K-means com Seleção Automática de K via Dunn-SkeVa em GPU

Wilson G. N. Junior[1], **Wellington S. Martins**[1], **André C. P. L. F. de Carvalho**[2]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 74690-900 – Goiânia – GO – Brazil

[2]Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)
Caixa Postal 668, 13560-970 – São Carlos – SP – Brazil

wilson2@discente.ufg.br, wsmartins@ufg.br, andre@icmc.usp.br

Abstract. *Choosing the ideal number of clusters K is a central problem in the K-means algorithm. The Dunn Index (DI) is a classical metric for evaluating clustering quality, but its quadratic cost $O(N^2)$ makes it impractical for large datasets. In this work, we propose a parallel GPU implementation of the Dunn Index using SkeVa sampling, which significantly reduces the computational cost of this metric. The solution is integrated into K-means as a pipeline for automatic selection of the best K. Experiments achieved speedups of $42.5\times$ to $92.0\times$ over the serial version with exact computation.*

Resumo. *A escolha do número ideal de clusters K é um problema central no algoritmo K-means. O Índice de Dunn (DI) é uma métrica clássica para avaliar a qualidade de agrupamentos, mas seu custo quadrático $O(N^2)$ inviabiliza sua aplicação em grandes volumes de dados. Neste trabalho, propomos uma implementação paralela em GPU do Índice de Dunn utilizando amostragem SkeVa, que reduz significativamente o custo computacional dessa métrica. A solução é então integrada ao K-means em um pipeline para seleção automática do melhor K de forma eficiente. Em experimentos conseguimos speedups de $42.5\times$ a $92.0\times$ em relação à versão serial com cálculo exato.*

1. Introdução

O agrupamento de dados (*clustering*) é uma das tarefas centrais do aprendizado de máquina não supervisionado, com aplicações em análise de imagens, bioinformática, segmentação de clientes e detecção de anomalias [Jain 2010]. Entre os algoritmos de agrupamento, o K-means [MacQueen 1967] é amplamente adotado pela sua simplicidade e eficiência, mas impõe uma exigência crítica: o número de clusters K deve ser fornecido como entrada. Na prática, K é raramente conhecido a priori. A abordagem padrão para contornar esse problema consiste em definir um intervalo $I = [K_{\min}, K_{\max}]$, executar o K-means para cada valor $K \in I$ e selecionar o K cujo particionamento maximiza algum índice de validação de agrupamentos (*Cluster Validity Index*, CVI). Esse procedimento exige $(K_{\max} - K_{\min} + 1)$ execuções completas do K-means seguidas do cálculo do CVI, tornando a complexidade do CVI um fator determinante para a viabilidade prática do processo.

O Índice de Dunn (DI) [Dunn 1974] é um dos CVIs mais utilizados na literatura: mede a razão entre a menor separação inter-cluster e o maior diâmetro intra-cluster, de modo que valores maiores indicam agrupamentos compactos e bem separados. Entretanto, por conta da comparação de pares de pontos o DI tem complexidade $O(N^2)$. Em revisão recente sobre CVIs para agrupamento automático [Ikotun et al. 2025], os autores confirmam que o DI é computacionalmente inviável para grandes volumes de dados — justamente o cenário onde encontrar K automaticamente traz maior valor prático, e onde o cálculo precisa ser repetido para cada K no intervalo de busca.

Parte desse problema já foi endereçada na literatura, embora de forma isolada. Traganitis et al. [Traganitis et al. 2015] propuseram o SkeVa (*Sketch-and-Validate*), uma técnica de amostragem que estima o diâmetro máximo de um cluster com custo $O(R \cdot S^2)$, onde $S \ll N$ é o tamanho da amostra e R o número de repetições. Ben Ncir et al. [Ncir et al. 2021] combinaram SkeVa com Apache Spark para calcular o DI de forma escalável em ambientes distribuídos, demonstrando resultados precisos para N na ordem de milhões. No entanto, essa abordagem é orientada à paralelização horizontal em CPU, sem aceleração por GPU.

Em trabalho anterior publicado na forma de resumo expandido [Junior and Martins 2026], apresentamos uma versão inicial desta abordagem, demonstrando que o cálculo do DI com amostragem SkeVa pode ser acelerado em GPU com CUDA, obtendo speedups de $9\times$ a $11\times$ em relação à versão serial para *datasets* de até um milhão de pontos. Os resultados promissores obtidos nesse trabalho inicial motivaram o desenvolvimento do presente trabalho, que o estende em duas frentes. Primeiro, o kernel de distâncias pareadas foi otimizado com processamento de múltiplos pares por *thread* e redução via *warp shuffle*, elevando o speedup do pipeline para até $92{,}0\times$ frente à versão serial. Segundo, o método é integrado ao K-means em um pipeline que seleciona automaticamente o melhor K em um intervalo de busca.

O código-fonte da implementação proposta está disponível publicamente em `https://github.com/wilsongn/find-best-k-cuda-skeva`.

O restante do artigo está organizado da seguinte forma: a Seção 2 apresenta os conceitos fundamentais; a Seção 3 discute os trabalhos relacionados; a Seção 4 descreve a proposta e suas otimizações; a Seção 5 apresenta os experimentos e resultados; e a Seção 6 conclui o trabalho.

2. Fundamentação Teórica

2.1. K-means e o Problema da Seleção de K

O K-means [MacQueen 1967] particiona um conjunto de dados $X = \{x_1, \ldots, x_N\} \subset \mathbb{R}^{NF}$ em K clusters $C_1, \ldots, C_K$ minimizando a soma dos quadrados intra-cluster (*Within-Cluster Sum of Squares* – WCSS):

$$\text{WCSS} = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|_2^2,$$

onde $\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$ é o centróide do cluster k. Como o problema de minimizar a WCSS é NP-difícil [Aloise et al. 2009], utiliza-se na prática o algoritmo iterativo de

Lloyd: cada ponto é atribuído ao centróide mais próximo e os centróides são recalculados até a convergência.

Como o resultado depende da escolha dos centróides iniciais, utiliza-se a inicialização K-means++ [Arthur and Vassilvitskii 2007], que seleciona centróides com probabilidade proporcional ao quadrado da distância ao centróide mais próximo já escolhido:

$$P(x_i) = \frac{D(x_i)^2}{\sum_{x \in X} D(x)^2},$$

onde $D(x_i) = \min_{j<k} \|x_i - \mu_j\|$. Na prática, o algoritmo é executado com U reinícios independentes e retém-se o particionamento de menor WCSS.

O parâmetro K deve ser fornecido como entrada, mas raramente é conhecido a priori. Conforme descrito na introdução, a abordagem padrão consiste em executar o K-means para cada K em um intervalo $[K_{\min}, K_{\max}]$ e avaliar cada resultado com um índice de validação de agrupamentos (CVI), tornando a complexidade do CVI um fator crítico para a viabilidade do processo.

2.2. Índice de Dunn

O Índice de Dunn (DI) [Dunn 1974] avalia a qualidade de um particionamento combinando duas propriedades: a separação entre clusters distintos e a compacidade dentro de cada cluster. Formalmente, o DI é definido como:

$$\text{DI} = \frac{\min\limits_{1 \le i < j \le K} \delta(C_i, C_j)}{\max\limits_{1 \le k \le K} \Delta(C_k)},$$

onde $\delta(C_i, C_j)$ é a separação entre os clusters C_i e C_j, e $\Delta(C_k)$ é o diâmetro do cluster C_k. O DI é maximizado quando os clusters são compactos internamente e bem separados entre si. Portanto, quanto maior o valor, melhor o particionamento.

O diâmetro de um cluster é definido como a maior distância euclidiana entre quaisquer dois pontos do cluster:

$$\Delta(C_k) = \max_{x_p, x_q \in C_k} \|x_p - x_q\|_2.$$

O cálculo exato exige $O(|C_k|^2)$ comparações por cluster, totalizando $O(N^2)$. Este é o principal gargalo computacional do DI.

A separação entre dois clusters pode ser calculada de diferentes formas. Na definição original de Dunn [Dunn 1974], utiliza-se a distância mínima ponto-a-ponto:

$$\delta(C_i, C_j) = \min_{x_p \in C_i,\, x_q \in C_j} \|x_p - x_q\|_2,$$

que também tem custo $O(N^2)$. Uma alternativa amplamente utilizada é a distância entre centróides:

$$\delta(C_i, C_j) = \|\mu_i - \mu_j\|_2,$$

onde μ_i e μ_j são os centróides dos clusters C_i e C_j, com custo $O(K^2 \cdot NF)$, onde NF é o número de atributos. Como tipicamente $K \ll N$, esse custo é muito menor que o da definição original.

2.3. Amostragem SkeVa (*Sketch-and-Validate*)

O SkeVa [Traganitis et al. 2015] é um framework de amostragem proposto para acelerar a análise de grandes volumes de dados. Inspirado no princípio RANSAC (*Random Sample Consensus*), originalmente desenvolvido para regressão robusta em visão computacional, o SkeVa opera em duas fases:

Sketch: extrai-se uma amostra reduzida dos dados (em número de instâncias e/ou dimensões) e executa-se a operação de interesse sobre essa amostra.

Validate: utiliza-se uma amostra independente para verificar a consistência do resultado obtido no sketch.

A repetição desse processo por múltiplas rodadas aumenta a confiabilidade da estimativa.

3. Trabalhos Relacionados

3.1. K-means Paralelo em GPU

Diversas implementações eficientes do K-means em GPU foram propostas na literatura. He et al. [He et al. 2022] otimizaram a precisão numérica da fase de atualização de centróides com um método de soma em duas etapas, disponibilizando o código publicamente — utilizado neste trabalho para gerar os particionamentos de entrada para o cálculo do DI. Bueno et al. [Bueno et al. 2024] propuseram o cálculo simultâneo de múltiplos valores de K no mesmo kernel, alcançando speedups de até $140\times$ em relação à RAPIDS cuML [RAPIDS Development Team 2018]. Em ambos os trabalhos, porém, K é fornecido como entrada e nenhuma métrica de validação de agrupamentos é utilizada para selecionar o melhor valor.

3.2. Índice de Dunn Paralelo e Escalável

Ben Ncir et al. [Ncir et al. 2021] propuseram o S-DI (*Scalable Dunn Index*), que adapta o princípio SkeVa [Traganitis et al. 2015] para estimar o diâmetro intra-cluster no cálculo do Índice de Dunn. Em vez de avaliar todos os pares de pontos dentro de cada cluster, o S-DI amostra subconjuntos de tamanho $n \ll |C_k|$ em R repetições, reduzindo a complexidade de $O(|C_k|^2)$ para $O(R \cdot n^2)$ por cluster. Os autores demonstraram convergência para o valor exato com erro inferior a 2% em *datasets* com dezenas a centenas de milhões de pontos. A implementação utiliza Apache Spark para distribuir o cálculo entre múltiplas máquinas, mas não explora aceleração por GPU.

Grün et al. [Grün et al. 2024] aceleraram o cálculo do Índice de Dunn em GPU com CUDA, implementando o cálculo de centróides e o kernel de distâncias pareadas inteiramente na GPU. No entanto, essa implementação não utiliza amostragem, calculando o DI de forma exata, o que limita sua aplicação a *datasets* de tamanho moderado.

Em trabalho anterior [Junior and Martins 2026], combinamos amostragem SkeVa com CUDA para acelerar o cálculo do DI, obtendo speedups de $9\times$ a $11\times$ frente à versão serial exata para *datasets* de até um milhão de pontos. Entretanto, o kernel não incluía otimizações como múltiplos pares por *thread* e redução via *warp shuffle*, o que tornava o cálculo ainda lento demais para ser repetido para cada K em um intervalo de busca. A aplicabilidade na seleção automática do melhor K também não era demonstrada.

4. Proposta

4.1. Visão Geral

A proposta deste trabalho consiste em uma implementação paralela em GPU do Índice de Dunn com amostragem SkeVa, com otimizações de kernel que tornam o cálculo suficientemente rápido para ser aplicado na seleção automática do melhor K para o algoritmo K-means. Para cada valor de K em $[K_{\min}, K_{\max}]$, executa-se o K-means, calcula-se o DI via SkeVa em GPU e registra-se o resultado. Ao final, retorna-se o K de maior índice.

A etapa de agrupamento utiliza a implementação de He et al. [He et al. 2022], que disponibiliza código aberto para K-means em CPU e GPU com precisão numérica controlada. Essa implementação é utilizada sem modificações e gera como saída um arquivo de rótulos para cada valor de K.

O Algoritmo 1 descreve o fluxo completo do pipeline proposto. As subseções seguintes detalham a arquitetura e técnicas aplicadas no algoritmo.

Algoritmo 1: FindBest-K

Input: Dataset X (N pontos, NF atributos), intervalo $[K_{\min}, K_{\max}]$, reinícios U, percentual de Sketch p_S, repetições R
Output: K^* com maior Índice de Dunn estimado

$K^* \leftarrow K_{\min}$;
$DI^* \leftarrow 0$;
para $K \leftarrow K_{\min}$ ***até*** $K_{\max}$ **faça**
  $C_1, \ldots, C_K \leftarrow$ KMeans$(X,\ K,\ U)$
  $\mu_1, \ldots, \mu_K \leftarrow$ CalculaCentroids$(X, C_1, \ldots, C_K)$;
  $\delta_{\min} \leftarrow \min_{i \neq j} \|\mu_i - \mu_j\|_2$
  para $k \leftarrow 1$ ***até*** K **faça**
    $\hat{\Delta}_k \leftarrow 0$;
    para $r \leftarrow 1$ ***até*** R **faça**
      $S^{(r)} \leftarrow$ AmostraAleatoria$(C_k,\ p_S \cdot |C_k|)$;
      Transfere $S^{(r)}$ para GPU;
      $d \leftarrow$ KernelPairwise$(S^{(r)})$
      $\hat{\Delta}_k \leftarrow \max(\hat{\Delta}_k,\ d)$;
    fim
  fim
  $\Delta_{\max} \leftarrow \max_k\ \hat{\Delta}_k$;
  $DI \leftarrow \delta_{\min} / \Delta_{\max}$;
  se $DI > DI^*$ **então**
    $DI^* \leftarrow DI$;
    $K^* \leftarrow K$;
  fim
fim
retorna K^*

4.2. Aplicação do SkeVa ao Índice de Dunn

Conforme descrito na Seção 2, o gargalo computacional do DI está no cálculo do diâmetro intra-cluster $\Delta(C_k)$, que exige a comparação de todos os pares de pontos dentro do cluster. Aplicamos o princípio SkeVa para estimar esse diâmetro sem avaliar todos os pares.

Para cada cluster C_k, o algoritmo executa R repetições independentes. Em cada repetição, amostra-se aleatoriamente S pontos do cluster e calcula-se o diâmetro da amostra na GPU: $\hat{\Delta}_S^{(r)} = \max_{x_p, x_q \in S^{(r)}} \|x_p - x_q\|_2^2$. O estimador do diâmetro do cluster é o máximo entre todas as repetições:

$$\hat{\Delta}_k = \max_{r=1}^{R} \hat{\Delta}_S^{(r)}.$$

A validação ocorre implicitamente pela repetição: ao amostrar subconjuntos independentes em cada rodada e reter apenas o maior diâmetro observado, aumenta-se a probabilidade de capturar os pontos extremos que definem o diâmetro real, seguindo o princípio de Ben Ncir et al. [Ncir et al. 2021]. A complexidade por cluster é reduzida de $O(|C_k|^2)$ para $O(R \cdot S^2)$, com $S \ll |C_k|$.

A separação inter-cluster é calculada como a distância mínima entre centróides na CPU, conforme discutido na Seção 2. Essa substituição da distância exata (ponto-a-ponto) pela distância entre centróides na etapa inter-cluster não é apenas uma conveniência para contornar o gargalo computacional, mas uma escolha arquitetural embasada na forte sinergia geométrica com o algoritmo K-means. Como o K-means minimiza a WCSS baseando-se na distância euclidiana dos pontos até um centro de massa, o algoritmo possui um viés inerente para particionar os dados em clusters convexos e isotrópicos (esféricos ou globulares). Nesse cenário, a distância entre os centróides é altamente correlacionada com a separação real das fronteiras dos grupos. Isso garante que a nossa heurística preserve a semântica de validação do Índice de Dunn, permitindo ao pipeline localizar o K ideal com precisão, mas mantendo a carga computacional pesada da GPU focada exclusivamente na estimação do diâmetro intra-cluster via SkeVa.

Uma decisão arquitetural importante é que o *dataset* completo permanece na memória do *host*. Em [Junior and Martins 2026], o dataset era transferido integralmente para a GPU antes do cálculo, impondo um gargalo de transferência proporcional a N a cada valor de K avaliado. Na implementação atual, apenas os pontos amostrados em cada repetição são enviados, eliminando esse gargalo e permitindo processar datasets maiores que a memória disponível na GPU.

4.3. Múltiplos pares por thread

O parâmetro PPT (*Pairs Per Thread*) define quantos pares de pontos cada *thread* processa por invocação do kernel. Em vez de calcular um único par e escrever imediatamente o resultado em memória compartilhada, cada *thread* itera sobre PPT pares consecutivos mantendo o máximo parcial em registrador durante todo o laço, realizando a escrita apenas uma vez ao final [Volkov 2010]. Como efeito colateral, o número de *threads* lançadas cai por um fator de PPT, reduzindo também a pressão sobre a memória compartilhada na redução. Os resultados dessa otimização isolada são discutidos na Seção 5, Tabela 1.

4.4. Redução hierárquica via *warp shuffle*

Com o máximo local em registrador, a redução ao máximo do bloco ocorre em três passos. Primeiro, as 32 *threads* de cada warp trocam valores via `__shfl_down_sync` com deslocamentos 16, 8, 4, 2 e 1 — sem tocar em memória compartilhada. Depois, só a *lane* 0 de cada warp escreve seu resultado em memória compartilhada, seguida de um único `__syncthreads()`. Por fim, o warp 0 reduz esses valores com mais um passo de *shuffle*. Para $T = 256$ *threads* (8 warps), isso resulta em 8 escritas e 1 sincronização, contra 256 escritas e $\log_2(256) = 8$ sincronizações na versão [Junior and Martins 2026]. A memória compartilhada cai de $T \cdot \texttt{sizeof(float)}$ para $\lceil T/32 \rceil \cdot \texttt{sizeof(float)}$ — de 1024 para 32 bytes.

O Algoritmo 2 descreve o kernel pairwise, que calcula a maior distância ao quadrado entre todos os pares de pontos de uma amostra. O kernel combina as duas otimizações descritas anteriormente: cada thread processa PPT pares consecutivos acumulando o máximo em registrador, e a redução ao resultado final do bloco é feita em três níveis — primeiro dentro de cada warp via shuffle, depois entre warps via memória compartilhada, e por fim entre blocos na CPU.

Algoritmo 2: KernelPairwise($S^{(r)}$)

Input: Amostra $S^{(r)}$ com n pontos e NF atributos
Output: Maior distância ao quadrado entre pares de pontos
`/* Cada thread processa PPT pares consecutivos */`
para cada *thread* t *em paralelo* **faça**
  $local\_max \leftarrow 0$;
  para $k \leftarrow 0$ ***até*** $PPT - 1$ **faça**
    $(i, j) \leftarrow$ par atribuído a esta thread;
    $dist \leftarrow \sum_{f=0}^{NF-1}(S_i^{(r)}[f] - S_j^{(r)}[f])^2$;
    $local\_max \leftarrow \max(local\_max,\ dist)$;
  fim
  `/* Redução intra-warp (32 threads) via shuffle */`
  $warp\_max \leftarrow$ ReducaoWarp($local\_max$);
fim
`/* Lane 0 de cada warp escreve em memória compartilhada */`
$sdata[warpId] \leftarrow warp\_max$;
Sincronizar threads do bloco;
`/* Warp 0 reduz os valores dos warps via shuffle */`
$block\_max \leftarrow$ ReducaoWarp($sdata$);
`/* Thread 0 escreve o resultado do bloco */`
$blockMax[blockIdx] \leftarrow block\_max$;
retorna $\max(blockMax)$; `/* redução final na CPU */`

4.5. Integração com o K-means

A integração entre o K-means de He et al. [He et al. 2022] e o cálculo do DI é realizada por um *script* que automatiza o processo para cada valor de K. Para cada $K \in [K_{\min}, K_{\max}]$, o *script*:

1. Configura os parâmetros do K-means (N, NF, K, caminho do *dataset*) no arquivo de cabeçalho `main.h`;
2. Compila e executa o K-means, que gera o arquivo `Labels.txt` com os rótulos de cada ponto;
3. Executa o programa `dunn_cuda_skeva_optimized` passando o *dataset* original e o arquivo de rótulos como entrada;
4. Registra o valor do DI e o tempo de execução.

Ao final, o *script* identifica o K^* com maior DI e salva os rótulos correspondentes. Essa abordagem mantém os dois componentes desacoplados: o K-means pode ser substituído por qualquer algoritmo de agrupamento que produza um arquivo de rótulos no mesmo formato.

5. Experimentos e Resultados

5.1. Configuração Experimental

Os experimentos foram realizados em uma máquina com GPU NVIDIA RTX 3050 6 GB, Intel Core i7-13620H e 8 GB de RAM. Os datasets sintéticos, com clusters gaussianos esféricos bem separados, foram gerados com o utilitário de geração de dados disponibilizado por He et al. [He et al. 2022]. Os datasets reais são descritos nas subseções seguintes. Salvo indicação contrária, os parâmetros do SkeVa foram fixados em $p_S = 30\%$ e $R = 8$, conforme determinado na Seção 5.3.

5.2. Número de Pares por Thread

Para avaliar o impacto do número de pares processados por *thread*, fixamos $N = 100.000$, $K = 5$, $NF = 4$, 30% do dataset no Sketch, 8 repetições e variamos o parâmetro PPT de 1 a 32. Os resultados são apresentados na Tabela 1.

Tabela 1. Impacto do número de pares por *thread* (PPT) no tempo e speedup

PPT	Tempo (s)	Speed.$_{\text{PPT}}$	Speed.$_{\text{serial}}$
Serial	12,152	—	1,00×
1 (base)	0,389	1,00×	31,2×
2	0,355	1,10×	34,2×
4	0,353	1,10×	34,4×
8	0,346	1,12×	35,1×
16	0,342	1,14×	35,5×
32	**0,339**	**1,15×**	**35,8×**

Com PPT=1, o kernel já alcança $31{,}2\times$ sobre o serial. Aumentar PPT melhora pouco: PPT=32 chega a $35{,}8\times$, um ganho de $1{,}15\times$ sobre a versão base. O kernel é *memory-bound* e mais aritmética por *thread* não resolve o gargalo de memória. O valor PPT=32 foi adotado nos demais experimentos por reduzir a pressão sobre a memória compartilhada na fase de redução.

5.3. Sensibilidade dos Parâmetros SkeVa

Para avaliar o impacto dos parâmetros de amostragem na qualidade da estimativa do diâmetro intra-cluster, utilizamos o dataset Dry Bean [Koklu and Ozkan 2020] ($N =$

13.611, $NF = 16$, $K = 7$), um dataset real com clusters de geometria não-esférica que evita o cenário favorável de clusters perfeitamente esféricos, onde qualquer amostra aleatória capturaria os pontos extremos com alta probabilidade. O valor de referência $DI_{\text{ref}} = 0{,}120586$ foi obtido com 100% dos pontos de cada cluster e utilizando centróides. Foram feitas 10 execuções cada uma com dados aleatórios e ao final foi calculada a média do DI estimado.

Tabela 2. Efeito do número de repetições

Repetições	**$T_{méd}$ (s)**	$\overline{DI}$	**Erro (%)**
1	0,0043	0,167922	39,26
2	0,0070	0,131497	9,05
4	0,0131	0,131252	8,84
8	**0,0223**	**0,121880**	**1,07**
16	0,0461	0,121010	0,35

Com 8 repetições, o erro cai para 1,07%. O ganho adicional de 16 repetições (erro de 0,35%) não justifica o custo computacional duplicado. Adotamos o número de repetições 8 como valor padrão.

Tabela 3. Efeito do percentual de Sketch

% Sketch	**$T_{méd}$ (s)**	$\overline{DI} \pm \sigma$	**Erro (%)**
10%	0,0109	$0{,}151272 \pm 0{,}027777$	25,45
20%	0,0157	$0{,}131553 \pm 0{,}019047$	9,09
30%	**0,0227**	$\mathbf{0{,}121880 \pm 0{,}001060}$	**1,07**
40%	0,0352	$0{,}121240 \pm 0{,}001001$	0,54
50%	0,0476	$0{,}120790 \pm 0{,}000613$	0,17

A Tabela 3 mostra os resultados variando p_S de 10% a 50% com 8 repetições. Com 10%, o erro de 25,45% mostra que a amostra é insuficiente para cobrir a geometria dos clusters. Em 30%, o erro cai para 1,07% com $\sigma = 0{,}001$. Aumentar para 40% ou 50% reduz o erro para 0,54% e 0,17%, mas exige 1,5× e 2,1× mais tempo. Adotamos $p_S = 30\%$ nos demais experimentos para termos um equilíbrio entre precisão e tempo de execução.

5.4. Speedup do Pipeline e Corretude da Seleção de K

As Tabelas 4 e 5 apresentam os resultados do pipeline completo em 8 configurações de dataset, variando entre dados sintéticos e reais. O speedup cresce com o intervalo de busca: no dataset Pendigits com $K \in [5, 15]$, chega a $92\times$; nos datasets com intervalos curtos, fica entre $42{,}5\times$ e $57\times$.

Tabela 4. Speedup do pipeline completo por dataset ($p_S = 30\%$, $R = 8$, PPT = 32).

Dataset	N	NF	K_{real}	$[K_{\min}, K_{\max}]$	T ser. (s)	T GPU (s)	Speedup
Sintético	400.000	4	4	$[2, 5]$	786,8	16,5	47,6×
Sintético	150.000	4	3	$[2, 5]$	110,4	2,6	42,5×
Sintético	150.000	4	5	$[4, 7]$	109,4	2,3	47,6×
Sintético	140.000	4	7	$[5, 10]$	143,1	1,7	82,2×
Sintético	100.000	4	5	$[3, 11]$	109,8	1,3	84,5×
Dry Bean	13.611	16	7	$[2, 10]$	5,5	0,1	55,0×
Shuttle	58.000	9	7	$[5, 9]$	30,8	0,5	57,0×
Pendigits	10.992	16	10	$[5, 15]$	4,6	0,1	92,0×

Nos sintéticos com clusters bem separados, GPU e serial concordam e acertam o K real em todos os casos. Nos datasets reais e no sintético com clusters sobrepostos, ambas as versões apresentam dificuldade, o que reflete limitações conhecidas do DI com separação por centróides em clusters não-esféricos [Saitta et al. 2007, Ikotun et al. 2025]. GPU e serial concordam em 5 dos 8 casos, indicando que a aproximação SkeVa não altera a escolha de K^* na maioria dos cenários.

Tabela 5. Corretude da seleção de K^* pelo pipeline.

Dataset	K_{real}	K^*_{serial}	K^*_{GPU}	Acordo	Correto serial	Correto GPU
Sintético	4	4	4	✓	✓	✓
Sintético	3	3	3	✓	✓	✓
Sintético	5	5	5	✓	✓	✓
Sintético	7	7	7	✓	✓	✓
Sintético	5	3	6	×	×	×
Dry Bean	7	3	3	✓	×	×
Shuttle	7	7	9	×	✓	×
Pendigits	10	14	9	×	×	×

Para fins de comparação entre duas configurações paralelas em GPU, avaliamos a configuração A ($p_S = 30\%$, $R = 8$), utilizada no pipeline principal, e a configuração B ($p_S = 100\%$, $R = 1$, sem centróides). Os resultados da configuração B são equivalentes ao cálculo exato do Dunn Index e alcançam os mesmos K^* que a versão serial, porém, com tempo de execução superior ao da configuração A. A Tabela 6 compara os tempos totais de cálculo do DI para cada dataset. A configuração A supera a configuração B em todos os cenários avaliados, com speedups de 3,7× a 9,0×, demonstrando que a amostragem SkeVa reduz substancialmente o custo computacional do DI em relação à solução que utiliza todos os pontos do dataset para realizar os cálculos.

Tabela 6. Comparação de tempo de execução (s) entre A e B.

Dataset	N	**T A (s)**	**T B (s)**	**Speedup**
Sintético	400.000	16,5	68,0	4,1×
Sintético	150.000	2,6	9,6	3,7×
Sintético	150.000	2,3	9,2	4,0×
Sintético	140.000	1,7	8,8	5,1×
Sintético	100.000	1,3	6,9	5,3×
Dry Bean	13.611	0,1	0,6	5,6×
Shuttle	58.000	0,5	2,2	4,1×
Pendigits	10.992	0,1	0,5	9,0×

6. Conclusão

Este trabalho propôs um pipeline para seleção automática do número de clusters K no K-means com aceleração em GPU. A principal contribuição é o cálculo paralelo do Índice de Dunn via amostragem SkeVa em CUDA, otimizado por duas técnicas de kernel: acumulação do máximo local em registrador com múltiplos pares por *thread*, e redução hierárquica via *warp shuffle*.

Os experimentos mostraram speedups de $42{,}5\times$ a $92{,}0\times$ no pipeline completo frente à versão serial com DI exato. Nos datasets sintéticos com clusters bem separados, GPU e serial concordaram e acertaram o K real em 4 dos 5 casos. O caso de falha ocorreu num dataset com clusters sobrepostos, onde ambas as versões erraram, indicando uma limitação do próprio DI e não da aproximação SkeVa. Nos datasets reais, ambas as versões erraram na maioria dos casos, comportamento consistente com limitações conhecidas do DI com separação por centróides em clusters não-esféricos [Saitta et al. 2007, Ikotun et al. 2025]. GPU e serial concordaram em 5 dos 8 casos no total, mostrando que a aproximação SkeVa não altera a escolha de K^* na maioria dos cenários.

Como trabalho futuro, destaca-se a integração de outras métricas de validação aceleradas em GPU que apresentem melhor desempenho na identificação do K ideal em datasets com clusters não-esféricos, como o Silhouette [Rousseeuw 1987] ou o Davies–Bouldin Index [Davies and Bouldin 1979], mantendo a arquitetura do pipeline proposto.

Referências

Aloise, D., Deshpande, A., Hansen, P., and Popat, P. (2009). Np-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248.

Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics.

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)*.

Davies, D. L. and Bouldin, D. W. (1979). A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):224–227.

Dunn, J. C. (1974). Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104.

Grün, E. S., Martins, W. S., and Franco, R. (2024). Acelerando o cálculo do índice dunn de validação de agrupamento. In *Anais do Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD)*, pages 1–3. SBC.

He, G., Vialle, S., and Baboulin, M. (2022). Parallel and accurate k-means algorithm on CPU-GPU architectures for spectral clustering. *Concurrency and Computation: Practice and Experience*, 34(14):e6621.

Ikotun, A. M., Habyarimana, F., and Ezugwu, A. E. (2025). Cluster validity indices for automatic clustering: A comprehensive review. *Heliyon*, 11(2):e41953.

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666.

Junior, W. G. N. and Martins, W. S. (2026). Uso de GPUs na validação de agrupamentos com amostragem SkeVa. In *Escola Regional de Alto Desempenho do Centro-Oeste (ERAD-CO)*. SBC.

Koklu, M. and Ozkan, I. A. (2020). Multiclass classification of dry beans using computer vision and machine learning techniques. *Computers and Electronics in Agriculture*, 174:105507.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297.

Ncir, C.-E. B., Hamza, A., and Bouaguel, W. (2021). Parallel and scalable Dunn index for the validation of big data clusters. *Parallel Computing*, 102:102751.

RAPIDS Development Team (2018). RAPIDS: Collection of libraries for end to end GPU data science. `https://rapids.ai`.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

Saitta, S., Raphael, B., and Smith, I. F. C. (2007). A comprehensive validity index for clustering. *Intelligent Data Analysis*, 11(6):529–548.

Traganitis, P. A., Slavakis, K., and Giannakis, G. B. (2015). Sketch and validate for big data clustering. *IEEE Journal of Selected Topics in Signal Processing*, 9(4):678–690.

Volkov, V. (2010). Better performance at lower occupancy. In *GPU Technology Conference (GTC)*, volume 10, page 16.