o que limita sua aplicação em cenários que exigem alto desempenho computacional.

As Unidades de Processamento Gráfico (Graphics Processing Unit (GPU)), por sua arquitetura massivamente paralela, tornaram-se uma opção eficiente para solucionar esses

desafios. Originalmente desenvolvidas para processamento gráfico, as GPUs evoluíram para plataformas de computação de propósito geral, permitindo a execução paralela de milhares de operações simultaneamente. Essa capacidade possibilita acelerar significativamente algoritmos que envolvem processamento intensivo de dados, especialmente aqueles baseados em cálculos independentes entre pares de elementos.

A utilização da plataforma Compute Unified Device Architecture (CUDA) permite explorar de forma eficiente os recursos computacionais das GPUs, oferecendo mecanismos para gerenciamento de memória, sincronização de *threads* e organização hierárquica de processamento. Diversos estudos demonstram que a paralelização de algoritmos geométricos e de aprendizado de máquina pode proporcionar ganhos expressivos de desempenho, reduzindo tempos de execução e ampliando a viabilidade de aplicação desses métodos em problemas reais (NICKOLLS et al., 2008; SANDERS; KANDROT, 2010).

Além da aceleração de algoritmos existentes, a computação paralela abre possibilidades para o desenvolvimento de novas abordagens metodológicas. Métodos que anteriormente eram inviáveis devido ao alto custo computacional passam a ser explorados com maior profundidade, permitindo avanços no desenvolvimento de técnicas de clusterização e classificação baseadas em grafos geométricos.

Dessa forma, investigar estratégias de paralelização aplicadas à construção e utilização do Grafo de Gabriel torna-se relevante tanto do ponto de vista científico quanto tecnológico, contribuindo para o desenvolvimento de soluções mais eficientes para análise de grandes volumes de dados.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é investigar e avaliar o impacto de estratégias de paralelização utilizando a plataforma Compute Unified Device Architecture (CUDA) na otimização de algoritmos baseados no Grafo de Gabriel. O estudo visa acelerar um classificador de margem larga existente e propor um novo algoritmo de clusterização autoral, analisando o ganho de eficiência e a viabilidade de execução em grandes volumes de dados.

1.2.2 Objetivos Específicos

Para atingir o objetivo geral, busca-se os seguintes objetivos específicos:

- Implementar uma versão sequencial do classificador para validar a lógica e aferir seu desempenho em conjuntos de dados de pequena escala.

- Desenvolver uma implementação paralela em CUDA, onde o cálculo de cada par de pontos do grafo é mapeado a uma *thread* distinta.
- Desenvolver uma implementação paralela otimizada, ajustando a granularidade do processamento para que cada *thread* seja responsável pelo cálculo de múltiplos pares de pontos, visando reduzir o gargalo de gerenciamento de *threads*.
- Analisar a complexidade de espaço do classificador implementado.
- Avaliar e comparar o desempenho das diferentes implementações.
- Analisar o impacto das estratégias de otimização de *threads* no desempenho.
- Propor, implementar e validar um novo algoritmo de clusterização que utiliza a estrutura do Grafo de Gabriel.
- Documentar os resultados, as estratégias de otimização e as conclusões obtidas.

1.3 Estrutura do Trabalho

Este trabalho está organizado da seguinte forma: O Capítulo 3 apresenta a fundamentação teórica sobre classificadores, computação paralela com GPUs e a plataforma CUDA. São discutidos conceitos fundamentais necessários para o entendimento das técnicas utilizadas, incluindo estruturas de grafos geométricos e métodos de aprendizado de máquina.

O Capítulo 4 detalha a metodologia de desenvolvimento, incluindo o ambiente utilizado, as estratégias de paralelização e as métricas de avaliação. Também são descritos os conjuntos de dados utilizados e os critérios adotados para validação dos experimentos.

Os resultados obtidos são apresentados e discutidos no Capítulo 5, onde são analisados os ganhos de desempenho das implementações paralelas e a eficiência do algoritmo de clusterização proposto.

Finalmente, o Capítulo 6 resume as conclusões do trabalho, suas contribuições científicas e tecnológicas, bem como sugestões para trabalhos futuros e possíveis extensões da pesquisa desenvolvida.

2 Revisão bibliográfica

Tabela 1 – Trabalhos relacionados utilizados na construção desta monografia.

Estudo	**Ano**	**Metodologia**	**Contribuição principal**	**Relação com este trabalho**
(GADE et al., 2018)	2017	Proposição de classificador geométrico	Define o NN-clas e sua construção do Grafo de Gabriel	Base conceitual do classificador estudado; não trata de paralelização
(GARCIA; de Carvalho; LORENA, 2015)	2015	Análise de ruído em dados	Propõe filtragem para melhorar classificadores geométricos	Utilizado na etapa de pré-processamento; execução inteiramente sequencial
(ARIAS-GARCIA et al., 2024)	2024	Implementação em hardware (FPGA)	Aceleração de classificadores geométricos via hardware dedicado	Mostra a relevância de acelerar classificadores; não usa GPU/CUDA
(ÇOLHAK et al., 2025)	2025	Benchmark experimental	Compara desempenho de ML em GPU vs CPU	Fundamenta a motivação para uso de GPU; foco em ML tradicional
(ZHANG et al., 2023)	2023	Framework acelerado em GPU	Integra simulação neural e IA em plataforma CUDA	Demonstra capacidade de GPUs para cargas massivas; não trata grafos geométricos
(EON et al., 2021)	2021	Revisão técnica	Análise de deep learning acelerado por GPU	Reforça a eficiência das GPUs, mas não aborda class. geométricos
(MAGNI; DUBACH; O'BOYLE, 2014)	2014	Otimização de kernels CUDA	Otimização automática de Grosseamento de Threads (*Thread Coarsening*) para melhorar o desempenho em GPU.	Base teórica fundamental para a otimização da granularidade.

O classificador NN-clas (GADE et al., 2018), constitui a base conceitual da parte inicial deste trabalho. Nele, o processo de decisão é estruturado a partir do Grafo de Gabriel, cujas arestas definem regiões de margem entre classes. Embora eficiente em termos de acurácia, o método apresenta elevado custo computacional devido à necessidade de avaliar todos os pares de pontos durante a construção do grafo. No contexto de pré-processamento, (GARCIA; de Carvalho; LORENA, 2015) analisam o impacto de ruído e propõem uma técnica de filtragem projetada para melhorar a qualidade de classificadores geométricos.

Com relação à exploração de arquiteturas paralelas, diversos estudos demonstram o potencial das GPUs para acelerar tarefas computacionalmente intensivas. (ÇOLHAK et al., 2025) compara o desempenho de bibliotecas de aprendizado de máquina executadas em CPU e GPU, mostrando ganhos no throughput e tempo de execução quando a arquitetura paralela é utilizada. De forma complementar, (ARIAS-GARCIA et al., 2024) apresentam uma otimização em FPGA para classificadores geométricos, evidenciando que reduzir o custo computacional dessas técnicas é um problema relevante na literatura.

Para maximizar a eficiência em arquiteturas massivamente paralelas, a otimização da granularidade da computação se torna crucial. (MAGNI; DUBACH; O'BOYLE, 2014) demonstram que a Otimização Automática do Grosseamento de Threads (*Thread Coarsening*) é essencial, pois o mapeamento ingênuo de uma thread por elemento de cálculo pode introduzir overhead de gerenciamento e limitar o speedup. O método de *Thread Coarsening* é fundamental para alcançar um equilíbrio entre o paralelismo massivo e a minimização do gargalo de agendamento de kernel.

(ZHANG et al., 2023) apresentam um framework computacional em GPU que integra simulação neural e inteligência artificial, ilustrando como arquiteturas massivamente paralelas podem lidar eficientemente com cálculos complexos e estruturas de dados extensas. Por fim, (EON et al., 2021) discutem em profundidade o papel das GPUs no treinamento de modelos de *deep learning*, demonstrando como a arquitetura SIMT permite explorar paralelismo massivo. Esses estudos reforçam a aplicabilidade das GPUs na aceleração de métodos computacionais.

Em conjunto, os trabalhos revisados indicam que: (i) classificadores de margem larga baseados em grafos apresentam alto custo computacional; (ii) existem esforços para melhorar sua performance em hardware especializado; e (iii) GPUs têm se mostrado eficazes para acelerar tarefas envolvendo operações paralelizáveis. Dessa forma, este trabalho busca contribuir ao investigar um aspecto ainda não fortemente explorado na literatura: a análise experimental do impacto da granularidade de processamento por *thread* na etapa mais custosa do NN-clas, a construção do Grafo de Gabriel. Não foram encontrados trabalhos que utilizem CUDA especificamente para paralelizar a construção do Grafo de Gabriel com a estratégia de *Thread Coarsening* proposta.

3 Fundamentação Teórica

3.1 Classificadores

3.1.1 Definição de Problemas de Classificação

A classificação é formalmente definida como uma tarefa de aprendizagem supervisionada cujo objetivo é atribuir instâncias de dados a categorias ou classes pré-definidas e discretas. O modelo resultante, conhecido como classificador, aprende a partir de um conjunto de dados de treino e, subsequentemente, prevê o rótulo de classe para novas instâncias (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

3.1.2 Principais Paradigmas de Classificadores

3.1.2.1 Classificação Binária

A classificação binária consiste em atribuir uma instância a uma de duas classes mutuamente exclusivas. Os rótulos são frequentemente representados como (0,1), (-1, 1), ("sim", "não") ou ("positivo", "negativo"). Alguns exemplos mais conhecidos são: detecção de fraude em transações, diagnóstico médico de uma condição específica e previsão de *churn* de clientes. Alguns algoritmos, como a Regressão Logística e as Máquinas de Vetores de Suporte (SVM), são inerentemente desenhados para problemas de classificação binária (HASTIE; TIBSHIRANI; FRIEDMAN, 2009). O classificador NN-clas se encaixa nessa categoria.

3.1.2.2 Classificação de Múltiplas Classes

A classificação de múltiplas classes é uma generalização da classificação binária para problemas com mais de duas classes. A principal restrição é que as classes sejam mutuamente exclusivas, o que significa que cada instância deve ser atribuída a exatamente uma classe de um conjunto de $k > 2$ classes. Exemplos comuns incluem o reconhecimento ótico de caracteres onde um caractere pode ser um de muitos dígitos ou letras, a classificação de espécies de plantas ou a análise de sentimento com múltiplas categorias (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

3.1.2.3 Desafio da Classificação Desbalanceada

A classificação desbalanceada é um desafio que pode ocorrer a qualquer um dos tipos previamente mencionados. Elas ocorrem quando a distribuição de classes entre as instâncias é desigual. Embora apresentado como desafio, o desbalanceamento de classes é

muito comum em diferentes domínios. Problemas como detecção de fraude, diagnóstico de doenças raras ou previsões são exemplos populares. Nestes cenários, o evento de interesse é raro em comparação com o estado normal.

A principal implicação é que modelos treinados em dados desbalanceados tendem a ser enviesados para a classe majoritária. Podem exibir alta acurácia, em geral, por prever sempre a classe dominante, mas falham na identificação da classe minoritária, o que é frequentemente a de maior interesse. Neste contexto, a métrica de acurácia se torna menos valiosa. Portanto, tal discrepância evidencia a importância de diferentes métricas como precisão, revocação (*recall*) e F1-score, além de técnicas de amostragem estratificadas.

3.1.3 Aprendizado Não Supervisionado e Clusterização

Enquanto a classificação supervisionada depende de rótulos prévios para treinar um modelo, o aprendizado não supervisionado lida com dados onde a variável de resposta é desconhecida ou inexistente. O objetivo principal é inferir a estrutura subjacente ou padrões pertencentes aos dados (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

A clusterização é uma das tarefas fundamentais do aprendizado não supervisionado. Ela consiste em dividir um conjunto de dados em subconjuntos (*clusters*), de tal forma que elementos dentro de um mesmo grupo sejam mais similares entre si do que com elementos de outros grupos (JAIN, 2010). Diferentes critérios de similaridade podem ser adotados, como distância euclidiana, densidade ou conectividade em um grafo.

3.2 Classificador de Margem Larga Baseado em Distância

Os classificadores de margem larga baseados em distância são uma classe de algoritmos que obtêm a informação estrutural dos dados através da construção de um grafo. No contexto deste trabalho, o algoritmo estudado utiliza o Grafo de Gabriel, que, por sua vez, depende apenas da distância entre as amostras de treinamento para sua construção.

O exemplo que será utilizado nesse trabalho é o classificador *NN-clas* (ARIAS-GARCIA et al., 2024), que servirá como objeto de estudo para a paralelização e análise de desempenho. O *NN-clas* consiste em um classificador de margem larga baseado em distância, construído com base no Grafo de Gabriel. Na primeira fase, o método deve construir o grafo com os dados de treinamento e, em seguida, realizar a filtragem de dados ruidosos, com o objetivo de eliminar pontos que comprometem a qualidade da classificação. Após a limpeza, o grafo é reconstruído somente com os vértices restantes e a etapa de identificação das arestas de suporte é iniciada. Na etapa de classificação, o algoritmo calcula a distância entre cada amostra de teste com os vértices das arestas de suporte, para que seja classificada com o rótulo do dado mais próximo.

3.2.1 Construção do Grafo de Gabriel

O conceito de Grafo de Gabriel foi originalmente introduzido por (GABRIEL; SOKAL, 1969) no contexto de análise geométrica. Seja um conjunto de dados de treinamento $S = \{x_i\}_{i=1}^{N} \subset R^n$ onde cada amostra pode estar associada a um rótulo $y_i \in \{+1, -1\}$. Define-se o grafo não direcionado $G = (V, A)$no qual cada vértice $v_i \in V$ representa um ponto $v_i \in S$. Existe uma aresta $(x_i, x_j) \in A$ se, e somente se, nenhum outro ponto $x_k \in S$, com $k \neq i, j$, estiver contido no interior da hiperesfera com diâmetro igual à distância entre x_i e x_j, o que forma o grafo que pode ser observado na Figura 1. Essa condição pode ser expressa pela inequação 3.1:

$$\|x_i - x_j\|^2 \leq \|x_i - x_k\|^2 + \|x_j - x_k\|^2, \quad \forall x_k \in V \setminus \{x_i, x_j\} \tag{3.1}$$

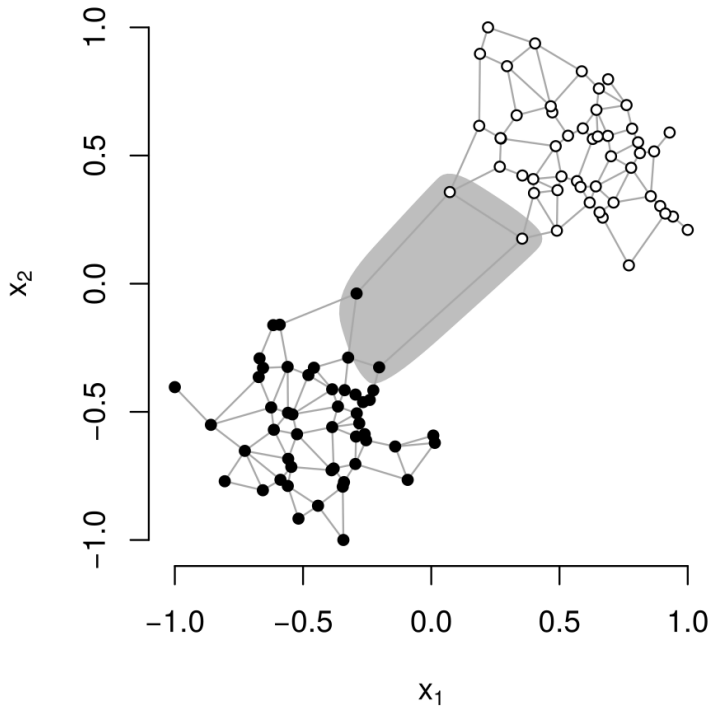

Figura 1 – Exemplo de um conjunto de dados de um problema de classificação binária com o correspondente grafo de Gabriel e o subgrafo destacado. **Fonte:** (GADE et al., 2018).

3.2.2 Filtragem de Ruído e Detecção de Arestas de Suporte

O Grafo de Gabriel, também possui um conjunto de Arestas de Suporte (AS), o qual representa todas as arestas de A que possuem um par de vértices (x_i, x_j) de classes distintas. Caso não haja sobreposição entre as amostras, é possível afirmar que os vértices de AS estão localizados na margem de separação entre as classes.

Para que a metodologia descrita seja eficaz em cenários com sobreposição ou ruído entre as classes, como visto na Figura 2, é necessário aplicar uma etapa de filtragem. Uma

abordagem aplicada é a proposta por (GARCIA; de Carvalho; LORENA, 2015), para eliminar as amostras ruidosas antes da detecção do conjunto de Arestas de Suporte (AS). O processo de filtragem pode ser descrito pelos seguintes passos.

1. Inicialmente é calculado o $q(x_i)$ para todo $x_i \in G$ conforme Equação 3.2. Os valores de $q(x_i)$ são agrupados por classe, de forma que Q^+ e Q^- representam a medida de qualidade entre as classes $+1$ e -1, respectivamente;

2. Posteriormente, deve-se calcular, de acordo com a Equação 3.3, os valores dos limiares t^+ e t^- de cada classe como a média das medidas de qualidade pertencentes a Q^+ e Q^-;

3. Finalmente, são removidos todos os vértices de G que possuem $q(x_i)$ menores que t^+ e t^-.

4. O resultado então pode ser observado na figura 3

$$q(x_i) = \frac{|\hat{Gr}(x_i)|}{|Gr(x_i)|} \tag{3.2}$$

$$t^+ = \frac{\sum\limits_{q(x_i)\in Q^+} q(x_i)}{|Q^+|}, \quad t^- = \frac{\sum\limits_{q(x_i)\in Q^-} q(x_i)}{|Q^-|} \tag{3.3}$$

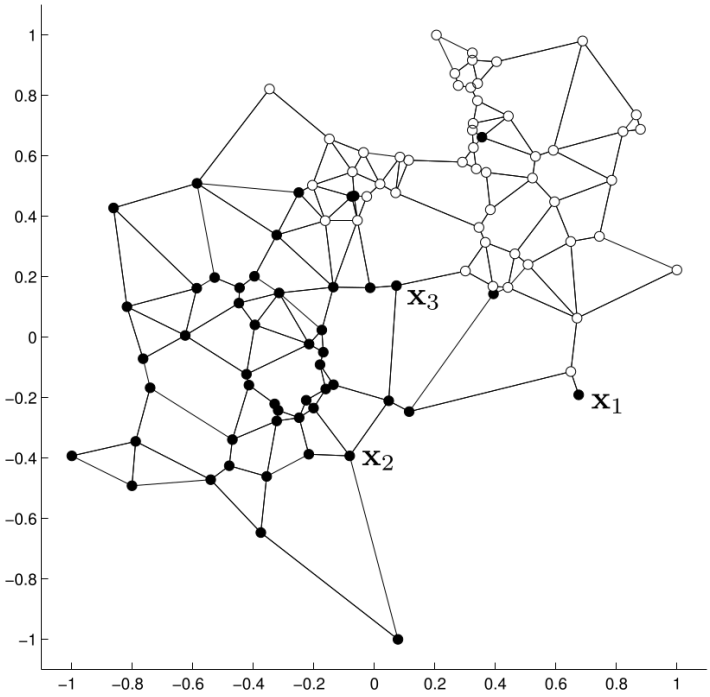

Figura 2 – *Dataset* com sobreposição. **Fonte:** (GADE et al., 2018).

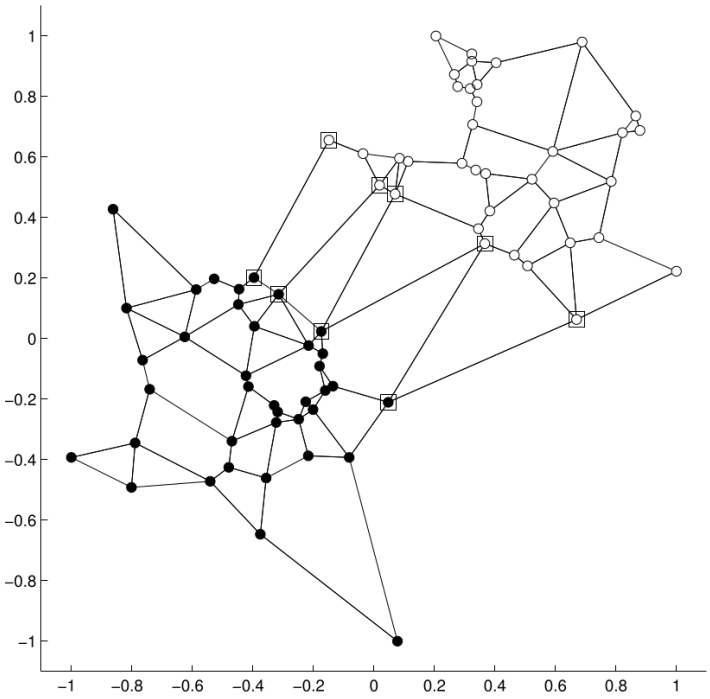

Figura 3 – *Dataset* após remover sobreposição. **Fonte:** (GADE et al., 2018).

3.2.3 Etapa de Classificação

eraApós a filtragem de ruído e a identificação das Arestas de Suporte (AS), a classificação de novas amostras é realizada. O método *NN-clas* (GADE et al., 2018) utiliza como referência apenas os vértices que compõem o conjunto AS, os quais são referidos como "vértices de borda".

Para classificar uma nova amostra de teste x, o algoritmo calcula a distância entre x e todos os vértices de borda. A amostra x recebe então o rótulo do vértice de borda que apresentar a menor distância.

Esta técnica de decisão é funcionalmente idêntica ao algoritmo K-Nearest Neighbors (KNN) com o parâmetro $k = 1$. No entanto, ao invés de comparar a nova amostra com todos os pontos do conjunto de treinamento S, a comparação é feita apenas com o subconjunto de vértices de borda, o que reduz significativamente o custo computacional da classificação.

3.2.4 Agrupamento Baseado em Grafos e Componentes Conexos

A modelagem de problemas de agrupamento através de grafos transforma o conjunto de dados em uma estrutura $G = (V, A)$, onde os vértices V representam as amostras e as arestas A representam a similaridade ou proximidade entre elas. Neste paradigma, o problema de encontrar grupos nos dados é convertido em um problema de particionamento de grafos.

Uma abordagem direta e eficaz para a definição de *clusters* em grafos esparsos,

como o Grafo de Gabriel, é a identificação de Componentes Conexos. Um componente conexo é definido como um subgrafo no qual existe um caminho entre qualquer par de vértices, e que não está conectado a nenhum vértice adicional no supergrafo (CORMEN et al., 2002).

No contexto deste trabalho, a estratégia de clusterização proposta utiliza a propriedade geométrica do Grafo de Gabriel. Se dois pontos estão conectados por uma aresta no grafo, assume-se que eles pertencem à mesma estrutura local. Dessa forma, a identificação de todos os componentes conexos do grafo resulta naturalmente na segmentação dos dados em *clusters* de formatos arbitrários, sem a necessidade de definir previamente o número de grupos ou parâmetros complexos de densidade.

3.2.5 Estrutura de Dados Union-Find (Disjoint Set Union)

A estrutura de dados Union-Find, é importante para o gerenciamento de elementos particionados em subconjuntos disjuntos e não sobrepostos. Ela é muito utilizada em algoritmos de grafos para determinar a conectividade de componentes e para a detecção de ciclos, sendo a base para algoritmos clássicos como o de Kruskal para Árvores Geradoras Mínimas (CORMEN et al., 2002).

- Find (Encontrar): Determina a qual subconjunto um determinado elemento pertence. Geralmente, isso é feito retornando um elemento representante do conjunto. Se Find(x) == Find(y), então x e y estão no mesmo conjunto.
- Union (Unir): Combina dois subconjuntos distintos em um único conjunto. Essa operação geralmente conecta a raiz de uma árvore à raiz da outra.

3.2.6 Método do Cotovelo (Elbow Method)

Para o algoritmo de clusterização foi utilizado o Método do Cotovelo (*Elbow Method*) que é uma heurística utilizada para definir limiares de corte em algoritmos hierárquicos. O método baseia-se na análise da variação explicada em função de um parâmetro variável.

Graficamente, o método mapeia o valor de uma métrica de custo no eixo Y contra o parâmetro de interesse no eixo X. O objetivo é identificar o "cotovelo"da curva: o ponto onde o ganho marginal de desempenho cai drasticamente, formando um ângulo agudo no gráfico. Matematicamente, esse ponto representa o equilíbrio entre a complexidade do modelo e a qualidade do ajuste. Em implementações automatizadas, o ponto de cotovelo pode ser definido como o ponto da curva que maximiza a distância perpendicular até a linha reta traçada entre o primeiro e o último ponto da distribuição (THORNDIKE, 1953).

3.3 Métricas de Avaliação de *Clusters*

Para a validação quantitativa dos agrupamentos, adotou-se uma estratégia de validação interna, fundamental em cenários onde os rótulos reais das classes não estão disponíveis ou não são utilizados para o treinamento. Conforme a taxonomia apresentada por (EZUGWU et al., 2022), os Índices de Validação Interna (IVIs) avaliam a qualidade da partição baseando-se exclusivamente nas características intrínsecas dos dados, como a geometria e a distribuição espacial.

Para garantir uma análise robusta e mitigar o viés de uma única métrica, foram selecionados cinco índices distintos que ponderam, de diferentes formas, os critérios de compacidade (coesão intra-cluster) e separação (isolamento inter-cluster): Coeficiente de *Silhouette*, Índice Davies-Bouldin, Índice Calinski-Harabasz, Índice de Dunn e Índice Xie-Beni.

3.3.1 Coeficiente de Silhouette

O Coeficiente de *Silhouette* é uma métrica amplamente utilizada que avalia o quão semelhante um objeto é ao seu próprio cluster (coesão) em comparação com outros clusters (separação) (EZUGWU et al., 2022). Para cada ponto i, o valor $s(i)$ é calculado como:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{3.4}$$

Onde $a(i)$ representa a distância média entre o ponto i e todos os outros pontos no mesmo cluster, e $b(i)$ é a distância média mínima entre i e os pontos do cluster vizinho mais próximo. O coeficiente varia de -1 a +1. Valores próximos a **+1** indicam que a instância está bem alocada e distante dos clusters vizinhos, enquanto valores próximos a 0 sugerem sobreposição (EZUGWU et al., 2022).

3.3.2 Índice Davies-Bouldin (DB)

O índice Davies-Bouldin avalia a razão entre a dispersão interna dos clusters e a distância entre eles. O índice calcula a média da similaridade máxima de cada cluster C_i com seu cluster mais semelhante C_j. Matematicamente, é definido por (EZUGWU et al., 2022):

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} \left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \right) \tag{3.5}$$

Onde k é o número de clusters, σ_i é a dispersão média (desvio padrão) do cluster i e $d(c_i, c_j)$ é a distância entre os centróides dos clusters i e j. Como este índice mede a similaridade entre clusters, valores menores indicam melhor qualidade de agrupamento.

3.3.3 Índice Calinski-Harabasz (CH)

Também conhecido como Critério da Razão de Variância, o índice CH mede a relação entre a dispersão inter-cluster e a dispersão intra-cluster. É dado pela equação (EZUGWU et al., 2022):

$$CH = \frac{Tr(B_k)}{Tr(W_k)} \times \frac{N-k}{k-1} \tag{3.6}$$

Onde $Tr(B_k)$ é o traço da matriz de dispersão entre grupos (*inter-cluster*), $Tr(W_k)$ é o traço da matriz de dispersão dentro dos grupos (*intra-cluster*), N é o número total de amostras e k o número de clusters. Valores maiores indicam *clusters* mais densos e bem separados.

3.3.4 Índice de Dunn (DI)

O Índice de Dunn foca na identificação de conjuntos compactos e bem separados, sendo sensível à presença de ruído. É calculado pela razão entre a menor distância inter-cluster e o maior diâmetro intra-cluster (EZUGWU et al., 2022):

$$DI = \min_{1\leq i\leq c}\left\{\min_{j\neq i}\left(\frac{d(C_i,C_j)}{\max_{1\leq l\leq c}\Delta(C_l)}\right)\right\} \tag{3.7}$$

Onde $d(C_i, C_j)$ representa a distância mínima entre clusters e $\Delta(C_l)$ o diâmetro máximo de um cluster. Valores maiores indicam melhor separação e compactação.

3.3.5 Índice Xie-Beni (XB)

O índice de validação Xie-Beni quantifica a compacidade e a separação medindo o erro quadrático médio normalizado pela distância mínima entre os centros dos clusters. Conforme (EZUGWU et al., 2022), é definido como:

$$XB = \frac{\sum_{i=1}^{k}\sum_{x\in C_i}||x-v_i||^2}{N\times \min_{i\neq j}||v_i-v_j||^2} \tag{3.8}$$

Onde v_i é o centróide do cluster i. Valores menores indicam partições mais compactas e separadas.

3.4 Computação Paralela com GPU e a Plataforma CUDA

Embora tenham sido originalmente concebidas para a aceleração de renderização gráfica, GPUs evoluíram para se tornarem plataformas robustas de computação de propósito geral. Diferentemente das CPUs, cuja arquitetura é projetada para minimizar a latência

na execução de fluxos sequenciais complexos, dedicando transistores a grandes caches e previsão de desvios, as GPUs são otimizadas para maximizar a vazão de dados (*throughput*). Sua arquitetura se dedica a unidades de processamento aritmético, permitindo a execução simultânea de milhares de *threads*. Essa característica as torna excepcionalmente eficientes não apenas para gráficos, mas para qualquer algoritmo que exiba alto grau de paralelismo de dados (OWENS et al., 2008).

CUDA é uma plataforma de computação paralela e modelo de programação criado pela NVIDIA que ajuda os desenvolvedores a acelerar suas aplicações aproveitando o poder dos aceleradores de GPU (OH, 2024). O NVIDIA CUDA Toolkit fornece um ambiente de desenvolvimento abrangente para desenvolvedores de C e C++ que criam aplicativos acelerados por GPU (NVIDIA, 2025).

3.4.1 A Arquitetura da GPU para Computação de Alto Desempenho (Modelo SIMT)

As GPUs modernas são projetadas em torno de uma arquitetura fortemente paralela composta por múltiplos Streaming Multiprocessors (SM). Cada SM é um processador independente e autossuficiente, capaz de criar, gerenciar, escalonar e executar centenas de *threads* simultaneamente (NVIDIA Corporation, 2025). Essa capacidade de executar milhares de *threads* concorrentemente é o que diferencia a GPU da CPU tradicional, cujo foco está em latência e execução sequencial.

O modelo de execução adotado pelo CUDA é o Single Instruction, Multiple Threads (SIMT). Nesse modelo, as *threads* são organizadas em grupos de 32 unidades paralelas chamadas *warps*. Cada *warp* executa uma única instrução em múltiplas *threads* de forma simultânea, eles têm seu próprio contador de endereço de instrução e estado de registradores e, portanto, são livres para desviar e executar independentemente (NVIDIA Corporation, 2025).

A eficiência do modelo SIMT depende fortemente da coerência de execução dentro de um *warp*. Todas as *threads* de um mesmo *warp* iniciam na mesma instrução e, idealmente, seguem o mesmo caminho de execução. Quando isso ocorre, o *warp* atinge eficiência total, pois o *hardware* executa uma única instrução para todas as *threads* ativas. Contudo, se as *threads* divergem em uma estrutura condicional ocorre o fenômeno de divergência de *thread*. Nesse caso, o *warp* precisa serializar a execução de cada caminho de código tomado, desativando temporariamente as *threads* que não pertencem ao caminho atual, reduzindo o paralelismo efetivo (NVIDIA Corporation, 2025).

A divergência, entretanto, é restrita ao nível do *warp*; *warps* distintos são executados de forma completamente independente, podendo seguir caminhos de controle diferentes sem interferência entre si. Assim, o desempenho ideal de um *kernel* CUDA é alcançado

quando o código minimiza a divergência intra-*warp*, algo análogo ao alinhamento de acessos à memória em arquiteturas de CPU: não afeta a correção, mas impacta significativamente o desempenho (NVIDIA Corporation, 2025).

3.4.2 Concorrência e Operações Atômicas em GPU

Em arquiteturas de memória compartilhada, o acesso simultâneo de múltiplas *threads* ao mesmo endereço de memória pode levar a condições de corrida, resultando em dados inconsistentes ou corrompidos. Isso ocorre quando uma *thread* tenta ler ou modificar uma variável enquanto outra *thread* está realizando uma operação na mesma posição de memória.

Para solucionar esse problema sem a necessidade de bloqueios complexos, a arquitetura CUDA fornece Operações Atômicas. Uma operação atômica é aquela que garante ser executada como uma unidade indivisível: nenhuma outra *thread* pode observar o estado da operação pela metade. Funções como *atomicAdd*, *atomicMin* ou *atomicCAS* realizam a sequência de leitura-modificação-escrita em um único ciclo de barramento ininterrupto (NVIDIA Corporation, 2025). Embora essenciais para garantir a correção de algoritmos paralelos, é válido destacar que o uso excessivo de operações atômicas pode serializar a execução das threads que competem pelo mesmo recurso, impactando o *throughput* total do sistema.

3.4.3 Aplicabilidade ao Problema de Aceleração

A arquitetura SIMT e o modelo de programação CUDA são diretamente aplicáveis ao problema central deste trabalho. A etapa mais custosa desse tipo de algoritmo é a construção do grafo, que frequentemente envolve o cálculo de distâncias entre todos os pares de pontos de treinamento.

Esta operação possui uma complexidade $O(n^3)$ e é inerentemente paralelizável, pois o cálculo da distância entre os pontos i e j é independente do cálculo entre os pontos k e l. Este cenário, conhecido como *embarrassingly parallel*, é ideal para a arquitetura SIMT.

É possível lançar um *kernel* CUDA onde cada *thread* é responsável pelo cálculo de uma ou mais distâncias. O conjunto de dados de treinamento pode ser pré-carregado na memória global, e as *threads* podem calcular as distâncias em paralelo e guardando os resultados.

O desafio reside em como organizar essa computação e qual a granularidade ideal de trabalho por *thread*. Mapear uma *thread* para cada par de pontos pode gerar um gargalo no dispositivo. Por outro lado, fazer uma *thread* para muitos pares poded subutilizar a GPU. A metodologia deste trabalho deve explorar tal otimização, avaliando diferentes distribuições a fim de melhorar o desempenho.

4 Metodologia e Desenvolvimento

4.1 Ambiente de Desenvolvimento e Ferramentas

O desenvolvimento, a implementação e a avaliação de desempenho dos algoritmos foram conduzidos em ambientes computacionais controlados. Para a etapa de comparação entre as versões sequencial e paralela do classificador NN-clas, utilizou-se a configuração de hardware e software detalhada a seguir:

- Processador (CPU): AMD Ryzen 3 2200G;
- Unidade de Processamento Gráfico (GPU): NVIDIA RTX 4070 Ti, 12 GB VRAM, 7680 núcleos CUDA;
- Memória RAM: 16 GB;
- Sistema Operacional: Linux Mint 21.3;
- Compilador e Ferramentas: nvcc 11.5, G++-9, C++17 Standard.

É importante ressaltar que, nesta configuração específica, existe uma disparidade de desempenho teórica entre a unidade central de processamento e a unidade de processamento gráfico. Reconhece-se que essa diferença de potência pode ocasionar gargalos (*bottlenecks*), especialmente nas etapas de transferência de memória entre *Host* e *Device*. No entanto, optou-se por manter essa configuração e desconsiderar o impacto desse gargalo na latência total do sistema, visto que o objetivo primordial deste estudo é avaliar o ganho de desempenho relativo (*speedup*) e a eficiência computacional dos *kernels* na GPU, isolando o comportamento da aceleração paralela.

Com o objetivo de validar o algoritmo de clusterização proposto em um cenário de hardware mais robusto, realizou-se a sessão de testes e validações em clusters utilizando um segundo ambiente com capacidades de processamento superiores:

- Processador (CPU): AMD Ryzen 9 7900 (24 núcleos) @ 5.49 GHz;
- Unidade de Processamento Gráfico (GPU): NVIDIA GeForce RTX 5070 Ti, 16303 MiB VRAM;
- Memória RAM: 64 GB;
- Sistema Operacional: Ubuntu 24.04.3 LTS;
- Compilador e Ferramentas: nvcc 13.0, G++-9, C++17 Standard.

4.2 Conjuntos de Dados

4.2.1 Sequencial vs. paralelo

Para os experimentos envolvendo a comparação entre execução sequencial e paralela, foram selecionados *datasets* previamente utilizados em (ARIAS-GARCIA et al., 2024). A escolha desses conjuntos de dados justifica-se pelo fato de serem amplamente conhecidos, apresentarem tamanhos reduzidos e diversidade de características, permitindo avaliar o comportamento dos algoritmos em cenários variados e com baixo custo computacional.

4.2.2 Múltiplos pares por thread

Para a etapa de experimentação envolvendo múltiplos pares atribuídos por thread, foram selecionados dois conjuntos de dados públicos: *Internet Firewall* Data e *Diabetes Dataset*.

O *Diabetes Dataset* contém atributos clínicos associados ao diagnóstico de diabetes, tais como nível de glicose no sangue, índice de massa corporal (Body Mass Index — BMI), idade e nível de hemoglobina glicada (HbA1c). Esses atributos são amplamente utilizados em modelos de predição médica. A Figura 4 apresenta a distribuição das classes, evidenciando um desbalanceamento moderado entre indivíduos classificados como não diabéticos (classe -1) e diabéticos (classe 1), neste caso é possível observar alta discrepância na distribuição das classes.

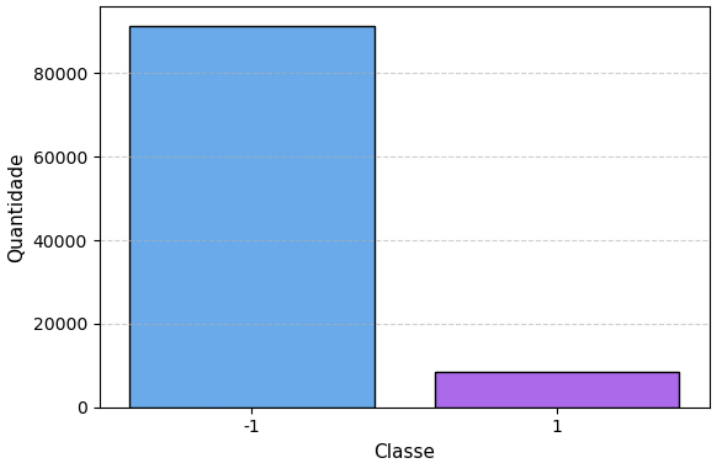

Figura 4 – Análise de distribuição de classes para o conjunto de dados *Diabetes*

O *Internet Firewall Data*, disponível publicamente no repositório *UCI Machine Learning Repository*, refere-se ao monitoramento e à detecção de eventos provenientes de tráfego de rede em um ambiente protegido por *firewall*. O conjunto reúne informações sobre diversos tipos de conexões, incluindo atributos relacionados a portas, protocolos, volume de pacotes e indicadores de anomalias. Seu objetivo principal é fornecer dados

adequados para o desenvolvimento e a avaliação de mecanismos de detecção de intrusões ou comportamentos suspeitos. Esse *dataset* é particularmente interessante por apresentar características típicas de aplicações de segurança cibernética, como alta dimensionalidade, variabilidade temporal e presença de padrões sutis de ataque, o que o torna apropriado para analisar a eficiência de diferentes estratégias de paralelização.

A Figura 5 apresenta a distribuição das classes presentes no conjunto *Internet Firewall Data*. Observa-se um leve desbalanceamento entre as classes rotuladas, com maior quantidade de instâncias pertencentes à classe 1.

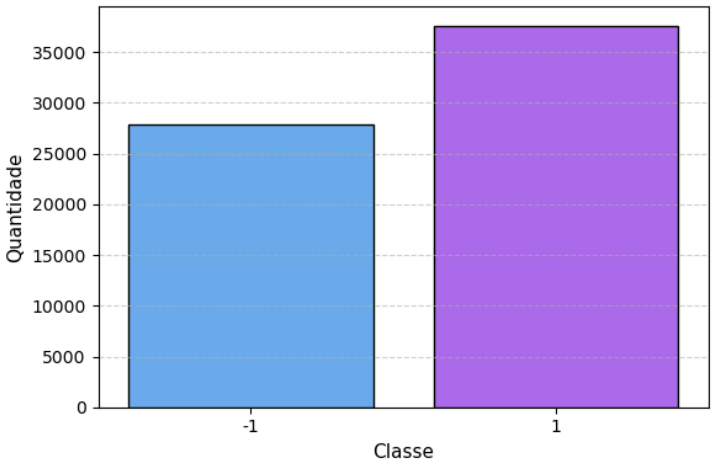

Figura 5 – Análise de distribuição de classes para o conjunto de dados *Firewall*

4.3 Dataset usado para clusterização

Para uma avaliação robusta do algoritmo proposto, foram selecionados 14 conjuntos de dados, divididos estrategicamente entre dados sintéticos e dados reais. A seleção buscou diversidade em termos de volume variando de 214 a 30.000 amostras, dimensionalidade e complexidade topológica.

Primeiramente, foram utilizados 7 conjuntos de dados sintéticos bidimensionais como Spiral, Fullmoon e Halfkernel. O objetivo destes conjuntos é validar visualmente a capacidade do algoritmo em delinear fronteiras de decisão não convexas e recuperar estruturas geométricas complexas em ambiente controlado.

Complementarmente, foram selecionados 7 conjuntos de dados reais do repositório UCI Machine Learning Repository. Estes dados introduzem desafios de dimensionalidade, ruído inerente e classes desbalanceadas, permitindo testar os limites das métricas de separação e compactação em cenários práticos. A Tabela 2 resume as características dos conjuntos de dados utilizados.

Tabela 2 – Resumo das características dos datasets utilizados

Dataset	**Instâncias** (n)	**Atributos** (d)
Synth Halfkernel	500	2
Synth Cluster	506	2
Synth Corners	504	2
Synth Spiral Gaussian	1.606	2
Synth Base Duas Luas	400	2
Synth Fullmoon	1.000	2
Synth Spiral	500	2
UCI Statlog Shuttle	30.000	7
UCI Glass	214	9
UCI Magic Gamma Telescope	19.020	10
UCI HTRU2 Pulsar	17.898	8
UCI Dry Bean	13.611	16
UCI Credit Card Clients	30.000	23
UCI Spambase	4.601	57

4.4 Implementação do Classificador

- Versão Sequencial: Implementação baseada em CPU utilizando uma abordagem iterativa convencional. O processamento é realizado de forma linear, onde um único fluxo de execução percorre todos os pares de pontos para o cálculo de distâncias e verificação da condição geométrica de Gabriel.

- Versão Paralela: Esta implementação mapeia o problema diretamente para a arquitetura massivamente paralela da GPU. Na construção do Grafo de Gabriel, adota-se a estratégia de "uma *thread* por par", onde cada unidade de processamento da GPU é responsável por validar a inequação de Gabriel para uma única aresta candidata. Nas etapas subsequentes, a granularidade é mantida ao nível de ponto, garantindo que cada amostra seja processada de forma independente.

- Versão Paralela Otimizada (*Thread Coarsening*): Diferente da abordagem ingênua, esta versão aplica a técnica de *Thread Coarsening* (Grosseamento de *Threads*) para otimizar a vazão de dados (*throughput*). O agrupamento de trabalho é implementado tanto na etapa de verificação da inequação quanto na etapa de extração de pesos.

  Nesta configuração, cada *thread* é responsável por iterar sobre um lote contínuo de arestas, definido por um parâmetro. Esta estratégia tem o objetivo de reduzir a sobrecarga de escalonamento do *grid* de execução e melhorar a localidade de dados, permitindo que a GPU processe múltiplos pares de pontos antes de ceder os recursos de execução, resultando em um uso mais eficiente.

4.5 Estratégias de Paralelização com CUDA

A implementação do classificador proposto utiliza a arquitetura CUDA para acelerar as etapas computacionalmente intensivas do algoritmo. A estratégia de paralelização foi desenhada para explorar o modelo SIMT, dividindo o processamento entre a CPU e a GPU. A abordagem de paralelização não é uniforme durante toda a execução e se adapta conforme a natureza dos dados em cada etapa do fluxo de processamento: construção do grafo, filtragem de ruído e classificação. As subseções a seguir detalham essas estratégias.

4.5.1 Gerenciamento de Memória e Estruturas de Dados

Devido à complexidade quadrática ($O(N^3)$) da construção do Grafo de Gabriel, o gerenciamento de memória torna-se um fator crítico. A implementação utiliza a biblioteca *Thrust* para o gerenciamento de vetores na GPU (`thrust::device_vector`), garantindo alocação segura e eficiente.

Uma estratégia agressiva de liberação de memória foi adotada para evitar o estouro da VRAM (*Video RAM*). Conforme observado no algoritmo, vetores que armazenam arestas temporárias são explicitamente limpos e compactados (`shrink_to_fit`) imediatamente após a etapa de filtragem de ruído. Isso permite que a GPU processe conjuntos de dados maiores, reaproveitando a memória liberada para as etapas subsequentes de classificação.

4.5.2 Paralelização na Construção do Grafo

A etapa de construção do Grafo de Gabriel exige a verificação da condição de adjacência para cada par possível de pontos no conjunto de treinamento. O número total de pares é dado por:

$$Total_{pares} = \frac{N \times (N-1)}{2} \tag{4.1}$$

Para esta etapa, a paralelização foi realizada com granularidade fina sobre as arestas candidatas. Em vez de atribuir uma *thread* por ponto, o algoritmo calcula o número total de pares e lança um *Grid* unidimensional de blocos. Cada *thread* na GPU é responsável por calcular a distância Euclidiana e a condição de Gabriel para um ou mais pares de pontos. Para garantir a robustez contra grandes volumes de dados, o cálculo do número de blocos utiliza variáveis de 64 bits (`long long`), prevenindo *overflow* de inteiros ao dimensionar o *Grid* de execução.

4.5.3 Otimização na Construção do Grafo e Mapeamento de Índices

A construção do Grafo de Gabriel apresenta um desafio de complexidade. Logo, se uma matriz de adjacência completa fosse alocada para um conjunto de dados massivo,

a capacidade da memória global da GPU seria rapidamente excedida. Para solucionar este problema, foi implementada uma estratégia de Mapeamento Implícito de Índices combinada com uma abordagem de escrita em duas etapas (*Two-Pass Approach*).

4.5.3.1 Mapeamento Implícito ($k \to i, j$)

Para evitar o custo de memória $O(N^2)$ associado ao armazenamento explícito de uma matriz de adjacência completa, o espaço de busca das arestas é linearizado em um vetor virtual contendo todos os pares não ordenados de pontos. Considerando apenas a parte triangular superior estrita da matriz ($i < j$), o número total de pares possíveis é representada pela Equação 4.1.

Cada ponto de execução recebe um índice linear global $k \in [0, Total_{pares} - 1]$ e deve determinar o par de índices (i, j) correspondente na matriz, sem utilizar estruturas auxiliares de memória. Para isso, assume-se a enumeração sequencial das arestas linha a linha na parte triangular superior, conforme ilustrado a seguir:

$$(0,1),(0,2),\ldots,(0,N-1),(1,2),(1,3),\ldots,(1,N-1),\ldots,(N-2,N-1).$$

Observa-se que a linha i contém

$$L_i = N - i - 1 \tag{4.2}$$

pares válidos. Assim, o número total de pares enumerados antes do início da linha i é dado por

$$P(i) = \sum_{r=0}^{i-1} (N - r - 1). \tag{4.3}$$

Essa soma corresponde à diferença entre dois números triangulares:

$$P(i) = \frac{N(N-1)}{2} - \frac{(N-i)(N-i-1)}{2}. \tag{4.4}$$

Portanto, o índice k pertence à linha i que satisfaz

$$P(i) \leq k < P(i+1). \tag{4.5}$$

Para evitar uma busca iterativa, resolve-se analiticamente essa relação para i. A inversão da soma resulta na seguinte expressão fechada:

$$i = N - 2 - \left\lfloor \frac{\sqrt{-8k + 4N(N-1) - 7}}{2} - 0.5 \right\rfloor. \tag{4.6}$$

Uma vez determinado o índice da linha i, o deslocamento de k dentro dessa linha é

$$\Delta = k - P(i). \tag{4.7}$$

Como os índices de coluna começam em $j = i+1$, o valor de j é obtido diretamente por

$$j = i + 1 + \Delta, \tag{4.8}$$

ou, substituindo $P(i)$ explicitamente,

$$j = k - \left(\frac{N(N-1)}{2} - \frac{(N-i)(N-i-1)}{2} \right) + i + 1. \tag{4.9}$$

Dessa forma, cada *thread* pode converter seu índice linear k no par (i, j) correspondente em tempo constante $O(1)$ e utilizando apenas memória constante $O(1)$, o que torna a abordagem adequada para o processamento paralelo em GPUs com restrições de memória.

4.5.3.1.1 Exemplo ilustrativo

Considere $N = 5$. Nesse caso, o número total de pares é

$$Total_{pares} = \frac{5 \cdot 4}{2} = 10. \tag{4.10}$$

A enumeração linear das arestas na parte triangular superior da matriz ocorre da seguinte forma:

	0	1	2	3	4
0	–	$(0,1)_0$	$(0,2)_1$	$(0,3)_2$	$(0,4)_3$
1		–	$(1,2)_4$	$(1,3)_5$	$(1,4)_6$
2			–	$(2,3)_7$	$(2,4)_8$
3				–	$(3,4)_9$
4					–

O subscrito indica o índice linear k associado a cada par (i, j).

Por exemplo, considere $k = 5$. Aplicando a Equação 4.6:

$$i = 5 - 2 - \left\lfloor \frac{\sqrt{-8(5) + 4 \cdot 5 \cdot 4 - 7}}{2} - 0.5 \right\rfloor \tag{4.11}$$

$$i = 3 - \lfloor 2.0 - 0.5 \rfloor = 3 - 1 = 1. \tag{4.12}$$

Em seguida calcula-se

$$P(i) = \frac{5 \cdot 4}{2} - \frac{(5-1)(5-2)}{2} = 10 - 6 = 4. \tag{4.13}$$

Logo,

$$j = i + 1 + (k - P(i)) = 1 + 1 + (5 - 4) = 3. \tag{4.14}$$

Portanto, o índice linear $k = 5$ corresponde ao par $(1, 3)$, conforme mostrado na tabela.

4.5.3.2 Compactação e Uso de Operações Atômicas

Para evitar "buracos"na memória e garantir que o vetor final de arestas seja denso, utilizou-se uma abordagem de duas passadas com otimização via *Bitmask*:

1. Passada 1 (Marcação): O kernel verifica a condição de Gabriel. Se a aresta existe, utiliza-se `atomicOr` para marcar um único *bit* em um vetor de inteiros, reduzindo o consumo de memória de flags em 32 vezes comparado ao uso de `bool` ou `int`. Simultaneamente, um contador global é incrementado via `atomicAdd` para determinar o tamanho exato da alocação necessária.

2. Alocação Intermediária: A CPU lê o contador, aloca exatamente a memória necessária para as arestas e, condicionalmente, para os pesos, dependendo se a etapa seguinte exige ou não as distâncias.

3. Passada 2 (Escrita): O kernel reprocessa os índices. As *threads* consultam o *bitmap* gerado na primeira passada. Apenas para os bits ativos, a *thread* realiza uma operação atômica para obter a posição de escrita no vetor denso final e armazena a aresta.

Essa estratégia garante que o consumo de memória da GPU seja proporcional ao número de arestas reais do grafo.

4.6 Algoritmo Proposto de Clusterização em GPU

Com base na fundamentação teórica sobre componentes conexos e na estrutura geométrica fornecida pelo Grafo de Gabriel, este trabalho propõe um algoritmo de clusterização inédito. A premissa central é que arestas curtas no Grafo de Gabriel representam fortes conexões locais, enquanto arestas longas tendem a conectar grupos distintos.

O método foi projetado para operar inteiramente na GPU, evitando a latência de transferência de memória entre *host* e *device* após a construção do grafo. O algoritmo divide-se em quatro estágios principais: análise da distribuição de pesos, filtragem de arestas, identificação de componentes conexos e fusão hierárquica.

4.6.1 Análise de Distribuição e Heurística do Cotovelo

A primeira etapa consiste em definir um limiar de corte T_{corte} para distinguir arestas que formam a estrutura interna dos *clusters* daquelas que os conectam. Para evitar a definição manual de parâmetros de densidade, implementou-se uma abordagem automática baseada na análise da curva de pesos das arestas.

O procedimento, executado na GPU, segue os passos:

1. Ordenação: Todas as arestas do Grafo de Gabriel são ordenadas de forma crescente com base em seus pesos, neste caso a distância euclidiana, utilizando o algoritmo `thrust::sort`.

2. Detecção do Cotovelo: Considera-se a curva formada pelos pesos ordenados. Traça-se uma reta imaginária conectando o primeiro ponto de menor peso ao último.

3. Cálculo de Distância: Um *kernel* CUDA paralelo calcula, para cada ponto da curva, a distância perpendicular até essa reta. O ponto que apresentar a maior distância é identificado como o "cotovelo"ou ponto de inflexão da distribuição.

4.6.2 Construção do Grafo Residual

Uma vez definido o limiar T_{corte}, aplica-se uma filtragem paralela sobre o conjunto de arestas. Utilizando a técnica de *Stream Compaction* via `thrust::copy_if` e iteradores zip, o algoritmo gera um Grafo Residual.

Neste subgrafo, apenas as arestas cujo peso $w \leq T_{corte} + \epsilon$ são mantidas Figura 6. As arestas longas são descartadas logicamente para fins de conectividade inicial, embora sejam preservadas na memória para a etapa final de fusão. O resultado é um grafo desconexo, onde cada subgrafo isolado representa um "núcleo"de um *cluster* natural.

4.6.3 Identificação de Componentes Conexos (Union-Find Paralelo)

Para identificar os *clusters* no grafo residual, implementou-se uma versão paralela do algoritmo *Union-Find* (ou *Disjoint Set Union - DSU*). Diferente da abordagem sequencial clássica, a versão em GPU lida com a concorrência de milhares de threads tentando unir vértices simultaneamente.

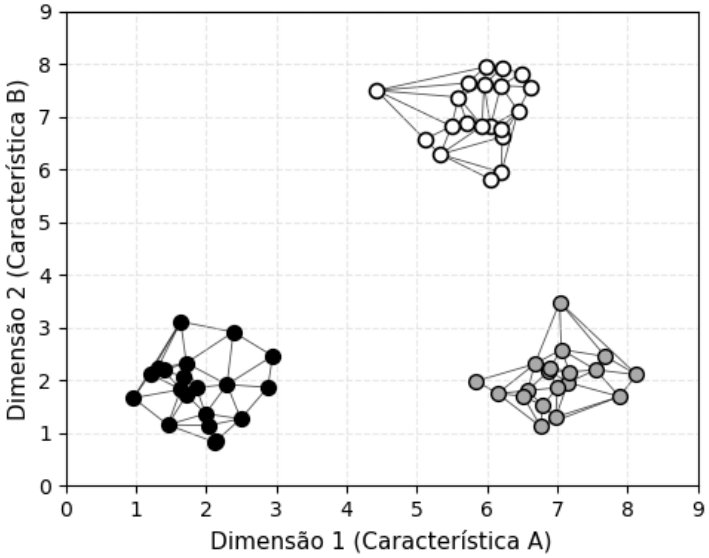

Figura 6 – Representação do grafo residual após a remoção de arestas.

A estrutura de dados é um vetor de rótulos (L), onde inicialmente $L[i] = i$, indicando que cada ponto é seu próprio pai. O processo ocorre iterativamente em dois *kernels* até a convergência:

1. Etapa de Hook (Gancho): Cada *thread* processa uma aresta (u, v) do grafo residual. Se os vértices pertencem a árvores diferentes, a *thread* tenta uní-los. Para evitar condições de corrida, utiliza-se a operação atômica `atomicMin`, garantindo que o vértice com o maior ID se torne filho do vértice com o menor ID. Isso resolve conflitos de escrita de forma determinística.

2. Etapa de Compress (Compressão de Caminho): Um segundo *kernel* percorre todos os vértices e atualiza seus rótulos para apontar diretamente para a raiz da árvore. Isso achata a estrutura, acelerando as iterações subsequentes.

O ciclo se repete até que nenhuma alteração nos rótulos seja detectada. Ao final, todos os pontos pertencentes ao mesmo componente conexo compartilham o mesmo rótulo raiz, definindo os *clusters* iniciais.

4.6.4 Fusão Hierárquica e Controle de Convergência

A segmentação baseada apenas no corte por distância pode resultar em um super-particionamento, especialmente se a densidade dos dados for heterogênea. Para solucionar isso e permitir que o usuário defina um número alvo de grupos (K), implementou-se uma etapa de fusão hierárquica aglomerativa na GPU.

O algoritmo conta o número atual de *clusters* únicos. Enquanto esse número for maior que o K desejado, o processo executa:

1. Busca da Melhor Aresta: Utiliza-se uma redução paralela (`thrust::min_element`) com um comparador customizado. Este avalia todas as arestas originais, mesmo as que foram filtradas anteriormente, e seleciona a aresta que possui o menor peso entre todas aquelas que conectam dois clusters distintos ($L[u] \neq L[v]$).

2. Fusão e Atualização: Identificada a melhor aresta de conexão, os dois *clusters* envolvidos são fundidos. Um *kernel* de re-rotulagem (`relabel_kernel`) é lançado para atualizar massivamente todos os pontos pertencentes ao *cluster* absorvido, atribuindo-lhes o novo rótulo.

Este processo garante que a fusão ocorra sempre pelo caminho de "menor resistência", preservando a coesão local enquanto atinge o número de *clusters* desejado K. Caso não existam mais arestas conectando componentes distintos antes de atingir K, o algoritmo encerra e reporta o número natural de grupos encontrados.

4.7 Métricas e Metodologia de Avaliação de Desempenho em Classificadores

Para avaliar o ganho de desempenho obtido com a paralelização em GPU, foram definidas métricas quantitativas e um procedimento experimental rigoroso. O objetivo desta avaliação é comparar objetivamente a performance da implementação paralela com a sua contraparte sequencial, que serve como baseline.

As seguintes métricas foram utilizadas para a análise comparativa:

- Tempo de Execução: Tempo total, medido em milissegundos (ms), para a execução completa do algoritmo, desde a leitura dos dados até a finalização do processamento.
- Consumo de Energia: estimativa da energia computacional da GPU utilizada durante a execução;
- Acurácia: proporção de previsões corretas em relação ao total de amostras;
- Memória: quantidade de memória alocada durante o processo;
- F1-Score: média harmônica entre precisão e recall.

5 Resultados

5.1 Paralelo vs. Sequencial

Tabela 3 – Comparação de tempo de execução, speedup, acurácia e F1-score entre as versões sequencial e paralela do NN-clas

Dataset	Tam.	Carac.	Seq. (s)	Paral. (s)	Speedup	Acc	F1-score
Australian	690	14	10.3789	0.0026	3933.76×	0.8487	0.8250
BankNote	1372	4	15.1423	0.0022	6882.50×	0.9978	0.9970
Breast Cancer	683	9	4.4336	0.0019	2333.84×	0.9648	0.9730
BreastHess	133	30	0.4752	0.0009	558.60×	0.7879	0.8510
Bupa	345	6	0.5933	0.0008	742.08×	0.5916	0.5360
Climate	540	18	23.4760	0.0064	3665.63×	0.8870	0.9380
Diabetes	768	8	5.5789	0.0021	2656.59×	0.7474	0.7980
German	1000	24	93.3308	0.0110	8484.62×	0.7120	0.7780
Golub	72	50	0.1709	0.0007	237.43×	0.7732	0.8280
Haberman	306	3	0.3409	0.0006	532.58×	0.6431	0.7520
Heart	270	13	1.1135	0.0010	1080.06×	0.8222	0.8360
ILPD	579	10	2.3315	0.0012	1942.64×	0.6028	0.6960
Parkinsons	195	22	0.5699	0.0008	759.91×	0.8045	0.8740
Sonar	208	60	3.4717	0.0014	2548.31×	0.7355	0.6870

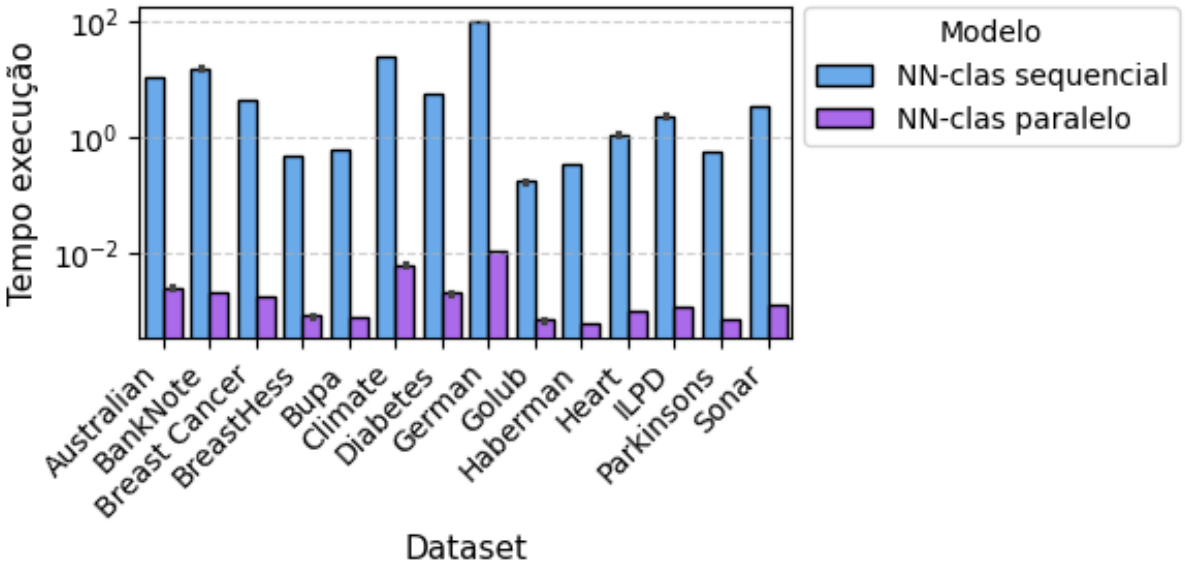

Figura 7 – Comparação no tempo de execução entre as implementações paralelas e sequenciais.

Os resultados apresentados na Tabela 3 evidenciam a eficiência da implementação paralela sobre a versão sequencial. Observa-se que, para todos os conjuntos de dados

testados, a versão acelerada por GPU obteve um desempenho ordens de magnitude superior, reduzindo tempos de execução que variavam de segundos (ou até minutos) para a escala de milissegundos.

O *speedup* obtido varia de aproximadamente 237× (*dataset* Golub) a 8.484× (*dataset* German). Esta variação demonstra correlação direta entre a complexidade computacional dos conjuntos de dados e o ganho de desempenho proporcionado pela GPU. Em datasets menores, como o Golub (72 instâncias), o *overhead* de transferência de memória entre *Host* e *Device* e a inicialização dos *kernels* têm um peso proporcionalmente maior, limitando o *speedup*, embora este ainda permaneça na casa das centenas.

É válido ressaltar que a redução no tempo de execução viabiliza a utilização do classificador proposto em cenários de tempo real, algo inviável com a implementação sequencial para bases de dados dessa magnitude. Por fim, as colunas de Acurácia (Acc) e F1-Score confirmam que a paralelização manteve a integridade lógica do algoritmo. Os valores de acurácia, atingindo até $99,78\%$ no caso do *BankNote*, e F1-Score consistentes indicam que as otimizações de memória e o uso de operações atômicas não introduziram erros de precisão numérica significativos na classificação.

5.2 Múltiplos Pares por Thread

A fim de otimizar o desempenho da solução paralela, avaliou-se o impacto da granularidade das tarefas, especificamente a quantidade de pares de pontos atribuídos a cada *thread*. A Figura 8 ilustra como o tempo de execução varia no *dataset Diabetes*, enquanto a Figura 9 apresenta o mesmo comportamento no *dataset Firewall*.

Observa-se que, quando a granularidade é muito fina, poucas comparações por thread, o tempo de execução aumenta significativamente. Nessa configuração, o custo de criação, sincronização e escalonamento das threads passa a representar uma fração desproporcional do tempo total. Assim, o *overhead* do paralelismo supera os ganhos proporcionados pela divisão do trabalho.

À medida que se aumenta a quantidade de pares por thread, o tempo de execução diminui. Esse comportamento é esperado, pois granularidades maiores:

- Amortizam o overhead do paralelismo, diluindo custos de criação e coordenação das threads ao longo de blocos maiores de computação útil;
- Aprimoram a localidade espacial e temporal, já que threads passam a operar em regiões mais coesas dos dados, reduzindo falhas de cache e aumentando a eficiência de memória.

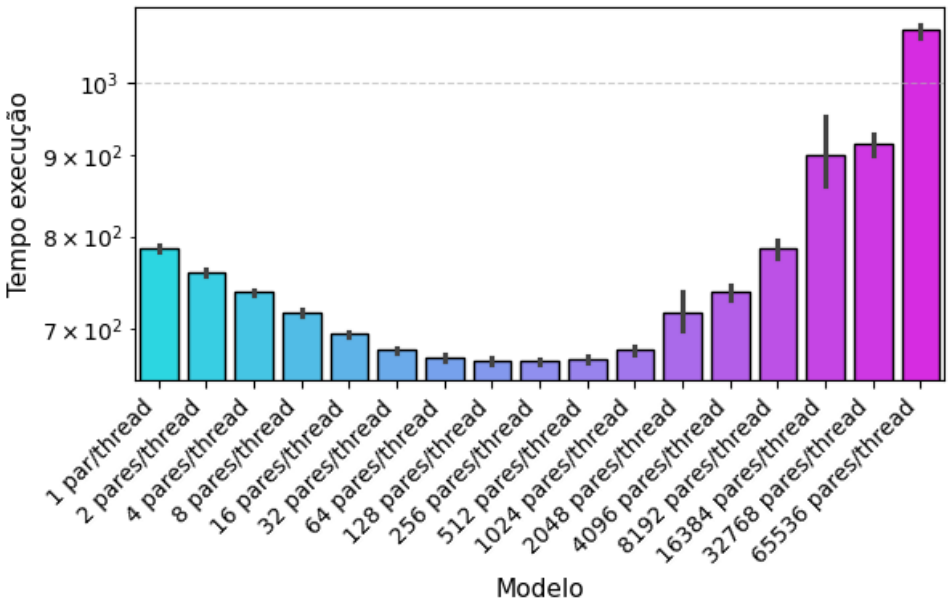

Figura 8 – Progressão de tempo ao variar a quantidade de pares por threads no dataset Diabetes.

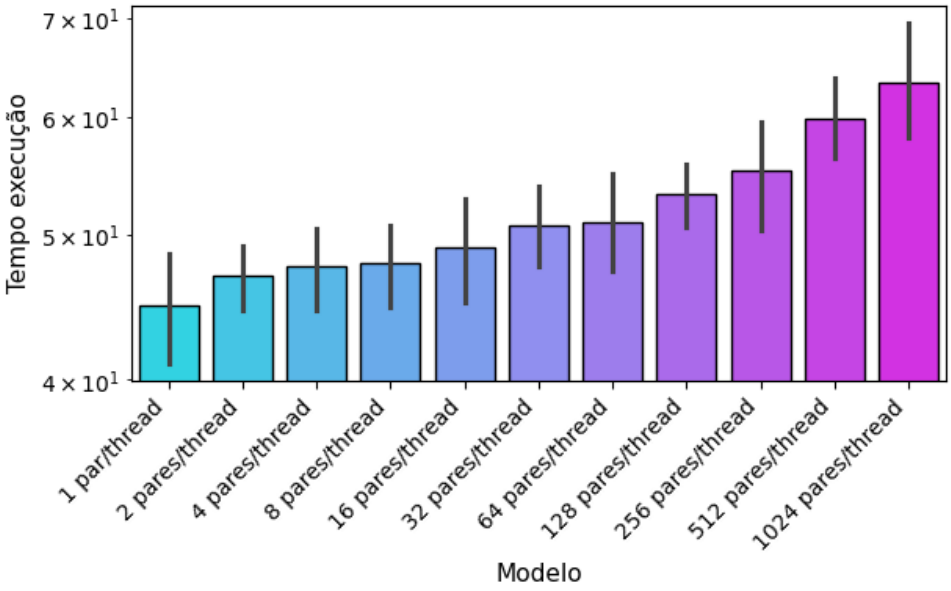

Figura 9 – Progressão de tempo ao variar a quantidade de pares por threads no dataset Firewall.

Entretanto, o aumento da granularidade não pode ser ilimitado. Após certo ponto, o desempenho volta a cair. Isso pode estar relacionado ao desbalanceamento de carga. Algumas *threads* recebem blocos maiores que outras, fazendo com que parte dos núcleos de processamento fique ociosa enquanto aguardam a conclusão das *threads* mais lentas. Esse efeito é mais evidente nas configurações com milhares de pares por *thread*, conforme indicado pelo crescimento abrupto nos gráficos.

O comportamento observado nas duas figuras aponta para a existência de um

ponto ótimo, no qual o custo de coordenação é suficientemente pequeno e a distribuição de carga ainda é equilibrada. Tal ponto de inflexão representa a configuração mais eficiente para o ambiente de testes considerado, refletindo uma relação adequada entre paralelismo, *overhead* e eficiência de memória.

5.3 Algoritmo de Clusterização

A avaliação do algoritmo de clusterização proposto foi dividida em duas etapas distintas para validar diferentes aspectos da metodologia. Na primeira etapa, avalia-se a capacidade do algoritmo em detectar automaticamente a estrutura dos dados utilizando a heurística do cotovelo para o corte de arestas. Na segunda etapa, avalia-se o impacto da etapa de Fusão Hierárquica, onde o algoritmo é forçado a convergir para um número de clusters K, permitindo uma comparação direta com o *Ground Truth* dos *datasets*.

Os resultados foram comparados com dois algoritmos clássicos da literatura: o K-Means e o DBSCAN. A análise baseou-se nos rankings médios obtidos em 14 conjuntos de dados.

5.3.1 Análise da Abordagem Automática

Nesta configuração, o algoritmo proposto define o número de *clusters* se baseando exclusivamente na geometria do Grafo de Gabriel e no limiar de corte T_{corte} identificado automaticamente.

5.3.1.1 Análise das Métricas de Coesão e Separação

As Tabelas 4, 5 e 6 apresentam, respectivamente, os resultados para o Coeficiente de Silhouette, Índice Davies-Bouldin e Índice Calinski-Harabasz. Observa-se que o K-Means obteve os melhores postos (Rank Médio entre 1.17 e 1.32) nestas três métricas. Este resultado corrobora o viés convexo destas métricas, que favorecem agrupamentos esféricos.

Por outro lado, na Tabela 7, referente ao Índice de Dunn (DI), o método proposto e o DBSCAN superaram significativamente o K-Means. O Índice de Dunn valoriza a separação topológica real, confirmando que a abordagem baseada em grafos é superior na detecção de densidades complexas e não-convexas, mesmo sem informar o número de *clusters*.

5.3.1.2 Análise Estatística e Interpretação das Métricas do K Automático

A Figura 10 (pág. 52) resume a validação estatística rigorosa realizada sobre os resultados experimentais. Inicialmente, o teste não-paramétrico de Friedman foi aplicado para verificar a existência de diferenças globais entre os algoritmos. O teste confirmou

Tabela 4 – Comparação do Coeficiente de Silhouette (↑) - Automático. Valores entre parênteses indicam o posto (rank).

Dataset	**K-Means**	**DBSCAN**	**Proposto**
Synth Halfkernel	0.472 (1)	0.113 (3)	0.221 (2)
Synth Cluster	0.650 (1)	0.222 (2.5)	0.222 (2.5)
Synth Corners	0.526 (1)	0.451 (2.5)	0.451 (2.5)
Synth Spiral Gaussian	0.501 (1.5)	0.501 (1.5)	0.439 (3)
Synth Base Duas Luas	0.516 (1)	0.291 (2)	0.100 (3)
Synth Fullmoon	0.452 (1)	0.228 (2)	0.148 (3)
Synth Spiral	0.443 (1)	0.376 (2)	-0.099 (3)
UCI Statlog Shuttle	0.399 (1)	-0.096 (3)	0.079 (2)
UCI Glass	0.458 (1)	0.255 (2)	0.074 (3)
UCI Magic Gamma	0.243 (1)	-0.339 (3)	0.075 (2)
UCI HTRU2 Pulsar	0.594 (1)	-0.436 (3)	0.077 (2)
UCI Dry Bean	0.284 (1)	-0.850 (3)	-0.417 (2)
UCI Credit Card	0.255 (1)	-0.349 (3)	-0.276 (2)
UCI Spambase	0.284 (1)	-0.419 (3)	-0.174 (2)
Average Rank	**1.18**	2.46	2.36

que existe diferença estatisticamente significativa entre os métodos avaliados em todos os cenários ($p < 0.05$), rejeitando a hipótese nula de equivalência de desempenho.

Para identificar quais pares de algoritmos diferem entre si, foi aplicado o teste pós-hoc de Nemenyi. A análise detalhada de cada índice permite compreender as forças e fraquezas do método proposto em relação ao estado da arte:

- Coeficiente de Silhouette: O teste de Nemenyi indicou a superioridade estatística do algoritmo K-Means em relação ao algoritmo proposto e ao DBSCAN. Este resultado deve-se ao viés convexo inerente a esta métrica, que utiliza distâncias euclidianas diretas para medir coesão e separação. Como o K-Means é otimizado para minimizar a variância em *clusters* esféricos, ele é naturalmente favorecido, enquanto métodos que detectam formas complexas são penalizados ao identificar estruturas não convexas, mesmo que topologicamente corretas.

- Índice Davies-Bouldin (DB): Nesta métrica, onde menores valores indicam melhor qualidade, o K-Means obteve o melhor posto médio. Contudo, o teste de Nemenyi revelou que apenas o DBSCAN foi estatisticamente inferior ao grupo de controle. O algoritmo proposto se posicionou de forma intermediária, demonstrando que, embora

Tabela 5 – Comparação do Índice Davies-Bouldin (↓) - Automático. Valores entre parênteses indicam o posto (rank).

Dataset	**K-Means**	**DBSCAN**	**Proposto**
Synth Halfkernel	0.525 (1)	1.366 (3)	0.892 (2)
Synth Cluster	0.539 (1)	1.717 (2.5)	1.717 (2.5)
Synth Corners	0.486 (1)	1.996 (2.5)	1.996 (2.5)
Synth Spiral Gaussian	0.637 (1.5)	0.637 (1.5)	0.773 (3)
Synth Base Duas Luas	0.800 (2)	1.554 (3)	0.600 (1)
Synth Fullmoon	0.400 (2)	0.740 (3)	0.163 (1)
Synth Spiral	0.016 (1)	0.201 (2)	0.478 (3)
UCI Statlog Shuttle	0.448 (1)	1.537 (3)	0.928 (2)
UCI Glass	0.830 (1)	1.358 (2)	1.644 (3)
UCI Magic Gamma	1.164 (1)	5.064 (2)	6.489 (3)
UCI HTRU2 Pulsar	0.496 (1)	1.957 (2)	4.674 (3)
UCI Dry Bean	0.818 (1)	9.334 (3)	1.541 (2)
UCI Credit Card	0.786 (1)	9.055 (3)	1.101 (2)
UCI Spambase	0.753 (2)	0.342 (1)	0.505 (2)
Average Rank	**1.32**	2.68	2.00

permita formas livres, mantém uma compactação interna superior à do DBSCAN, evitando a dispersão excessiva frequentemente penalizada por este índice.

- Índice Calinski-Harabasz (CH): Semelhante ao coeficiente de Silhouette, esta métrica baseada em variância favoreceu fortemente o K-Means, com diferença estatisticamente significativa confirmada pelo teste de Nemenyi. O índice penaliza clusters alongados ou de densidade variável, o que explica a pontuação inferior tanto do algoritmo proposto quanto do DBSCAN, confirmando que ambos operam sob uma lógica de densidade e não de geometria euclidiana estrita.

- Índice de Dunn (DI): Este índice representa o ponto de virada na análise, pois valoriza a separação topológica real entre densidades. Os testes estatísticos indicaram um empate técnico entre o algoritmo proposto e o DBSCAN, sendo ambos significativamente superiores ao K-Means. Este resultado valida a proposta do trabalho, comprovando estatisticamente que o método baseado no Grafo de Gabriel é tão robusto quanto o estado da arte na detecção da estrutura natural dos dados, superando abordagens clássicas em cenários de topologia complexa.

- Índice Xie-Beni (XB): O algoritmo proposto apresentou desempenho inferior nesta

Tabela 6 – Comparação do Índice Calinski-Harabasz (↑) - Automático. Valores entre parênteses indicam o posto (rank).

Dataset	**K-Means**	**DBSCAN**	**Proposto**
Synth Halfkernel	747.25 (1)	41.51 (2)	21.68 (3)
Synth Cluster	426.44 (1)	24.28 (2.5)	24.28 (2.5)
Synth Corners	748.40 (1)	293.94 (2.5)	293.94 (2.5)
Synth Spiral Gaussian	4996.24 (1.5)	4996.24 (1.5)	1425.86 (3)
Synth Base Duas Luas	596.38 (1)	123.17 (2)	109.74 (3)
Synth Fullmoon	1475.99 (1)	278.66 (2)	102.61 (3)
Synth Spiral	1436.07 (1)	608.67 (2)	30.91 (3)
UCI Statlog Shuttle	5718.84 (1)	586.21 (3)	1131.68 (2)
UCI Glass	120.00 (1)	53.30 (3)	100.69 (2)
UCI Magic Gamma	7868.18 (1)	8.57 (3)	318.20 (2)
UCI HTRU2 Pulsar	1694.99 (1)	4.48 (3)	2167.26 (2)
UCI Dry Bean	2786.04 (1)	6.53 (3)	1045.14 (2)
UCI Credit Card	1955.36 (1)	1.82 (3)	612.71 (2)
UCI Spambase	451.99 (3)	847.40 (1)	638.04 (2)
Average Rank	**1.17**	2.42	2.57

métrica. A explicação reside na formulação matemática do índice, que penaliza *clusters* cujos centróides geométricos estão próximos, mesmo que os grupos não se toquem. Isso prejudica a avaliação do algoritmo em datasets onde ele resolve corretamente a separação topológica, mas mantém proximidade espacial entre os núcleos dos *clusters*.

5.3.1.3 Análise de Robustez: Teste Bonferroni-Dunn

Para refinar a comparação e verificar se as diferenças de desempenho em relação ao algoritmo de melhor ranking são estatisticamente significativas, aplicou-se o teste pós-hoc de Bonferroni-Dunn, com um nível de significância $\alpha = 0.05$ e Diferença Crítica (CD) de 0.8472, Figura 11 (pág. 54).

Os resultados desta análise, detalhados a seguir, reforçam a competitividade da abordagem proposta:

- Equivalência Estatística no Índice Davies-Bouldin: Embora o ranking médio do K-Means (1.32) tenha sido superior ao do algoritmo proposto (2.00), a diferença entre eles (0.679) é inferior ao limiar crítico ($CD = 0.8472$). Isso implica que, sob a ótica estatística rigorosa do teste de Bonferroni-Dunn, o algoritmo proposto

Tabela 7 – Comparação do Índice de Dunn (↑) - Automático. Valores entre parênteses indicam o posto (rank).

Dataset	**K-Means**	**DBSCAN**	**Proposto**
Synth Halfkernel	0.021 (3)	0.097 (1)	0.091 (2)
Synth Cluster	0.012 (3)	0.030 (1.5)	0.030 (1.5)
Synth Corners	0.033 (1)	0.029 (2)	0.017 (3)
Synth Spiral Gaussian	0.082 (3)	0.412 (1)	0.292 (2)
Synth Base Duas Luas	0.030 (3)	0.115 (2)	0.434 (1)
Synth Fullmoon	0.017 (3)	0.182 (2)	0.489 (1)
Synth Spiral	0.143 (2)	32.35 (1)	6.12 (3)
UCI Statlog Shuttle	0.159 (1)	0.051 (2)	0.017 (3)
UCI Glass	0.163 (2)	0.053 (3)	0.235 (1)
UCI Magic Gamma	0.123 (1)	0.085 (2)	0.013 (3)
UCI HTRU2 Pulsar	0.186 (1)	0.044 (3)	0.047 (2)
UCI Dry Bean	0.038 (2)	0.065 (1)	0.010 (3)
UCI Credit Card	0.024 (2)	0.000 (3)	0.117 (1)
UCI Spambase	0.125 (1)	0.053 (2)	0.000 (3)
Average Rank	2.61	**1.61**	1.79

possui desempenho equivalente ao melhor classificador (K-Means) nesta métrica, diferenciando-se significativamente apenas do DBSCAN.

- Consistência no Índice de Dunn: O teste confirmou que o algoritmo proposto é estatisticamente indistinguível do DBSCAN, com uma diferença de apenas 0.179, muito abaixo do limiar de 0.8472. O K-Means, por outro lado, ultrapassou o limiar de diferença crítica, confirmando sua inadequação para as estruturas topológicas avaliadas por este índice.

- Métricas de Forma Convexa e Xie-Beni: Nas métricas Silhouette, Calinski-Harabasz e Xie-Beni, o teste confirmou que a distância entre o algoritmo proposto e o algoritmo de controle (K-Means) excede o CD, reafirmando as diferenças significativas discutidas na análise de Nemenyi.

Em síntese, a análise estatística demonstra que, enquanto o K-Means domina nas métricas de forma convexa, o algoritmo proposto atinge desempenho equivalente ao estado da arte (DBSCAN) nas métricas de densidade e topologia (Índice de Dunn), validando sua eficácia para problemas de clusterização não supervisionada com formatos arbitrários.

Tabela 8 – Comparação do Índice Xie-Beni (↓) - Automático. Valores entre parênteses indicam o posto (rank).

Dataset	**K-Means**	**DBSCAN**	**Proposto**
Synth Halfkernel	10.88 (3)	5.89 (1)	9.91 (2)
Synth Cluster	4.43 (3)	0.87 (1.5)	0.87 (1.5)
Synth Corners	13.98 (3)	1.76 (1.5)	1.76 (1.5)
Synth Spiral Gaussian	41.28 (3)	4.12 (1)	10.72 (2)
Synth Base Duas Luas	16.26 (3)	4.34 (2)	2.48 (1)
Synth Fullmoon	11.63 (3)	6.71 (2)	0.85 (1)
Synth Spiral	91.76 (3)	3.16 (2)	2.62 (1)
UCI Statlog Shuttle	10.25 (2)	5.12 (1)	33.22 (3)
UCI Glass	1.62 (1)	9.17 (2)	22.71 (3)
UCI Magic Gamma	3.89 (1)	1.45e10 (3)	31.09 (2)
UCI HTRU2 Pulsar	1.30 (1)	35.06 (3)	4.36 (2)
UCI Dry Bean	1.81 (2)	0.00 (1)	3.51e10 (3)
UCI Credit Card	18.26 (3)	0.00 (1)	8.95 (2)
UCI Spambase	5.10 (2)	5.30 (3)	5.13e8 (3)
Average Rank	**1.39**	1.82	2.79

5.3.2 Análise com Fusão Hierárquica (K Otimizado)

Nesta segunda fase de testes, ativou-se o módulo de fusão hierárquica aglomerativa na GPU. O objetivo foi avaliar se o super-particionamento, poderia ser mitigado ao fundir componentes conexos até atingir o número de classes original do dataset (K alvo).

As Tabelas 9 a 13 apresentam os resultados comparativos desta abordagem.

5.3.2.1 Impacto nas Métricas de Forma (Silhouette, DB e CH)

Ao observar a Tabela 9, nota-se uma melhoria substancial no desempenho do algoritmo proposto em comparação à versão automática. O Rank Médio do algoritmo melhorou de 2.36 para **2.15**, ultrapassando o DBSCAN (2.58). Isso ocorre porque a fusão hierárquica conecta micro-clusters vizinhos, aumentando a coesão interna dos grupos.

5.3.2.2 Desempenho em Métricas Topológicas (Dunn e Xie-Beni)

O destaque da abordagem com Fusão Hierárquica pode ser observado na Tabela 12, referente ao Índice de Dunn. Nesta configuração, o algoritmo proposto atingiu o Rank

Tabela 9 – Comparação do Coeficiente de Silhouette (↑) - K Otimizado.

Dataset	**K-Means**	**DBSCAN**	**Proposto**
Synth Halfkernel	0.473 (1)	0.151 (3)	0.217 (2)
Synth Cluster	0.505 (1)	0.220 (2.5)	0.220 (2.5)
Synth Corners	0.526 (1)	0.452 (2.5)	0.452 (2.5)
Synth Spiral Gaussian	0.750 (1.5)	0.750 (1.5)	0.397 (3)
Synth Base Duas Luas	0.517 (1)	0.292 (2.5)	0.292 (2.5)
Synth Fullmoon	0.520 (1)	0.285 (2.5)	0.285 (2.5)
Synth Spiral	0.430 (1)	0.077 (2)	0.069 (3)
UCI Statlog Shuttle	0.995 (2)	-0.106 (3)	0.998 (1)
UCI Magic Gamma	0.431 (2)	-0.395 (3)	0.667 (1)
UCI HTRU2 Pulsar	0.595 (2)	-0.437 (3)	0.881 (1)
UCI Dry Bean	0.842 (1)	-0.851 (3)	-0.326 (2)
UCI Credit Card	0.553 (1)	-0.499 (3)	-0.154 (2)
UCI Spambase	0.848 (1)	-0.419 (2)	-0.745 (3)
Average Rank	**1.27**	2.58	2.15

Tabela 10 – Comparação do Índice Davies-Bouldin (↓) - K Otimizado.

Dataset	**K-Means**	**DBSCAN**	**Proposto**
Synth Halfkernel	0.685 (1)	1.900 (2)	2.395 (3)
Synth Cluster	0.642 (1)	94.623 (2.5)	94.623 (2.5)
Synth Corners	0.616 (1)	0.762 (2.5)	0.762 (2.5)
Synth Spiral Gaussian	0.350 (1.5)	0.350 (1.5)	1.426 (3)
Synth Base Duas Luas	0.596 (1)	1.232 (2.5)	1.232 (2.5)
Synth Fullmoon	0.614 (1)	1.210 (2.5)	1.210 (2.5)
Synth Spiral	0.771 (1)	1.435 (2)	3.142 (3)
UCI Statlog Shuttle	0.365 (2)	1.965 (3)	0.002 (1)
UCI Magic Gamma	1.260 (2)	1.507 (3)	0.239 (1)
UCI HTRU2 Pulsar	0.679 (2)	2.159 (3)	0.095 (1)
UCI Dry Bean	0.258 (1)	7.456 (3)	0.958 (2)
UCI Credit Card	0.891 (1)	1.242 (3)	0.988 (2)
UCI Spambase	0.586 (1)	2.081 (3)	1.018 (2)
Average Rank	**1.27**	2.58	2.15

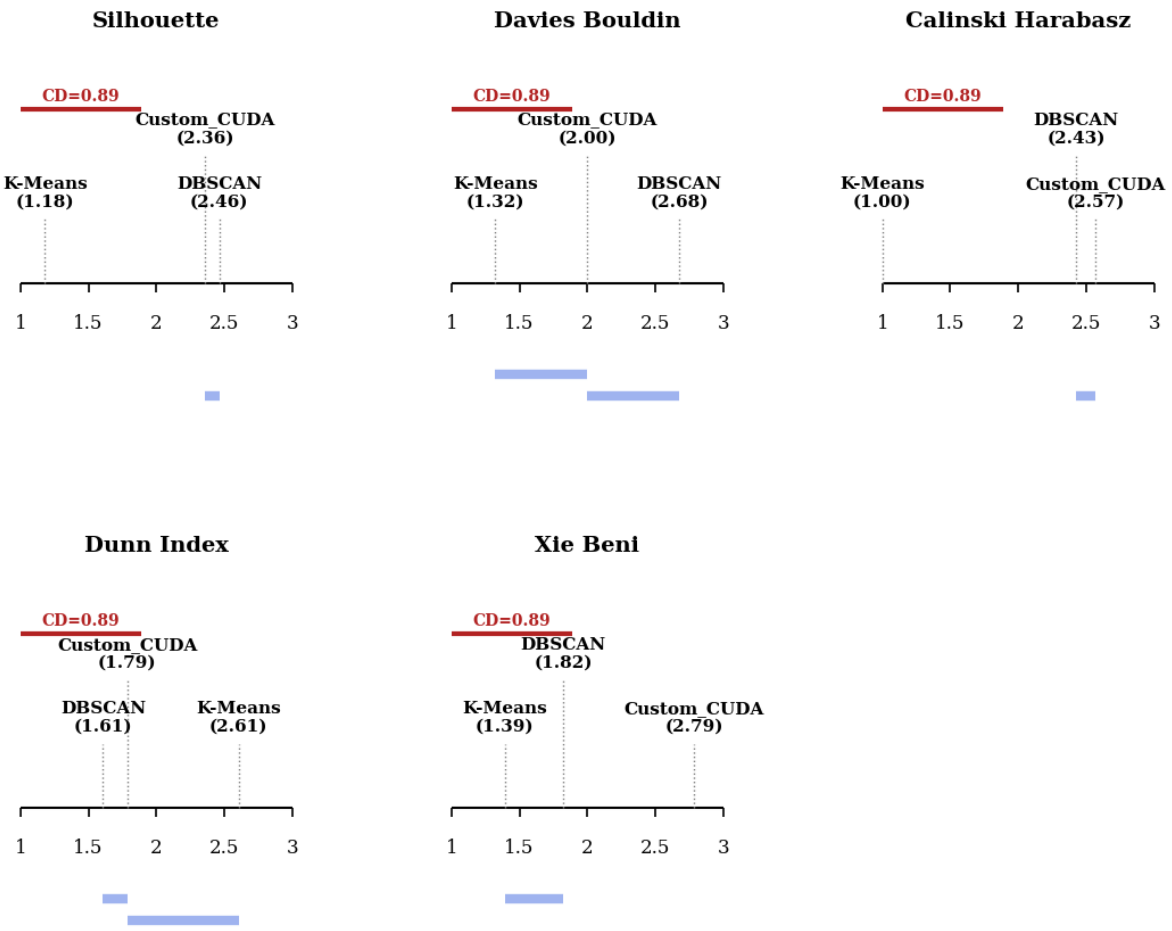

Figura 10 – Representação pós-hoc de Nemenyi para os testes automáticos.

Médio de 1.69, tornando-se o método com melhor desempenho global, superando tanto o DBSCAN (1.73) quanto o K-Means (2.58).

5.3.2.3 Análise Estatística e Interpretação das Métricas (K Otimizado)

A Figura 12 (pág. 56) resume a validação estatística realizada para os resultados com K Otimizado. O teste de Friedman confirmou diferenças globais significativas ($p < 0.05$) para a maioria das métricas, com exceção do índice Xie-Beni.

A aplicação do teste pós-hoc de Nemenyi revelou um cenário mais competitivo para o algoritmo proposto em comparação à etapa automática:

- Coeficiente de Silhouette e Davies-Bouldin: Diferente da etapa automática, onde o K-Means era isoladamente superior, na etapa com K otimizado o teste de Nemenyi indicou que não há diferença estatística significativa entre o K-Means e o algoritmo proposto ($p > 0.05$). Apenas o DBSCAN foi identificado como significativamente inferior ao K-Means. Isso sugere que, ao ajustar o número de clusters, o algoritmo

Tabela 11 – Comparação do Índice Calinski-Harabasz (↑) - K Otimizado.

Dataset	**K-Means**	**DBSCAN**	**Proposto**
Synth Halfkernel	662.54 (1)	125.83 (2)	73.00 (3)
Synth Cluster	426.44 (1)	0.04 (2.5)	0.04 (2.5)
Synth Corners	748.41 (1)	498.27 (2.5)	498.27 (2.5)
Synth Spiral Gaussian	6494.18 (1.5)	6494.18 (1.5)	4327.32 (3)
Synth Base Duas Luas	855.91 (1)	228.49 (2.5)	228.49 (2.5)
Synth Fullmoon	1476.00 (1)	278.66 (2.5)	278.66 (2.5)
Synth Spiral	581.41 (1)	122.61 (2)	47.38 (3)
UCI Statlog Shuttle	63094.34 (1)	0.11 (3)	13505.60 (2)
UCI Magic Gamma	7868.19 (1)	1.19 (3)	12.53 (2)
UCI HTRU2 Pulsar	16949.97 (1)	1.39 (3)	360.86 (2)
UCI Dry Bean	27860.42 (1)	0.14 (3)	0.91 (2)
UCI Credit Card	19553.64 (1)	1.83 (2)	0.69 (3)
UCI Spambase	4519.93 (1)	3.47 (3)	6.38 (2)
Average Rank	**1.04**	2.50	2.46

Tabela 12 – Comparação do Índice de Dunn (↑) - K Otimizado.

Dataset	**K-Means**	**DBSCAN**	**Proposto**
Synth Halfkernel	0.019 (3)	0.059 (2)	0.099 (1)
Synth Cluster	0.063 (3)	0.243 (1.5)	0.243 (1.5)
Synth Corners	0.033 (3)	0.294 (1.5)	0.294 (1.5)
Synth Spiral Gaussian	0.829 (1.5)	0.829 (1.5)	0.003 (3)
Synth Base Duas Luas	0.030 (3)	0.115 (1.5)	0.115 (1.5)
Synth Fullmoon	0.017 (3)	0.183 (1.5)	0.183 (1.5)
Synth Spiral	0.014 (3)	0.061 (2)	0.133 (1)
UCI Statlog Shuttle	0.311 (2)	0.056 (3)	0.634 (1)
UCI Magic Gamma	0.012 (3)	8.574 (1)	0.178 (2)
UCI HTRU2 Pulsar	0.002 (3)	0.045 (2)	0.047 (1)
UCI Dry Bean	0.038 (2)	6539.77 (1)	5.69×10^{-5} (3)
UCI Credit Card	0.002 (2)	0.000 (3)	0.019 (1)
UCI Spambase	0.001 (2)	0.085 (1)	5.13×10^{-7} (3)
Average Rank	2.58	1.73	**1.69**

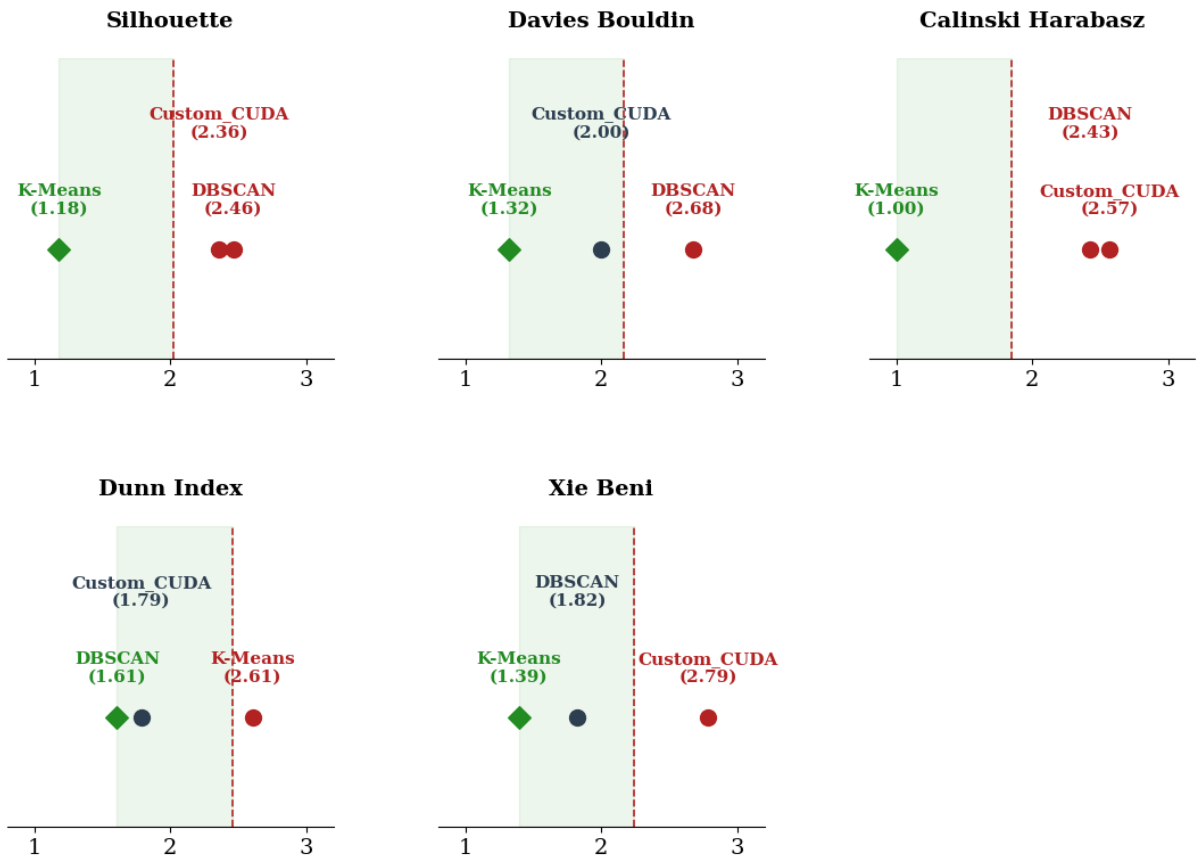

Figura 11 – Representação Bonferroni-Dunn para os testes automáticos.

proposto consegue gerar partições com compactação comparável à do K-Means, mantendo a flexibilidade geométrica.

- Índice Calinski-Harabasz (CH): Nesta métrica, o K-Means manteve sua superioridade estatística em relação a ambos os algoritmos baseados em densidade ($p < 0.001$). O índice CH penaliza fortemente clusters não-esféricos, favorecendo a geometria voronoi do K-Means independente do ajuste de K.

- Índice de Dunn (DI): Confirmando a robustez topológica do método, o algoritmo proposto e o DBSCAN apresentaram desempenho estatisticamente superior ao K-Means ($p < 0.05$). Não houve diferença significativa entre o método proposto e o DBSCAN, indicando que a estratégia de fusão hierárquica preserva a qualidade da separação inter-cluster baseada em densidade.

- Índice Xie-Beni (XB): O teste de Friedman retornou um p-value de 0.113, indicando que não há evidências estatísticas suficientes para rejeitar a hipótese nula. . Assim, não foi observada diferença estatisticamente significativa entre os três algoritmos para essa métrica quando K é fixado.

Tabela 13 – Comparação do Índice Xie-Beni (↓) - K Otimizado.

Dataset	**K-Means**	**DBSCAN**	**Proposto**
Synth Halfkernel	0.205 (1)	1.249 (2)	1.705 (3)
Synth Cluster	0.124 (1)	3058.09 (2.5)	3058.09 (2.5)
Synth Corners	0.140 (1)	0.177 (2.5)	0.177 (2.5)
Synth Spiral Gaussian	0.041 (1.5)	0.041 (1.5)	10.724 (3)
Synth Base Duas Luas	0.163 (1)	0.434 (2.5)	0.434 (2.5)
Synth Fullmoon	0.116 (1)	0.672 (2.5)	0.672 (2.5)
Synth Spiral	0.176 (1)	3.235 (3)	2.628 (2)
UCI Statlog Shuttle	1.06×10^{-4} (2)	2.068 (3)	7.40×10^{-5} (1)
UCI Magic Gamma	0.389 (3)	1.45×10^{-6} (1)	0.080 (2)
UCI HTRU2 Pulsar	0.131 (2)	3.507 (3)	0.011 (1)
UCI Dry Bean	0.018 (2)	0.000 (1)	5.65×10^{5} (3)
UCI Credit Card	0.183 (2)	0.000 (1)	1.455 (3)
UCI Spambase	0.051 (1)	0.530 (2)	4.16×10^{9} (3)
Average Rank	**1.50**	2.11	2.38

5.3.2.4 Análise de Robustez: Teste Bonferroni-Dunn (K Otimizado)

Para validar a competitividade do método proposto contra o algoritmo de melhor ranking em cada cenário (Controle), aplicou-se o teste de Bonferroni-Dunn com Diferença Crítica ($CD = 0.8472$) Figura 13 (pág. 57).

Os resultados demonstram uma evolução clara em relação à abordagem automática:

- Empate Técnico com o Estado da Arte em Métricas Convexas: Nas métricas Silhouette e Davies-Bouldin, onde o K-Means é o controle (Rank 1.32), o algoritmo proposto obteve um Rank de 2.07. A diferença (0.750) é menor que a Diferença Crítica (0.8472). Isso comprova estatisticamente que, com a fusão hierárquica, o algoritmo proposto torna-se equivalente ao K-Means em métricas de coesão, superando o DBSCAN.

- Liderança Compartilhada no Índice de Dunn: Considerando o DBSCAN como controle (Rank 1.68), o algoritmo proposto ficou com Rank 1.71. A baixa diferença (0.036) confirma que o método proposto é tão eficaz quanto o DBSCAN para maximizar a separação topológica. Em contrapartida, o K-Means (Rank 2.61) excedeu o CD, sendo estatisticamente pior que ambos.

- Limitações em Calinski-Harabasz: Apesar das melhorias, no índice CH o algoritmo proposto (Rank 2.43) ainda excede a diferença crítica em relação ao K-Means (Rank 1.00), reafirmando que métricas baseadas estritamente na variância dos centróides favorecem inerentemente a abordagem de partição de Voronoi do K-Means.

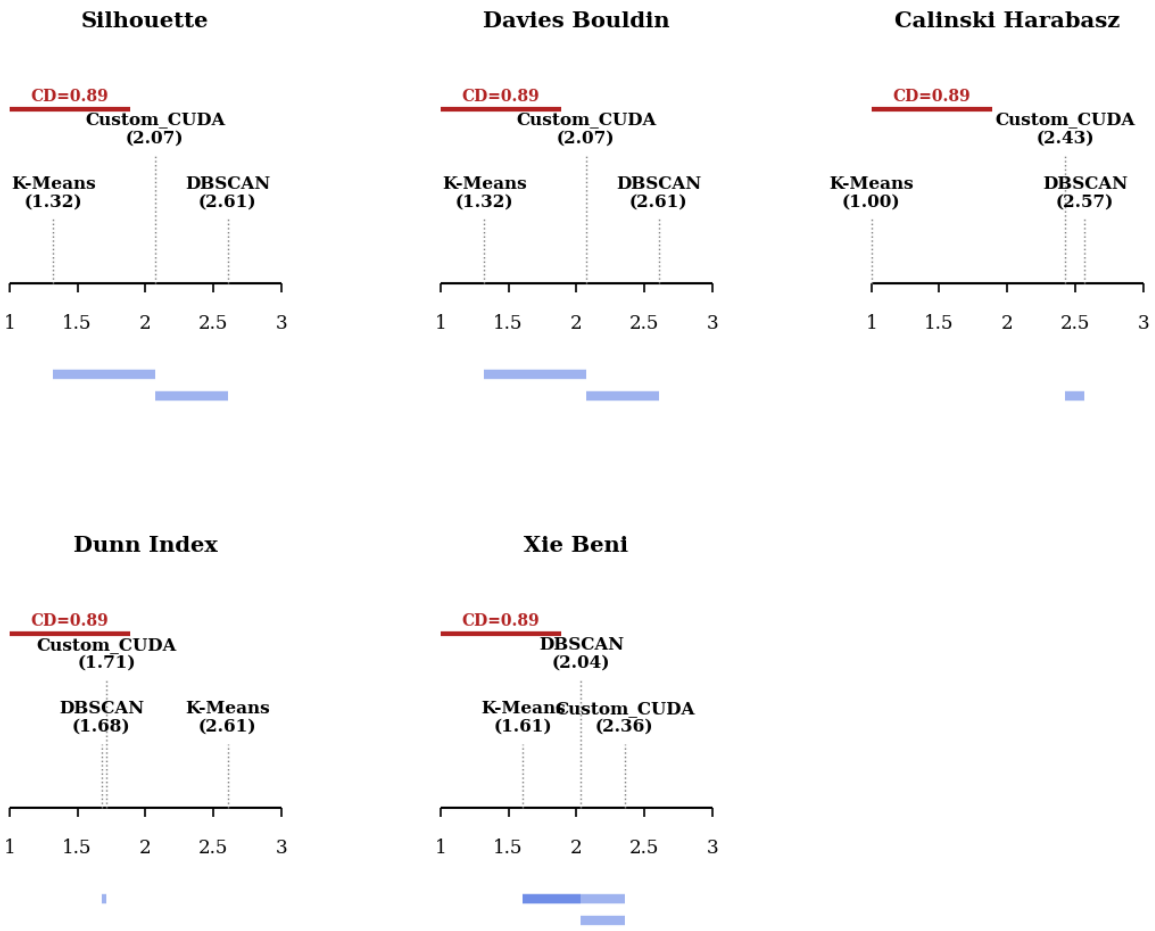

Figura 12 – Representação pós-hoc de Nemenyi K otimizado.

Esta análise conclui que a etapa de Fusão Hierárquica não apenas corrige o super-particionamento, mas eleva o nível de robustez do algoritmo proposto, permitindo que ele compita estatisticamente com o K-Means em métricas clássicas e com o DBSCAN em métricas de densidade simultaneamente.

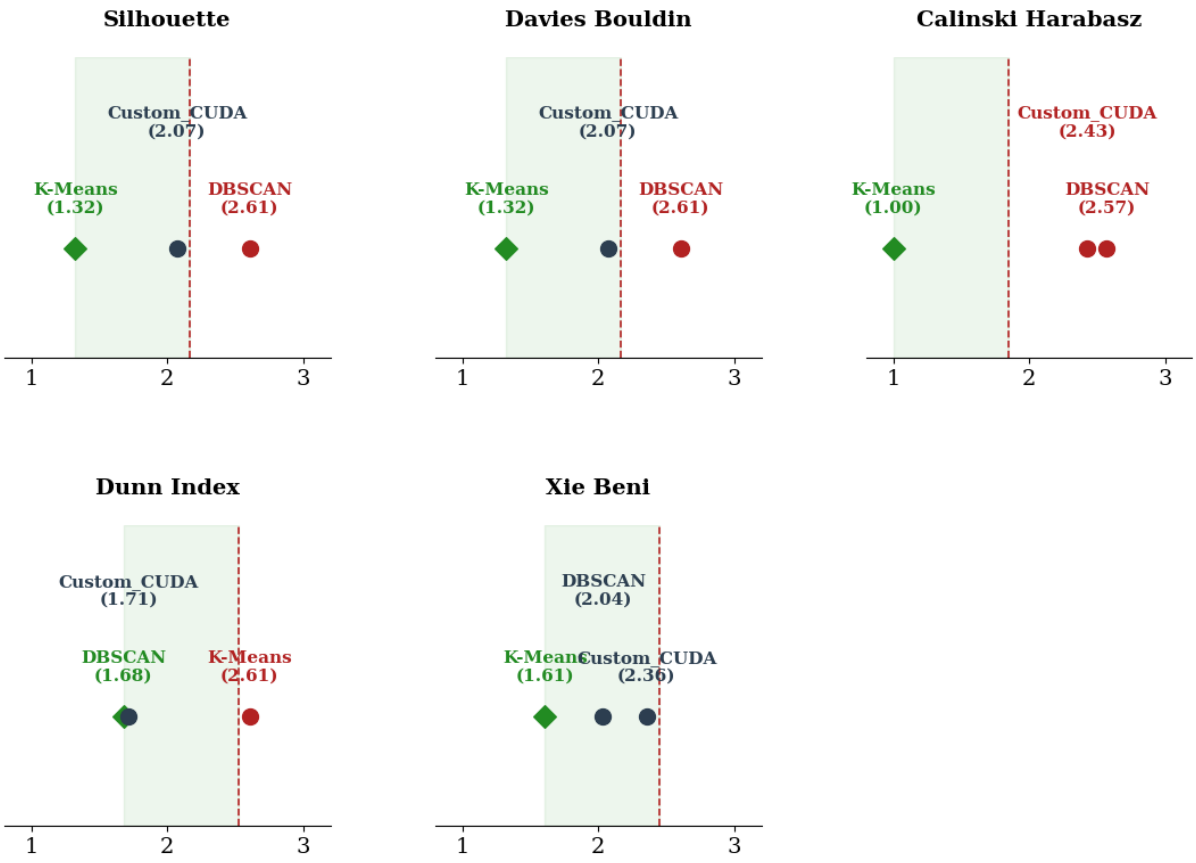

Figura 13 – Representação Bonferroni-Dunn K Otimizado.

6 Conclusão

Este trabalho apresentou o desenvolvimento e a análise de estratégias de paralelização para um classificador de margem larga baseado em grafos, utilizando a arquitetura de processamento em GPUs via plataforma CUDA. A motivação central da pesquisa se baseou na necessidade de acelerar o tempo de execução deste algoritmo, que, em sua versão sequencial, apresenta limitações de desempenho ao lidar com grandes volumes de dados, inviabilizando sua aplicação em cenários de tempo real ou de larga escala.

O objetivo geral de investigar e avaliar o impacto da distribuição de carga na paralelização foi atingido com êxito. Os experimentos realizados demonstraram que a utilização de GPUs proporcionou ganhos de desempenho expressivos em comparação à implementação sequencial. Conforme evidenciado nos resultados, obteve-se um *speedup* que ultrapassou a marca de 8000 vezes em determinados conjuntos de dados, como o *German*, reduzindo o tempo de processamento de dezenas de segundos para a ordem de milissegundos.

Em relação aos objetivos específicos definidos no início deste estudo, conclui-se que:

- A implementação da versão sequencial permitiu validar a lógica do classificador e estabelecer uma linha de base sólida para as comparações de desempenho.
- O desenvolvimento da versão paralela inicial, mapeando o cálculo de pares de pontos a *threads* distintas, mostrou-se eficaz para explorar o paralelismo massivo da GPU.
- A otimização através do ajuste de granularidade (múltiplos pares por *thread*) permitiu investigar o comportamento do algoritmo sob diferentes cargas de trabalho, identificando configurações que maximizam o uso dos recursos computacionais.
- A avaliação de desempenho confirmou a superioridade da abordagem paralela em termos de tempo de execução, mantendo a consistência nos valores de acurácia e F1-score, o que atesta a correção da implementação paralela em relação à original.

A análise dos resultados sugere que a arquitetura paralela das GPUs é uma solução viável e altamente eficiente para superar os gargalos computacionais deste tipo de classificador. A drástica redução no tempo de execução viabiliza a utilização destes modelos em problemas complexos e áreas críticas, como diagnósticos médicos e engenharia, onde a agilidade na resposta é fundamental.

Adicionalmente, este trabalho cumpriu o objetivo de expandir a aplicabilidade da estrutura do Grafo de Gabriel para tarefas não supervisionadas. O algoritmo de clusterização

proposto, implementado inteiramente em GPU, demonstrou robustez estatística quando comparado ao estado da arte. As análises baseadas nos testes de Friedman e Wilcoxon indicaram que a abordagem proposta atinge qualidade de agrupamento equivalente ao DBSCAN em métricas de densidade e topologia, superando-o significativamente no índice Xie-Beni. Destaca-se que tal desempenho foi obtido utilizando a heurística do cotovelo, eliminando a necessidade de busca exaustiva por hiperparâmetros, o que representa uma vantagem operacional relevante em ambientes de produção.

Em suma, esta monografia reafirma o potencial da computação de alto desempenho como viabilizadora de métodos geométricos complexos na Inteligência Artificial. Ao mitigar o gargalo computacional de $O(n^3)$ através do paralelismo massivo, demonstrou-se que é possível aliar a precisão teórica dos modelos baseados em grafos à eficiência exigida pelas aplicações modernas. Espera-se que as estratégias de otimização e os algoritmos aqui apresentados sirvam de base para novas investigações que busquem unir a geometria computacional à escalabilidade das arquiteturas paralelas.

6.1 Trabalhos Futuros

Apesar dos resultados, existem oportunidades para a continuidade e expansão deste trabalho. Como sugestões, destacam-se:

- Análise aprofundada de eficiência energética e de memória: Realizar um estudo detalhado sobre o consumo energético e gasto de memória das diferentes implementações, e sua caracterização em conceitos como algoritmos verdes.
- Execução em Múltiplas GPUs: Investigar estratégias de paralelismo distribuído utilizando diferentes GPUs para lidar com conjuntos de dados massivos que excedam a memória de um único dispositivo.
- Aplicação em *Big Data*: Avaliar o comportamento do classificador paralelizado em bases de dados com milhões de instâncias, testando os limites de escalabilidade da solução proposta.
- Melhoria do ponto de corte: Apesar da técnica do ponto de cotovelo ser promissora outras técnicas como cortes estatístico podem ser avaliadas.

Referências

ARIAS-GARCIA, J. et al. Improved design for hardware implementation of graph-based large margin classifiers for embedded edge computing. *IEEE Transactions on Neural Networks and Learning Systems*, v. 35, n. 1, p. 1320–1329, 2024. Citado 4 vezes nas páginas 19, 20, 22 e 32.

CHEN, M.; MAO, S.; LIU, Y. Big data: A survey. *Mobile Networks and Applications*, Springer, v. 19, n. 2, p. 171–209, 2014. Citado na página 16.

CORMEN, T. H. et al. Algoritmos: teoria e prática. *Editora Campus*, v. 2, p. 296, 2002. Citado na página 26.

EON, W. et al. Deep learning with gpus. In: KIM, S.; DEKA, G. C. (Ed.). *Advances in Computers*. [S.l.]: Elsevier, 2021. v. 122, p. 167–215. Citado 2 vezes nas páginas 19 e 20.

EZUGWU, A. E. et al. A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, v. 110, p. 104743, 2022. ISSN 0952-1976. Disponível em: <https://www.sciencedirect.com/science/article/pii/S095219762200046X>. Citado 2 vezes nas páginas 27 e 28.

GABRIEL, K. R.; SOKAL, R. R. A new statistical approach to geographic variation analysis. *Systematic Biology*, v. 18, n. 3, p. 259–278, 09 1969. ISSN 1063-5157. Disponível em: <https://doi.org/10.2307/2412323>. Citado na página 23.

GADE, L. et al. Nn-clas: classificador geométrico de margem larga baseado na regra do vizinho mais próximo. In: . [S.l.: s.n.], 2018. p. 1–12. Citado 5 vezes nas páginas 9, 19, 23, 24 e 25.

GANDOMI, A.; HAIDER, M. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, v. 35, n. 2, p. 137–144, 2015. ISSN 0268-4012. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0268401214001066>. Citado na página 16.

GARCIA, L. P.; de Carvalho, A. C.; LORENA, A. C. Effect of label noise in the complexity of classification problems. *Neurocomputing*, v. 160, p. 108–119, 2015. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0925231215001241>. Citado 2 vezes nas páginas 19 e 24.

HAN, J.; PEI, J.; TONG, H. *Data mining: concepts and techniques*. [S.l.]: Morgan kaufmann, 2022. Citado na página 16.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd. ed. [S.l.]: Springer, 2009. ISBN 978-0387848570. Citado 2 vezes nas páginas 21 e 22.

JAIN, A. K. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, v. 31, n. 8, p. 651–666, 2010. ISSN 0167-8655. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR). Disponível em:

<https://www.sciencedirect.com/science/article/pii/S0167865509002323>. Citado na página 22.

MAGNI, A.; DUBACH, C.; O'BOYLE, M. Automatic optimization of thread-coarsening for graphics processors. *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, 08 2014. Citado 2 vezes nas páginas 19 e 20.

NICKOLLS, J. et al. Scalable parallel programming with cuda. In: *ACM SIGGRAPH 2008 Classes*. New York, NY, USA: Association for Computing Machinery, 2008. (SIGGRAPH '08). ISBN 9781450378451. Disponível em: <https://doi.org/10.1145/1401132.1401152>. Citado na página 17.

NVIDIA. *CUDA Toolkit Documentation*. 2025. Acesso em: 11 out. 2025. Disponível em: <https://docs.nvidia.com/cuda/>. Citado na página 29.

NVIDIA Corporation. *CUDA C++ Programming Guide*. [S.l.], 2025. Version 13.0. Acessado em: 27 out. 2025. Disponível em: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/>. Citado 2 vezes nas páginas 29 e 30.

OH, F. *O Que É CUDA?* 2024. Acesso em: 11 out. 2025. Disponível em: <https://blog.nvidia.com.br/blog/o-que-e-cuda/>. Citado na página 29.

OWENS, J. D. et al. Gpu computing. *Proceedings of the IEEE*, v. 96, n. 5, p. 879–899, 2008. Citado na página 29.

SANDERS, J.; KANDROT, E. *CUDA by example: an introduction to general-purpose GPU programming*. [S.l.]: Addison-Wesley Professional, 2010. Citado na página 17.

THORNDIKE, R. L. Who belongs in the family? *Psychometrika*, Springer-Verlag, v. 18, n. 4, p. 267–276, 1953. Citado na página 26.

ZHANG, Y. et al. A gpu-based computational framework that bridges neuron simulation and artificial intelligence. *Nature Communications*, v. 14, p. 5798, 2023. Citado 3 vezes nas páginas 16, 19 e 20.

ÇOLHAK, F. et al. *Accelerating IoV Intrusion Detection: Benchmarking GPU-Accelerated vs CPU-Based ML Libraries*. 2025. <https://arxiv.org/abs/2504.01905>. Available at: <https://arxiv.org/abs/2504.01905>. Citado 3 vezes nas páginas 16, 19 e 20.