

MultiDBSCAN: Exploração Paralela de Múltiplos Raios em GPU

Walter Bueno¹, Matheus Pereira¹, Olavo Barros¹, José Nacif¹, Ricardo Ferreira¹

¹Departamento de Informática – Universidade Federal de Viçosa (UFV)
36570-900 – Viçosa – MG – Brazil

{walter.b.neto, matheus.antony, olavo.barros, jnacif, ricardo}@ufv.br

Abstract. DBSCAN identifies arbitrarily shaped clusters based on point density. However, its results are sensitive to the neighborhood radius ε , which often requires evaluating a range of values and comparing the resulting partitions. GPU implementations commonly process a single ε per execution and therefore repeat distance computations for every tested radius. This paper presents MultiDBSCAN, an approach that evaluates multiple ε values in one execution while keeping $\minPts$ fixed. Distances are computed in tiles of the dot-product matrix and shared among the radius, while independent Union-Find structures construct the clusters of each configuration. Across six datasets and five radii, MultiDBSCAN achieved geometric-mean speedups of $5.09\times$ over the sum of five independent NVIDIA cuML executions and $32.78\times$ over the sum of five CUDA-DClust+ executions, while maintaining a mean adjusted Rand index (ARI) above 0.98 relative to cuML.

Resumo. O DBSCAN identifica agrupamentos de formatos arbitrários com base na densidade dos pontos. Entretanto, seus resultados são sensíveis ao raio de vizinhança ε , o que, na prática, exige avaliar uma faixa de valores e comparar as partições resultantes. Implementações em GPU geralmente processam um único ε por execução e, portanto, repetem o cálculo das distâncias para cada raio testado. Este trabalho apresenta o MultiDBSCAN, que avalia vários valores de ε em uma única execução, mantendo $\minPts$ fixo. As distâncias são obtidas em ladrilhos da matriz de produtos escalares e compartilhadas entre os raios, enquanto estruturas Union-Find independentes constroem os agrupamentos de cada configuração. Em seis conjuntos de dados e cinco raios, o método alcançou aceleração média geométrica de $5,09\times$ sobre a soma de cinco execuções independentes do NVIDIA cuML e de $32,78\times$ sobre a soma de cinco execuções do CUDA-DClust+, mantendo ARI médio acima de 0,98 em relação ao cuML.

1. Introdução

O DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) [Ester et al. 1996] é um algoritmo de clusterização amplamente utilizado. Destaca-se por sua capacidade de identificar agrupamentos de formatos arbitrários e isolar ruídos em várias áreas: no monitoramento geoespacial de doenças infecciosas [Kuo et al. 2018], na otimização de sistemas de transporte e de tráfego [Zhang et al. 2021] e na resposta a emergências [Kiran et al. 2024]. O DBSCAN também é amplamente empregado na

detecção de anomalias (*outliers*) [Çelik et al. 2011, Zhou et al. 2024], no processamento de imagens, na análise de dados astronômicos e na segmentação de usuários em sistemas de recomendação.

O funcionamento do DBSCAN baseia-se na identificação de regiões de alta densidade, que formam agrupamentos (*clusters*), enquanto pontos localizados em regiões de baixa densidade são classificados como ruído (*noise*). Entretanto, a capacidade de identificar os agrupamentos depende de dois hiperparâmetros. O primeiro é ε , que define o raio da vizinhança. O segundo parâmetro é *minPts*, o número mínimo de pontos necessário para que um ponto seja classificado como ponto central (*core point*). Assim, um dos principais desafios é determinar valores adequados, pois diferentes combinações de ε e *minPts* podem produzir agrupamentos distintos.

Portanto, é comum executar o algoritmo repetidas vezes, variando as combinações de ε e *minPts* em uma exploração em grade. Essa estratégia permite avaliar os resultados produzidos por cada configuração, mas impõe um custo computacional. Neste trabalho, a proposta é reduzir esse custo com GPUs, explorando o potencial de compartilhar recursos entre avaliações com diferentes valores de ε , mantendo *minPts* fixo. A estratégia envolve maximizar o reuso dos dados, uma vez que a memória é o gargalo, limitando o potencial de paralelismo. Como a busca exata de vizinhos realiza comparações par a par, a execução tende a ser limitada pela largura de banda da memória (*memory-bound*) e pelo custo do cálculo de distâncias. Como o conjunto de dados permanece inalterado entre as configurações, há a oportunidade de explorar a monotonicidade de ε e o reuso de dados para amortizar parte desse trabalho repetido.

Este trabalho está dividido da seguinte forma: a Seção 2 apresenta os trabalhos relacionados, com ênfase nas implementações paralelas do DBSCAN e na necessidade de executar separadamente cada valor de ε . A Seção 3 descreve o MultiDBSCAN, incluindo o cálculo das distâncias em ladrilhos, a contagem compartilhada dos vizinhos e a construção dos agrupamentos. A Seção 4 apresenta os resultados experimentais e as comparações com o cuML [Sarkar 2022] e o CUDA-DClust+ [Poudel and Gowanlock 2021b]. Por fim, a Seção 5 apresenta as conclusões e as possibilidades de trabalhos futuros.

2. Trabalhos Relacionados

Apesar de sua capacidade de identificar grupos de formatos arbitrários, como mencionado, o DBSCAN apresenta alta sensibilidade ao parâmetro de raio ε . Essa limitação foi identificada em seu trabalho introdutório [Ester et al. 1996] e reforçada em trabalhos posteriores [Bushra and Yi 2021], gerando a necessidade de realizar múltiplas execuções a fim de identificar um valor adequado de ε [Ankerst et al. 1999].

O DBSCAN possui complexidade quadrática, $O(N^2)$, quando todos os pontos são comparados entre si. Porém, em casos particulares, como em dados bidimensionais ou com o uso de R-tree [Goulart et al. 2024], essa complexidade pode ser reduzida para $O(N \log N)$. GPUs vêm sendo usadas para reduzir o custo computacional do algoritmo [Gupta et al. 2024]. O CUDA-DClust, proposto por Böhm et al. [Böhm et al. 2009], utiliza técnicas de agrupamento como a expansão simultânea de múltiplas cadeias independentes com fusão por colisão, enquanto Andrade et al. [Andrade et al. 2013] apresentam o algoritmo G-DBSCAN, que utiliza grafos de adjacência em GPU. Outros trabalhos, como a implementação de DBSCAN do ecossistema

NVIDIA RAPIDS, disponível no cuML [Sarkar 2022], incorporaram estruturas de indexação espacial aceleradas em GPU para otimizar as operações de busca por vizinhos.

Mais recentemente, Poudel e Gowanlock propuseram o CUDA-DClust+ [Poudel and Gowanlock 2021b], uma revisão do CUDA-DClust [Böhm et al. 2009] voltada às arquiteturas modernas de GPU. Em sua avaliação, o método também foi comparado com o G-DBSCAN [Andrade et al. 2013] e com o CPU-DBSCAN [Thapa et al. 2010]. Entre suas principais otimizações estão a construção da estrutura de indexação diretamente na GPU e o uso de *kernel fusion* para combinar as etapas de *index search* e *cluster expansion*. O método também transfere para a GPU grande parte do gerenciamento das *seed lists*, que armazenam os pontos ainda pendentes durante a expansão de cada agrupamento. Essas estratégias reduzem as transferências de dados e as sincronizações entre CPU e GPU. Os autores avaliaram o método em seis conjuntos de dados reais e reportaram aceleração de até aproximadamente $23\times$ em relação ao G-DBSCAN.

Apesar dessas otimizações, as abordagens apresentadas processam apenas uma configuração de ε por execução. Consequentemente, a exploração de uma faixa de valores exige que o algoritmo seja executado novamente para cada raio, repetindo etapas como a busca de vizinhos e o cálculo das distâncias. O MultiDBSCAN explora a monotonicidade dos valores de ε e o paralelismo da GPU para avaliar múltiplos raios em uma única execução, compartilhando o cálculo geométrico entre as configurações. Trabalhos anteriores aplicaram a estratégia de múltiplas avaliações em GPU ao K-means [Bueno et al. 2024, Barros et al. 2026].

3. MultiDBSCAN

3.1. Visão Geral da Solução

Seja $X \in \mathbb{R}^{N \times D}$ um conjunto contendo N pontos, cada um com D dimensões. O algoritmo avalia simultaneamente E valores ordenados de raio, de ε_0 até ε_{E-1} , mantendo $minPts$ fixo durante toda a execução. A abordagem proposta compartilha o cálculo geométrico entre os diferentes raios. Em cada passagem, a distância entre um par de pontos é calculada e reutilizada por todas as configurações de ε .

O fluxo completo é apresentado no Algoritmo 1. Após a centralização dos pontos e o cálculo das normas quadráticas, uma primeira passagem pelos ladrilhos conta os vizinhos para todos os valores de ε por meio dos histogramas cumulativos. Em seguida, os pontos centrais são identificados, e uma segunda passagem pela região triangular superior une os pontos centrais e localiza os candidatos de borda de cada configuração. Por fim, os caminhos das estruturas Union-Find são comprimidos e os rótulos dos pontos centrais, de borda e de ruído são produzidos.

Algoritmo 1: DBSCAN para múltiplos valores de ε .

Entrada: $X \in \mathbb{R}^{N \times D}$; raios ordenados $\mathcal{E} = \{\varepsilon_0, \dots, \varepsilon_{E-1}\}$; $minPts$; tile T
Saída: labels $L \in \mathbb{Z}^{E \times N}$

```
1  $X \leftarrow \text{CentralizarColunas}(X)$   $s \leftarrow \text{NormasQuadraticasParalelas}(X)$ 
    $H \leftarrow \mathbf{0}_{E \times N}$ ;
   // Primeira passagem: contar os vizinhos
2 foreach par de tiles  $(X_r, X_c)$  da matriz completa do
3    $G \leftarrow X_r X_c^\top$ ,  $\text{AcumularHistogramasMultiEps}(G, s, \mathcal{E}, H)$ ;
4  $V \leftarrow \text{SomadasCumulativasParalelas}(H)$ ;
   // Inicializar os estados de todas as configurações
5 foreach  $(e, i)$  em paralelo do
6   if  $V[e, i] \geq minPts$  then
7      $P[e, i] \leftarrow i$ ;
8   else
9      $P[e, i] \leftarrow -1$ ;
10   $B[e, i] \leftarrow +\infty$ ;
   // Segunda passagem: construir clusters e localizar
   bordas
11 foreach par de tiles  $(X_r, X_c)$  da região triangular superior do
12    $G \leftarrow X_r X_c^\top$ ;
13    $\text{ProcessarParesMultiEps}(G, s, \mathcal{E}, V, minPts, P, B)$ ;
14  $P \leftarrow \text{ComprimirCaminhosParalelos}(P)$ ;
15 foreach  $(e, i)$  em paralelo do
16   if  $P[e, i] \geq 0$  then  $L[e, i] \leftarrow P[e, i]$ ;
17   else if  $B[e, i] < \infty$  then  $L[e, i] \leftarrow P[e, B[e, i]]$ ;
18   else  $L[e, i] \leftarrow -1$ ;
19 retorne  $L$ ;
```

3.2. Centralização e Cálculo em Ladrilhos

Cada dimensão é centralizada pela subtração de sua respectiva média. Como a distância euclidiana é invariante a translações, essa transformação não altera as relações de vizinhança entre os pontos e reduz a magnitude das coordenadas.

A distância euclidiana quadrática entre dois pontos é calculada por $d_{ij}^2 = \|x_i\|_2^2 + \|x_j\|_2^2 - 2x_i^\top x_j$, na qual $\|x_i\|_2^2$ representa a norma quadrática do ponto x_i . As normas de todos os pontos são calculadas uma única vez e reutilizadas. O cálculo simultâneo das distâncias entre todos os pares de pontos produz uma matriz de dimensões $N \times N$, cujo armazenamento pode se tornar inviável para conjuntos de dados grandes. Para evitar esse custo de memória, a matriz é processada em ladrilhos.

Cada ladrilho de linhas contém um grupo de pontos que será comparado com outro grupo, pertencente a um ladrilho de colunas. O produto escalar entre esses dois grupos é calculado pela rotina `cublasSgemv`. O resultado é uma matriz temporária na qual cada posição (i, j) corresponde ao produto escalar entre um ponto do ladrilho de linhas e um ponto do ladrilho de colunas. As normas previamente calculadas são combinadas com os produtos escalares, produzindo as distâncias quadráticas do ladrilho. Essas distâncias são imediatamente utilizadas na contagem de vizinhos e, em seguida, o mesmo espaço de memória é reutilizado para o próximo par de ladrilhos, evitando

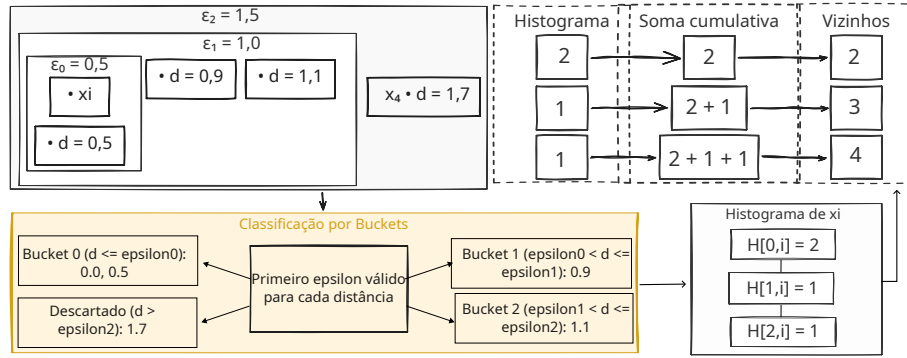


Figura 1. Classificação das distâncias em intervalos e obtenção da quantidade de vizinhos para múltiplos valores de ε . Cada distância é inserida no primeiro intervalo que a aceita, e a soma cumulativa obtém as contagens para todos os raios.

armazenar simultaneamente todas as N^2 distâncias.

3.3. Contagem de Vizinhos para Múltiplos Valores de ε

A contagem dos vizinhos é realizada durante a primeira passagem pelos ladrilhos de distâncias. Nessa passagem, os pontos do ladrilho de linhas são utilizados como referência, enquanto os pontos do ladrilho de colunas são avaliados como possíveis vizinhos. Todos os pares de ladrilhos são processados para que cada ponto tenha sua própria contagem de vizinhos.

Para evitar a repetição da contagem para cada valor de ε , os raios são mantidos em ordem crescente e cada distância é associada somente ao primeiro raio que a aceita. Essa associação produz um histograma para cada ponto. Posteriormente, a soma cumulativa desse histograma fornece a quantidade de vizinhos em todos os valores de ε .

A Figura 1 exemplifica esse procedimento para um ponto x_i e três valores de raio: $\varepsilon_0 = 0,5$, $\varepsilon_1 = 1,0$ e $\varepsilon_2 = 1,5$. O próprio ponto x_i possui distância zero e, juntamente com o ponto situado à distância 0,5, é associado ao primeiro intervalo. Portanto, a primeira posição do histograma recebe o valor 2. O ponto situado à distância 0,9 não pertence ao primeiro raio, mas é aceito por ε_1 e acrescenta uma ocorrência à segunda posição. Da mesma maneira, a distância 1,1 é aceita inicialmente por ε_2 e acrescenta uma ocorrência à terceira posição. A distância 1,7 não é aceita por nenhum dos raios avaliados e, por isso, é descartada.

Como resultado, o histograma do ponto x_i contém os valores 2, 1 e 1. Esses valores ainda representam apenas os novos vizinhos encontrados em cada intervalo. A soma cumulativa transforma esse histograma nas contagens 2, 3 e 4. Dessa forma, x_i possui dois vizinhos para ε_0 , três para ε_1 e quatro para ε_2 .

Essa estratégia permite reutilizar cada distância para todos os valores de ε . Assim, o cálculo das distâncias não precisa ser repetido para cada configuração. O custo adicional dos múltiplos valores de ε fica concentrado na atualização e na soma cumulativa dos histogramas.

3.4. Identificação dos Pontos Centrais e Construção dos Agrupamentos com Union-Find

Após a contagem dos vizinhos, cada ponto possui uma quantidade de vizinhos associada a cada valor de ε . Um ponto é classificado como central quando essa quantidade é maior ou igual a $minPts$. Essa classificação é realizada em paralelo para todas as combinações entre pontos e valores de ε . Como a vizinhança cresce à medida que ε aumenta, um ponto que se torna central permanece central nas configurações seguintes. O resultado dessa etapa é uma representação dos pontos centrais de cada configuração, utilizada posteriormente na construção dos agrupamentos.

Após a identificação dos pontos centrais, é realizada uma segunda passagem pelos ladrilhos de distâncias. Essa nova passagem é necessária porque os ladrilhos calculados durante a contagem dos vizinhos são consumidos imediatamente e não são mantidos em memória. Como a distância entre dois pontos é simétrica, somente uma das metades da matriz de distâncias precisa ser processada. Para cada par de pontos, o método identifica a partir de qual valor de ε eles são considerados vizinhos. Essa relação também é válida para todos os valores maiores de ε avaliados pelo algoritmo. Para cada valor de ε , é mantida uma estrutura Union-Find independente. Quando dois pontos vizinhos são classificados como centrais na mesma configuração, eles são unidos nessa estrutura. Ao final da passagem, pontos centrais conectados direta ou indiretamente pertencem ao mesmo conjunto e, portanto, formam um agrupamento.

3.5. Atribuição dos Pontos de Borda

A identificação dos pontos de borda é realizada durante a mesma passagem utilizada para unir os pontos centrais. Quando um par de vizinhos contém um ponto central e outro que não é central, o primeiro é registrado como uma possível referência de agrupamento para o segundo.

Um ponto pode estar na vizinhança de pontos centrais pertencentes a agrupamentos diferentes. Para tornar o resultado reproduzível, é selecionado sempre o candidato de menor índice. Após a conclusão das uniões, cada ponto de borda recebe o mesmo rótulo do ponto central selecionado. Por fim, os pontos centrais recebem o identificador de seu respectivo conjunto no Union-Find, os pontos de borda recebem o identificador de um agrupamento vizinho e os pontos que não atendem a nenhuma dessas condições são classificados como ruído.

3.6. Trabalho por Warp na Contagem de Vizinhos

O *kernel* responsável pela construção dos histogramas utiliza blocos de 256 *threads*, correspondentes a oito *warps*. Em vez de atribuir uma linha inteira do ladrilho a uma única *thread*, o método permite que um grupo de W *warps* coopere no processamento dessa linha. Cada *warp* contém 32 *threads*, chamadas de *lanes*, e cada *lane* avalia uma coluna diferente do ladrilho. Por exemplo, quando são utilizados dois *warps* por linha, o primeiro *warp* processa inicialmente as colunas de 0 a 31, enquanto o segundo processa as colunas de 32 a 63. Caso a linha contenha mais colunas, os mesmos *warps* continuam o processamento nos grupos seguintes. Dessa forma, o primeiro *warp* avança para as colunas de 64 a 95 e o segundo para as colunas de 96 a 127, repetindo esse procedimento até o final da linha.

Tabela 1. Ambiente utilizado nos experimentos.

Componente	Configuração
GPU	NVIDIA GeForce RTX 4060 Ti, com 8.188 MiB de memória
CPU	AMD Ryzen 7 5700X, com 8 núcleos físicos e 16 <i>threads</i>
Memória principal	15 GiB disponibilizados ao ambiente WSL2
Sistema operacional	Ubuntu 24.04.4 LTS executado sobre WSL2
Driver NVIDIA	versão 610.88
CUDA Toolkit	versão 12.0
cuBLAS	versão 12.0.2.224
RAPIDS/cuML	versão 25.10.0
CuPy	versão 14.1.1 para CUDA 12
Compilador CUDA	NVCC 12.0.140
Compilador C++	GCC/G++ 13.3.0
Compilação	-O3 e -arch=sm_89
Precisão	float32 (MultiDBSCAN e cuML) e float64 (CUDA-DClust+)

Cada distância é associada ao primeiro intervalo de ε que a aceita. As *lanes* de um *warp* realizam essa classificação simultaneamente e votam no intervalo correspondente. A quantidade de votos é, então, contabilizada e armazenada em um histograma parcial mantido na memória compartilhada. Essa organização permite combinar os resultados das 32 *lanes* antes de atualizar o histograma global, reduzindo a quantidade de acessos concorrentes à memória.

Quando mais de um *warp* trabalha sobre a mesma linha, cada um produz seu próprio histograma parcial. Ao final do processamento, o primeiro *warp* do grupo soma esses resultados e atualiza o histograma associado ao ponto daquela linha. Assim, a cooperação ocorre tanto na avaliação das distâncias quanto na combinação das contagens produzidas pelos diferentes *warps*. A implementação permite utilizar um, dois, quatro ou oito *warps* por linha. Com um *warp* por linha, um bloco pode processar oito linhas simultaneamente. Com dois *warps*, processa quatro linhas; com quatro *warps*, processa duas; e, com oito *warps*, todo o bloco coopera sobre uma única linha. Portanto, aumentar o número de *warps* por linha eleva o paralelismo aplicado a cada ponto, mas reduz a quantidade de pontos processados simultaneamente pelo bloco.

4. Resultados Experimentais

4.1. Ambiente Experimental

Os experimentos foram conduzidos em uma única plataforma, descrita na Tabela 1, de modo que todas as implementações fossem avaliadas sob as mesmas condições de hardware e software. As execuções utilizaram a mesma GPU, sem concorrência intencional com outras cargas computacionais. O MultiDBSCAN foi compilado com -O3 para a arquitetura sm_89. O MultiDBSCAN e o cuML utilizaram pontos armazenados em float32; o CUDA-DClust+ preservou a representação em float64 de sua implementação original. No MultiDBSCAN, o cuBLAS foi configurado com CUBLAS_PEDANTIC_MATH.

O tempo de *fit* representa a região computacional de agrupamento e não o tempo

de ponta a ponta. No MultiDBSCAN, a medição começa após a centralização e a cópia dos pontos para a GPU e termina antes da cópia dos rótulos para a CPU; o intervalo medido engloba a criação das estruturas internas, as duas passagens e a finalização. No cuML, os pontos também são copiados previamente para a GPU, e a medição envolve a chamada de ajuste do modelo para cada raio. No CUDA-DClust+, o intervalo inclui a construção do índice e a expansão dos agrupamentos, conforme a instrumentação de sua própria implementação. Em todos os casos, a leitura e a escrita de arquivos, assim como as transferências externas ao intervalo de ajuste, foram excluídas. Por isso, as comparações desta seção referem-se ao tempo de *fit*, e não ao tempo de ponta a ponta.

4.2. Múltiplos Valores de ε Simultâneos

Na comparação principal, cada configuração do MultiDBSCAN e do cuML foi executada uma vez para aquecimento, seguida de cinco repetições medidas. A ordem dos métodos foi rotacionada entre as repetições. Para o CUDA-DClust+, foram consideradas somente as medições concluídas com sucesso em cada valor de ε . A Tabela 2 apresenta as médias das medições válidas.

Tabela 2. Tempo de execução: MultiDBSCAN, cuML e CUDA-DClust+ (Fast).

Conjunto	N	D	Faixa de ε	ε^*	MultiDBSCAN		cuML				Fast			
					Gprod/s	Tempo (s)	Vários		Único		Vários		Único	
							Tempo (s)	Acel.	Tempo (s)	Acel.	Tempo (s)	Acel.	Tempo (s)	Acel.
NGSIM	400.000	2	0,500–1,500	1,250	41,95	<u>5,73</u>	22,85	3,99	<u>4,55</u>	0,79	172,94	30,18	34,33	5,99
Spatial	400.000	2	0,002–0,010	0,008	42,80	<u>5,62</u>	23,42	4,17	<u>4,70</u>	0,84	169,24	30,13	33,54	5,97
Iono	400.000	2	0,500–1,500	1,500	41,90	<u>5,74</u>	24,65	4,29	<u>5,09</u>	0,89	165,79	28,88	33,61	5,85
Porto	160.000	2	0,002–0,010	0,008	35,32	<u>1,09</u>	8,86	8,10	<u>2,12</u>	1,94	55,45	50,73	10,95	10,02
Spatial3D	400.000	3	0,020–0,100	0,080	42,46	<u>5,66</u>	31,07	5,49	<u>6,24</u>	1,10	169,44	29,92	33,55	5,93
Iono3D	400.000	3	0,500–1,500	1,500	42,05	<u>5,73</u>	31,28	5,46	<u>6,40</u>	1,12	178,32	31,15	35,96	6,28
Média	–	–	–	–	41,08	<u>4,93</u>	23,69	5,09	<u>4,85</u>	1,06	151,86	32,78	30,32	6,54

Notas: “Vários” representa a soma das cinco execuções escalares, e “Único”, a execução com o ε^* adotado pelo CUDA-DClust+ [Poudel and Gowanlock 2021b]. Acelerações superiores a 1 indicam vantagem do MultiDBSCAN. Gprod/s representa bilhões de produtos escalares por segundo. Na linha “Média”, os tempos e a vazão são médias aritméticas, enquanto as acelerações são médias geométricas. Em “Vários”, está sublinhado o menor tempo entre os três métodos; em “Único”, está sublinhado o menor tempo entre as duas execuções escalares. O tempo do MultiDBSCAN é o mesmo nas duas comparações e corresponde à execução conjunta dos cinco valores de ε .

A Tabela 2 compara o tempo de *fit* do MultiDBSCAN com o cuML e o CUDA-DClust+ (Fast) [Poudel and Gowanlock 2021b], utilizando os seis conjuntos de dados disponibilizados pelos autores para a avaliação [Poudel and Gowanlock 2021a]. Na configuração “Vários”, correspondente à soma das cinco execuções escalares necessárias para cobrir a faixa de ε , o MultiDBSCAN apresentou o menor tempo em todos os conjuntos de dados. Em relação ao cuML, as acelerações variaram de $3,99\times$, no NGSIM, a $8,10\times$, no Porto, com média geométrica de $5,09\times$. Em relação ao Fast, as acelerações variaram de $28,88\times$, no Iono, a $50,73\times$, no Porto, com média geométrica de $32,78\times$. O MultiDBSCAN também alcançou uma vazão média de 41,08 Gprod/s.

Na configuração “Único”, em que o cuML e o Fast executam somente o ε^* adotado como configuração principal, o cuML foi mais rápido que o MultiDBSCAN no NGSIM, Spatial e Iono, apresentando acelerações entre $0,79\times$ e $0,89\times$. Por outro lado, o MultiDBSCAN permaneceu mais rápido no Porto, Spatial3D e Iono3D, com acelerações de $1,94\times$, $1,10\times$ e $1,12\times$, respectivamente. O Fast foi superado pelo MultiDBSCAN mesmo

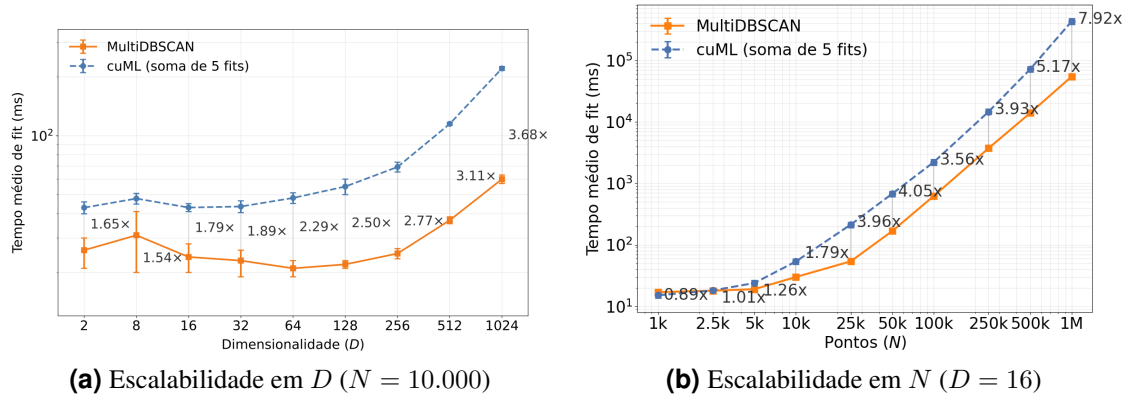


Figura 2. Tempo de *fit* do MultiDBSCAN e do cuML conforme (a) a dimensionalidade D aumenta com $N = 10.000$ e (b) o número de pontos N aumenta com $D = 16$. O tempo do cuML corresponde à soma de cinco execuções, uma por raio. Os segmentos verticais e suas anotações representam a aceleração do MultiDBSCAN; as barras de erro indicam um desvio padrão quando há medições repetidas.

nesse cenário, com acelerações entre $5,85\times$, no Iono, e $10,02\times$, no Porto, e média geométrica de $6,54\times$. Esses resultados evidenciam a principal vantagem do MultiDBSCAN: o compartilhamento do cálculo das distâncias entre diferentes valores de ε reduz substancialmente o custo de avaliar várias configurações, embora o cuML possa ser mais rápido quando apenas uma configuração isolada é considerada.

Além do ganho de desempenho, o MultiDBSCAN apresentou elevada concordância com o cuML. O índice de Rand ajustado (ARI) médio permaneceu acima de 0,98 em todos os conjuntos de dados, atingindo concordância perfeita no Iono (ARI = 1,000.0). O menor resultado foi observado no Spatial3D, com ARI mínimo de 0,962.7 e médio de 0,982.6. A concordância na classificação de ruído também permaneceu elevada, com médias superiores a 0,997 e mínimos não inferiores a 0,988.

Para investigar as discordâncias, foram construídos casos controlados com distâncias próximas ao limiar definido por ε . Os resultados mostraram que diferenças podem ser introduzidas pela conversão das coordenadas para `float32`, pelo cancelamento numérico na identidade baseada na matriz de Gram e pela ordem de acumulação das operações realizadas pelo cuBLAS. Portanto, os resultados indicam elevada similaridade entre os agrupamentos, mas não equivalência numérica completa, especialmente para pares próximos ao limiar de vizinhança. A análise separada de pontos centrais, ruído e componentes centrais permite distinguir essas diferenças numéricas das atribuições admissíveis de pontos de borda.

Os conjuntos sintéticos foram produzidos com o gerador `make_blobs`, usando semente 42, oito centros, desvio padrão 0,55 em cada agrupamento e armazenamento em `float32`. Nos experimentos de escalabilidade, variaram-se N , D e os raios, enquanto o gerador e $\text{minPts} = 5$ permaneceram fixos.

Os experimentos sintéticos de alta dimensionalidade comparam apenas o MultiDBSCAN e o cuML. A implementação do CUDA-DClust+ define a dimensionalidade durante a compilação, e sua indexação pode examinar até 3^D células adjacentes. Como

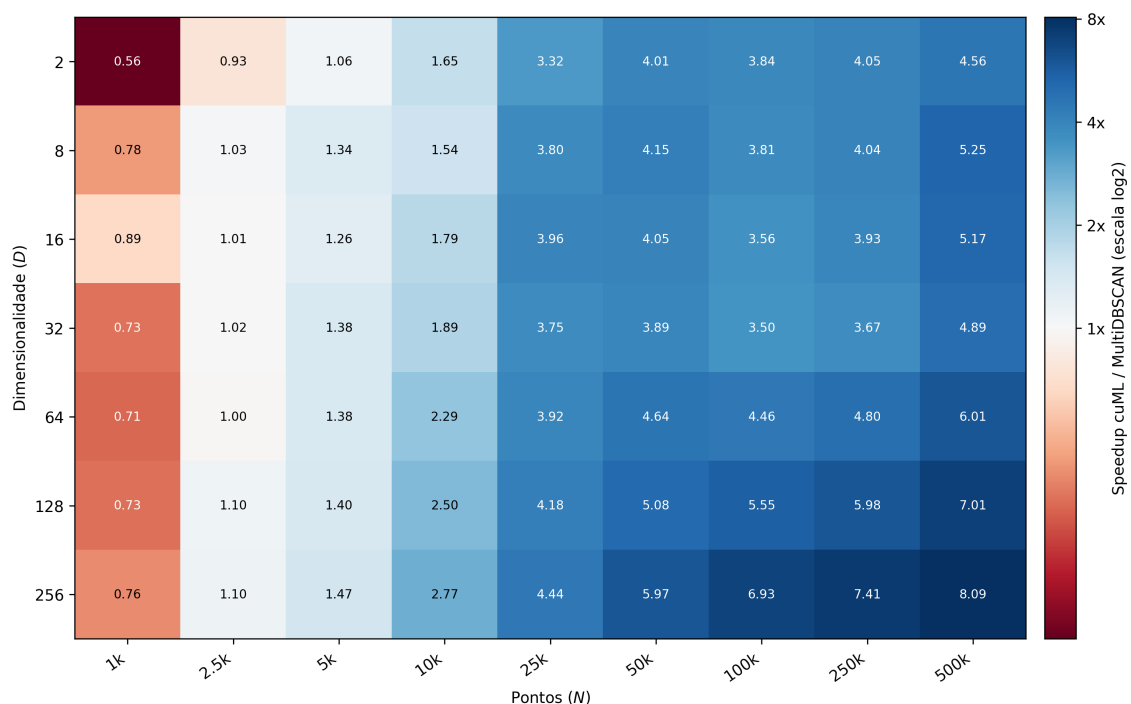


Figura 3. Aceleração de *fit* do MultiDBSCAN em relação à soma de cinco execuções do cuML para combinações com $N \leq 500.000$ e $D \leq 256$. Valores acima de 1,00 (azul) indicam vantagem do MultiDBSCAN, enquanto valores abaixo de 1,00 (vermelho) indicam vantagem do cuML. A escala de cores é logarítmica na base 2, e os valores nas células são as razões medidas.

as configurações disponibilizadas e avaliadas por seus autores contemplam duas ou três dimensões, a comparação com o CUDA-DClust+ foi restrita aos seis conjuntos de baixa dimensionalidade de sua avaliação original.

A Figura 2 apresenta dois recortes de escalabilidade. Na Figura 2a, com $N = 10.000$, o tempo médio do MultiDBSCAN permaneceu abaixo de 40 ms até $D = 512$ e atingiu 52,22 ms em $D = 1024$. No mesmo intervalo, o tempo agregado do cuML aumentou de 50,74 para 191,51 ms. Apesar de pequenas oscilações entre dimensões consecutivas, a aceleração aumentou de $1,65\times$, em $D = 2$, para $3,68\times$, em $D = 1024$.

Na Figura 2b, com $D = 16$, o cuML foi mais rápido em $N = 1.000$, com aceleração de $0,89\times$. Em $N = 2.500$, os métodos ficaram praticamente empatados ($1,01\times$). A partir de $N = 5.000$, o MultiDBSCAN foi mais rápido em todas as configurações apresentadas, alcançando $5,17\times$ em $N = 500.000$ e $7,92\times$ em $N = 1.000.000$. Esses dois últimos casos foram executados uma única vez devido ao custo quadrático e, portanto, representam estimativas pontuais, sem medida de variabilidade. Os demais casos utilizaram um aquecimento e cinco execuções medidas.

A Figura 3 detalha a interação entre N e D . Para $N = 1.000$, o cuML foi mais rápido em todas as dimensões, com razões entre $0,56\times$ e $0,89\times$. Em $N = 2.500$, os tempos ficaram próximos: o cuML manteve vantagem em $D = 2$ ($0,93\times$), houve empate aproximado em $D = 64$ ($1,00\times$), e o MultiDBSCAN foi mais rápido nas demais dimensões. A partir de $N = 5.000$, o MultiDBSCAN superou o cuML em todas as células

apresentadas. A tendência geral de aumento do ganho com N e D culminou em $8,09\times$ para $N = 500.000$ e $D = 256$. Como as células com $N = 500.000$ correspondem a uma única execução, esse máximo deve ser interpretado como uma estimativa pontual.

4.2.1. Impacto da Quantidade de Valores de ε

A comparação anterior utilizou cinco valores de ε . Para avaliar como o desempenho varia com a quantidade de configurações processadas simultaneamente, foi realizado um experimento adicional com $E \in \{1, 3, 5, 9, 17, 25, 50, 75, 100\}$. Utilizou-se um conjunto sintético com $N = 5.000$ pontos, $D = 16$ dimensões e $\min Pts = 5$. Para $E > 1$, os valores de ε foram distribuídos uniformemente no intervalo fechado $[2,8; 3,6]$, incluindo seus extremos. Para $E = 1$, utilizou-se o ponto médio $\varepsilon = 3,2$. Dessa maneira, a faixa analisada permaneceu fixa e somente sua resolução foi modificada.

O MultiDBSCAN processou os E valores em uma única execução, enquanto o tempo do cuML corresponde à soma de E execuções independentes, uma para cada valor de ε . A aceleração foi calculada por

$$S_E = \frac{\sum_{e=1}^E T_{\text{cuML}}(\varepsilon_e)}{T_{\text{MultiDBSCAN}}(E)}. \quad (1)$$

Valores de S_E superiores a 1 indicam vantagem do MultiDBSCAN. Cada configuração foi executada uma vez para aquecimento, seguida de dez execuções medidas. A ordem dos métodos foi alternada entre as repetições, e a Tabela 3 apresenta a média e o desvio padrão dos tempos obtidos.

Tabela 3. Impacto da quantidade de valores de ε no tempo e na memória.

E	$\Delta\varepsilon$	Tempo de fit (s)		Acel.	Custo/config. (s)		Memória (MiB)
		Multi-EPS	cuML (E fits)		Multi-EPS	cuML	
1	–	0,0180 $\pm$ 0,0027	0,0048 $\pm$ 0,0006	0,27 $\pm$ 0,05x	0,01800	0,00483	16,40
3	0,40	0,0183 $\pm$ 0,0025	0,0134 $\pm$ 0,0005	0,73 $\pm$ 0,09x	0,00609	0,00445	16,55
5	0,20	0,0185 $\pm$ 0,0023	0,0223 $\pm$ 0,0012	1,21 $\pm$ 0,18x	0,00369	0,00447	16,71
9	0,10	0,0207 $\pm$ 0,0025	0,0401 $\pm$ 0,0015	1,94 $\pm$ 0,24x	0,00230	0,00445	17,01
17	0,050	0,0210 $\pm$ 0,0023	0,0747 $\pm$ 0,0012	3,57 $\pm$ 0,30x	0,00123	0,00440	17,62
25	0,033	0,0238 $\pm$ 0,0026	0,1070 $\pm$ 0,0027	4,50 $\pm$ 0,41x	0,00095	0,00428	18,23
50	0,016	0,0289 $\pm$ 0,0020	0,2165 $\pm$ 0,0055	7,48 $\pm$ 0,48x	0,00058	0,00433	20,14
75	0,011	0,0354 $\pm$ 0,0018	0,3226 $\pm$ 0,0095	9,10 $\pm$ 0,48x	0,00047	0,00430	22,05
100	0,008	0,0408 $\pm$ 0,0013	0,4234 $\pm$ 0,0119	10,39 $\pm$ 0,38x	0,00041	0,00423	23,95

Notas: Os tempos representam média $\pm$ desvio padrão de 10 execuções. O tempo do cuML é a soma de E execuções independentes.

Para uma única configuração, o cuML apresentou o menor tempo, com 0,004.8 segundos, contra 0,018.0 segundos do MultiDBSCAN. Com três valores de ε , o MultiDBSCAN ainda permaneceu mais lento, atingindo aceleração de $0,73\times$. A primeira quantidade avaliada para a qual a abordagem proposta superou as execuções independentes do cuML foi $E = 5$, com aceleração de $1,21\times$. Portanto, nas condições deste experimento, a configuração com $E = 5$ representa o ponto de equilíbrio observado entre as duas abordagens.

A vantagem aumentou progressivamente com E , alcançando $10,39\times$ para $E = 100$. Enquanto o tempo agregado do cuML cresceu de 0,004.8 para 0,423.4 segundos, o tempo do MultiDBSCAN aumentou de apenas 0,018.0 para 0,040.8 segundos. Esse comportamento ocorre porque o cuML repete todo o DBSCAN para cada valor de ε , enquanto o MultiDBSCAN compartilha o cálculo das distâncias entre todas as configurações.

O compartilhamento também reduziu o custo médio por configuração do MultiDBSCAN de 18,00 milissegundos, em $E = 1$, para 0,41 milissegundos, em $E = 100$. Em contrapartida, o pico de memória estimado aumentou de 16,40 para 23,95 MiB devido às estruturas mantidas separadamente para cada valor de ε . Esse crescimento foi consideravelmente menor que o da quantidade de configurações, pois o ladrilho de distâncias, que representa uma parcela importante da memória, é reutilizado durante toda a execução.

5. Conclusão

Este trabalho apresentou o MultiDBSCAN, uma abordagem em GPU para a avaliação simultânea de múltiplos valores de ε , mantendo $minPts$ fixo. Diferentemente das implementações convencionais, que exigem uma nova execução do algoritmo para cada configuração, a solução proposta reutiliza o cálculo das distâncias entre as configurações e combina o processamento em ladrilhos da matriz de produtos escalares, a contagem de vizinhos por histogramas cumulativos e estruturas Union-Find independentes. Os resultados experimentais para cinco valores de ε demonstraram aceleração média geométrica de $5,09\times$ e $32,78\times$ do MultiDBSCAN em relação, respectivamente, à soma de cinco execuções independentes do cuML e à soma de cinco execuções independentes do CUDA-DClust+. Os agrupamentos produzidos mantiveram elevada concordância com o cuML, com ARI médio superior a 0,98. A escalabilidade também foi demonstrada com o aumento do número de pontos e da dimensionalidade, evidenciando o potencial de ganho em cenários nos quais múltiplas configurações precisam ser avaliadas, como na exploração e no ajuste de hiperparâmetros do DBSCAN. Trabalhos futuros podem investigar o compartilhamento da computação para diferentes valores de $minPts$, reduzir o *overhead* observado em conjuntos de dados menores e explorar formas de adaptação automática dos parâmetros em função das características do conjunto de dados. Outro aspecto a ser investigado é o uso em cenários com densidade heterogênea [Abdulhameed et al. 2024, Fahim 2023], tendo em vista que o DBSCAN pode apresentar limitações ao lidar com densidades distintas devido ao uso de um valor fixo de ε . Esse limiar global pode fazer com que agrupamentos menos densos sejam tratados como ruído.

Agradecimentos e Disponibilidade de Código

FAPEMIG (APQ-01577-22), CNPq, Especialização UFV em Robótica. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES) Código de Financiamento 001. O código-fonte, os scripts de benchmark e os arquivos necessários para reproduzir os experimentos estão disponíveis publicamente em <https://github.com/Morphy999/DBSCANMultiE>.

Referências

- Abdulhameed, T. Z., Yousif, S. A., Samawi, V. W., and Al-Shaikhli, H. I. (2024). Ss-dbscan: semi-supervised density-based spatial clustering of applications with noise for meaningful clustering in diverse density data. *IEEE Access*, 12:131507–131520.
- Andrade, G., Ramos, G., Madeira, D., Sachetto, R., Ferreira, R., and Rocha, L. (2013). G-dbscan: A gpu accelerated algorithm for density-based clustering. *Procedia Computer Science*, 18:369–378.
- Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J. (1999). Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60.
- Barros, O., Freitas, I., Pereira, P. H., Neto, W. d. B., Alcolea, A., Villarroya-Gaudó, M., Resano, J., Nacif, J. A. M., and Ferreira, R. (2026). Rethinking per-thread computation for machine learning design exploration: A work-efficient gpu strategy for k-means and xgboost. *Concurrency and Computation: Practice and Experience*, 38(11):e70744.
- Bueno, W., Silva, O., Nacif, J. A., and Ferreira, R. (2024). Implementação paralela de múltiplos k-means em gpu. In *Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD)*, pages 37–48. SBC.
- Bushra, A. A. and Yi, G. (2021). Comparative analysis review of pioneering dbscan and successive density-based clustering algorithms. *IEEE Access*, 9:87918–87935.
- Böhm, C., Noll, R., Plant, C., and Wackersreuther, B. (2009). Density-based clustering using graphics processors. pages 661–670.
- Çelik, M., Dadaşer-Çelik, F., and Dokuz, A. Ş. (2011). Anomaly detection in temperature data using dbscan algorithm. In *2011 international symposium on innovations in intelligent systems and applications*, pages 91–95. IEEE.
- Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD’96, page 226–231. AAAI Press.
- Fahim, A. (2023). Adaptive density-based spatial clustering of applications with noise (adbscan) for clusters of different densities. *Computers, Materials and Continua*, 75(2):3695–3712.
- Goulart, J. L. d. J. et al. (2024). Árvore de decisão por agrupamento com dbscan aproximativo.
- Gupta, M., Mishra, A., Mishra, A., and Gangrade, J. (2024). Accelerated gpu-based clustering, classification and regression using rapids: A comparative study. In *International Conference on Computation of Artificial Intelligence & Machine Learning*, pages 239–253. Springer.
- Kiran, B. V. S., Kowshik, M. A., Manohar, A. L., Siddardha, D. R., Kavitha, M., and Kavitha, S. (2024). Enhancing accident prediction through integrated knn and dbscan algorithms for superior accuracy. In *2024 8th International Conference on Inventive Systems and Control (ICISC)*, pages 423–428. IEEE.

- Kuo, F.-Y., Wen, T.-H., and Sabel, C. E. (2018). Characterizing diffusion dynamics of disease clustering: a modified space–time dbscan (mst-dbscan) algorithm. *Annals of the American Association of Geographers*, 108(4):1168–1186.
- Poudel, M. and Gowanlock, M. (2021a). Conjuntos de dados do CUDA-DClust+ para o HiPC 2021. https://rcdata.nau.edu/gowanlock_lab/datasets/CUDA_DCLUST_datasets/CUDA_DCLUST_HiPC2021.zip. Acesso em: 15 ago. 2026.
- Poudel, M. and Gowanlock, M. (2021b). Cuda-dclust+: Revisiting early gpu-accelerated dbscan clustering designs. In *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 354–363. IEEE.
- Sarkar, T. (2022). Gpu-based data science for high productivity. In *Productive and Efficient Data Science with Python: With Modularizing, Memory profiles, and Parallel/GPU Processing*, pages 299–326. Springer.
- Thapa, R. J., Trefftz, C., and Wolffe, G. (2010). Memory-efficient implementation of a graphics processor-based cluster detection algorithm for large spatial databases. In *2010 IEEE International Conference on Electro/Information Technology*, pages 1–5. IEEE.
- Zhang, X., Lauber, L., Liu, H., Shi, J., Wu, J., and Pan, Y. (2021). Research on the method of travel area clustering of urban public transport based on sage-husa adaptive filter and improved dbscan algorithm. *PLoS one*, 16(12):e0259472.
- Zhou, S., Xu, H., Zheng, Z., Chen, J., Li, Z., Bu, J., Wu, J., Wang, X., Zhu, W., and Ester, M. (2024). A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions. *ACM Computing Surveys*, 57(3):1–38.