

Explicabilidade para Kmeans com árvores de decisão em GPU

Isabela de Castro Freitas, Pedro Henrique Coura Pereira, Olavo Barros,
José A.M. Nacif, Ricardo Ferreira

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

{isabela.castro.freitas, pedro.coura, olavo.barros, jnacif, ricardo}@ufv.br

Abstract. *Unsupervised algorithms such as K-means cluster data but lack explainability, hindering human analysis in critical decisions. Although decision trees like those built by the ExGreedy algorithm provide explainability for these clusters, computational cost remains a bottleneck at scale. This paper proposes a hybrid CPU/GPU implementation to accelerate ExGreedy. We show that, beyond theoretical complexity, code-level optimizations and algorithmic restructuring are essential. In the improved sequential version, child nodes inherit pre-filtered orderings from parent nodes. Furthermore, independent and repetitive tasks were offloaded to a global preprocessing stage on the GPU. Experiments on seven datasets confirmed the correctness of the results and achieved speedups between $3.18\times$ and $8.58\times$ on medium- and large-scale data, overcoming the GPU initialization overhead seen in smaller datasets.*

Resumo. *Algoritmos não supervisionados como o K-means agrupam dados, mas não fornecem explicabilidade, o que dificulta a análise humana em decisões críticas. Embora árvores de decisão como as do algoritmo ExGreedy permitam explicitar esses agrupamentos, o custo computacional permanece um gargalo em larga escala. Este artigo propõe uma implementação híbrida CPU/GPU para acelerar o ExGreedy. Mostra-se que, além da complexidade teórica, otimizações em nível de código e reestruturação algorítmica são essenciais. Na versão sequencial aprimorada, os nós filhos herdam a ordenação pré-filtrada dos nós pais. Em seguida, adaptou-se a execução transferindo tarefas repetitivas e independentes para um pré-processamento global na GPU. Experimentos em sete conjuntos de dados confirmaram a corretude dos resultados e alcançaram ganhos de desempenho entre $3,18\times$ e $8,58\times$ em dados de média e grande escala, superando a sobrecarga da inicialização da GPU observada em conjuntos de dados menores.*

1. Introdução

Os algoritmos de aprendizado não supervisionado são amplamente utilizados para encontrar padrões em conjuntos de dados. Entre eles estão os métodos de clusterização, que dividem as amostras em grupos com base em distância, densidade ou outros critérios que variam conforme o algoritmo e a área de aplicação. Esses métodos têm sido fundamentais em áreas como educação, publicidade, medicina, biologia e bioinformática [Ezugwu et al. 2022]. No entanto, como esses algoritmos servem de base para decisões de grande impacto social ou financeiro, a dificuldade de humanos analisarem

e entenderem seus resultados se torna um problema, de modo a comprometer tanto a compreensão dos motivos que levaram a um determinado agrupamento quanto a verificabilidade da qualidade desses resultados.

Diante dessa necessidade de facilitar o entendimento humano sobre as partições geradas pela clusterização, uma nova área de pesquisa vem ganhando espaço na comunidade científica: a explicabilidade de modelos de aprendizado de máquina ou *machine learning*. Para o método K-means [MacQueen 1967], por exemplo, uma abordagem frequentemente utilizada é a construção de árvores de decisão [Gabidolla and Carreira-Perpiñán 2022, Hu et al. 2025, Hu et al. 2026] que aproximam, de forma explicável, a partição obtida pelo algoritmo. Essa técnica foi introduzida por [Moshkovitz et al. 2020] e vem sendo aprimorada por diversos trabalhos, como o de [Laber and Murtinho 2021], que propõe o algoritmo ExGreedy para construir árvores explicáveis a partir de uma partição já obtida via K-means.

Apesar dos avanços em qualidade da explicação e das tentativas de muitos trabalhos em minimizar a complexidade dos algoritmos propostos, o custo computacional desses algoritmos [Penha et al. 2018, Penha et al. 2023, Bueno et al. 2024] ainda representa um gargalo para conjuntos de dados volumosos, que são frequentes em situações reais. Por isso, neste trabalho, propomos uma implementação híbrida (CPU-GPU) do algoritmo ExGreedy de [Laber and Murtinho 2021], adaptando partes do algoritmo original para viabilizar sua execução em arquiteturas paralelas.

Embora a análise e redução da complexidade assintótica via notação $O(n)$, como apresentada pelo algoritmo de Exgreedy [Laber and Murtinho 2021], permaneça fundamental para mensurar a escalabilidade teórica dos algoritmos, ela é insuficiente para prever o desempenho real em arquiteturas modernas. O algoritmo EXGreedy tem complexidade $O(ndkH + nd \log n)$, onde n é número de amostras, d é número de dimensões, H é a profundidade da árvore resultante.

Entretanto, com o fim da era da miniaturização dos semicondutores, os ganhos computacionais passaram a depender da otimização no topo da pilha tecnológica: *software*, algoritmos e arquitetura [Leiserson et al. 2020]. Nesse cenário, para gerar código eficiente, é preciso ir além do número bruto de operações e considerar aspectos como o reuso de dados na hierarquia de memória e os custos heterogêneos das operações em hardware (como o gargalo de acesso à memória em comparação à alta capacidade computacional de aceleradores como GPUs). Dessa forma, deve-se aplicar a reengenharia do software e a sua adequação às estruturas suportadas pelo *hardware* para superar os gargalos computacionais.

Para tanto, o algoritmo original do ExGreedy, disponibilizado em Python e C, foi primeiramente traduzido para uma implementação sequencial de referência inteiramente em C. Em seguida, essa versão foi analisada e reestruturada em uma segunda versão sequencial, otimizada, que elimina reordenações redundantes entre os nós da árvore. A partir dessa versão otimizada, desenvolvemos então a implementação híbrida, na qual as etapas de cálculo de distâncias e ordenação são delegadas à GPU. Comparamos as três versões (sequencial traduzida, sequencial otimizada e híbrida) tanto em termos de equivalência entre as partições geradas quanto em tempo de execução, com o objetivo de tornar a geração de explicabilidade para o K-means mais rápida sem comprometer sua

qualidade.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta os fundamentos teóricos necessários à compreensão do trabalho, abordando os conceitos de K-means, interpretabilidade e explicabilidade, além dos princípios de computação paralela em GPU. A Seção 3 discute os trabalhos relacionados, situando o algoritmo Ex-Greedy no contexto da literatura sobre explicabilidade em métodos de clusterização. A Seção 4 detalha a metodologia adotada, descrevendo as etapas de tradução, otimização sequencial e implementação híbrida CPU-GPU do algoritmo. A Seção 5 apresenta e discute os resultados experimentais obtidos, incluindo a avaliação de desempenho e de corretude das árvores geradas. Por fim, a Seção 6 apresenta as conclusões do trabalho e discute direções para trabalhos futuros.

2. Fundamentos

2.1. K-means, interpretabilidade e explicabilidade

O algoritmo K-means consiste em um método de particionamento de dados em K grupos distintos, utilizando a distância euclidiana como critério de atribuição. Cada amostra pertence ao grupo cujo centroide, ponto central que representa o grupo, está mais próximo. As posições dos centroides são ajustadas iterativamente até que a partição se estabilize. Como o cálculo de distância euclidiana é realizado sobre todas as dimensões do conjunto de dados simultaneamente, quanto maior a dimensionalidade dos dados, mais difícil se torna, para um humano, compreender intuitivamente por que uma amostra específica foi atribuída a um determinado grupo, ainda que o resultado seja matematicamente correto.

Segundo a definição proposta por [Rudin 2019], um modelo é dito interpretável quando sua lógica de decisão é transparente e compreensível por humanos durante o próprio processo do algoritmo; de forma complementar, um método fornece *explicabilidade* quando produz, a partir do resultado de um modelo não interpretável, uma aproximação compreensível do processo que levou àquele resultado. O K-means se classifica como interpretável apenas quando a dimensionalidade dos dados é baixa o suficiente para permitir análise direta; para conjuntos de dados de alta dimensionalidade, o algoritmo se torna, na prática, não interpretável, destacando a importância de métodos que forneçam explicabilidade por meio de estruturas auxiliares compreensíveis, como árvores de decisão [Hu et al. 2025, Hu et al. 2026].

2.2. Computação paralela em GPU

Unidades de processamento gráfico (GPUs) possuem arquitetura massivamente paralela e são capazes de executar milhares de *threads* simultaneamente sobre diferentes dados, seguindo o mesmo fluxo de operações. Plataformas como o CUDA permitem delegar trechos específicos de um programa à GPU, enquanto o restante do código permanece sendo executado na CPU, sendo necessária transferência explícita de dados entre as duas memórias. Este modelo de programa é particularmente proveitoso para instruções que podem ser separadas em cálculos independentes e repetitivos para grandes volumes de dados. Contudo, o custo fixo de inicialização de *kernels* e sincronização entre CPU e GPU pode superar os ganhos obtidos quando o volume de trabalho delegado à GPU é pequeno [Cheng et al. 2014, Ferreira et al. 2019, Ferreira 2025].

3. Revisão de literatura

A explicabilidade em métodos de clusterização tem se tornado um tema relevante para a literatura, isso se deve em parte às críticas à utilização de modelos de *machine learning* caixa-preta para a tomada de decisões de alto risco ou importância e pela defesa da utilização de modelos interpretáveis em vez desses, sempre que possível [Rudin 2019]. Contudo, alguns métodos considerados não interpretáveis ainda possuem grande utilidade em diversas áreas da ciência, de forma que há grande interesse em desenvolver métodos que garantam a eles explicabilidade [Hu et al. 2025, Hu et al. 2026]

Uma das abordagens existentes para fornecer explicabilidade para o algoritmo K-means consiste na construção de árvores de decisão [Costa and Pedreira 2023], cujos cortes são escolhidos levando em consideração as posições dos centroides obtidos durante a execução prévia do K-means [Moshkovitz et al. 2020]. Esta abordagem é trabalhada e adaptada por diversos trabalhos [Hu et al. 2025], como para permitir que a árvore tenha um número maior de folhas em relação ao permitido pelo algoritmo original, com objetivo de reduzir o custo de corte [Frost et al. 2020].

O ExGreedy, algoritmo base deste artigo, foi proposto por [Laber and Murtinho 2021] como uma abordagem de construção de árvores de decisão para tornar explicáveis os *clusters* gerados pelo K-means. Essa abordagem foi posteriormente aprofundada e reorganizada em [Murtinho 2025], cujo repositório de código, utilizado neste trabalho, unifica o ExGreedy a uma variante com penalidade crescente conforme a profundidade da árvore, denominada ExShallow e proposta anteriormente pelos mesmos autores [Laber et al. 2023]. Nesse repositório, o ExGreedy corresponde ao caso em que essa penalidade (*depth factor*) é desativada. Essa estratégia de penalização, contudo, não é o foco deste trabalho.

Por fim, a utilização de GPU para aceleração de diversos campos da computação vem ganhando espaço crescente na computação de alto desempenho [Owens et al. 2008]. A técnica de paralelizar algoritmos originalmente sequenciais para que sejam executados em GPU é muito aplicada na área de *machine learning* [Pan and Mishra 2022], por exemplo. Para este trabalho objetivamos paralelizar o método que garante explicabilidade do K-means ao construir as árvores e não é nosso foco a paralelização do K-means em si, que possui várias implementações em GPU [Li et al. 2013, He et al. 2025]

4. Metodologia

A metodologia proposta neste trabalho fundamenta-se na adaptação e paralelização do algoritmo ExGreedy [Laber and Murtinho 2021] para arquiteturas baseadas em GPU. Para isso, o desenvolvimento foi dividido em quatro etapas principais: A tradução do algoritmo original para uma implementação sequencial de *baseline* em C, a análise dessa implementação para identificação de trechos passíveis de paralelização, da qual resultou uma segunda versão sequencial, otimizada, o projeto da arquitetura híbrida em CUDA, construída a partir dessa versão otimizada e a avaliação experimental do desempenho computacional.

Antes da adaptação para GPU, foi preciso preparar uma versão de referência adequada para a comparação de desempenho. O código original do ExGreedy [Murtinho 2025] é dividido em arquivos em duas linguagens. O arquivo contendo

a lógica de avaliação dos cortes (`best_cut`) foi originalmente implementado em C, e, os demais arquivos, com a lógica de construção e estrutura da árvore (`shallow_tree`) e estruturas auxiliares, implementados em Python. Como o objetivo deste trabalho é avaliar o ganho de desempenho da paralelização em GPU, uma comparação entre a implementação em Python e a versão em CUDA/C não seria adequada, visto que a diferença de desempenho entre as duas linguagens de programação é amplamente reconhecida. Por isso, os arquivos em Python foram traduzidos para C e integrados ao módulo `best_cut` original, resultando em uma implementação sequencial única e inteiramente em C (`tree_full.c`), utilizada como *baseline* para todas as comparações de tempo de execução apresentadas neste trabalho. Essa tradução preservou a lógica original do algoritmo, sem alterações nos critérios de decisão de corte ou na ordem das operações.

A partir dessa versão traduzida, foi realizada uma análise do algoritmo com o objetivo de identificar quais de suas operações apresentam potencial de paralelização e quais possuem dependência sequencial. O código constrói a árvore de decisão de forma recursiva, de modo que, em cada nó, avalia-se, para cada uma das dimensões do conjunto de dados, o melhor ponto de corte que separa os pontos e os centroides válidos naquele nó através de uma varredura (*sweep line*), que depende do acesso aos dados e centroides ordenados de forma crescente por dimensão, e também da ordenação dos centroides por distância a cada amostra. Na implementação traduzida, essas ordenações são recalculadas a cada nó da árvore, por meio de `qsort`, resultando em um custo computacional e de tempo que cresce com a profundidade da árvore e com o número de dimensões.

A partir dessa análise, notamos que a etapa de ordenação, recalculada a cada nó consistia em uma operação que poderia ser realizada previamente à construção da árvore e reaproveitada por cada nó, visto que cada nó filho possui apenas um subconjunto já ordenado das amostras e centroides de seu nó pai. Essa reestruturação é particularmente proveitosa para viabilizar a versão paralela, uma vez que como cada chamada de *kernel* na GPU envolve custos fixos de lançamento e sincronização, seria pouco eficiente reordenar os dados a cada nó da árvore por meio de sucessivas chamadas, sendo mais adequado concentrar essas ordenações em uma única etapa anterior à construção da árvore. Para isolar esse ganho puramente algorítmico do ganho a ser obtido especificamente pelo uso da GPU, elaboramos, ainda em CPU, uma segunda versão sequencial, denominada *sequencial otimizado*. Nessa versão, a construção da árvore foi reestruturada de um percurso recursivo em profundidade para um percurso iterativo em largura (BFS), no qual cada nó filho herda do nó pai as ordenações já filtradas, contendo apenas as amostras e centroides que atingem aquele nó após o corte, dispensando assim novas chamadas a `qsort` a cada nível da árvore. Essa reestruturação não altera nenhum critério de decisão do algoritmo original, apenas a forma como as ordenações são realizadas e reaproveitadas, e serve como uma segunda *baseline*, permitindo distinguir, na avaliação experimental, o quanto do tempo economizado pela versão paralela se deve à reestruturação algorítmica e o quanto se deve especificamente ao uso da GPU.

Partindo da estrutura já estabelecida pela versão sequencial otimizada, em particular do mecanismo de herança de ordenações entre nós pais e filhos, foi implementado um código que divide as tarefas do algoritmo entre CPU e GPU, disponibilizado publicamente em repositório próprio [Freitas 2025b], com os devidos créditos e *link* para o repositório original do ExGreedy [Murtinho 2025].

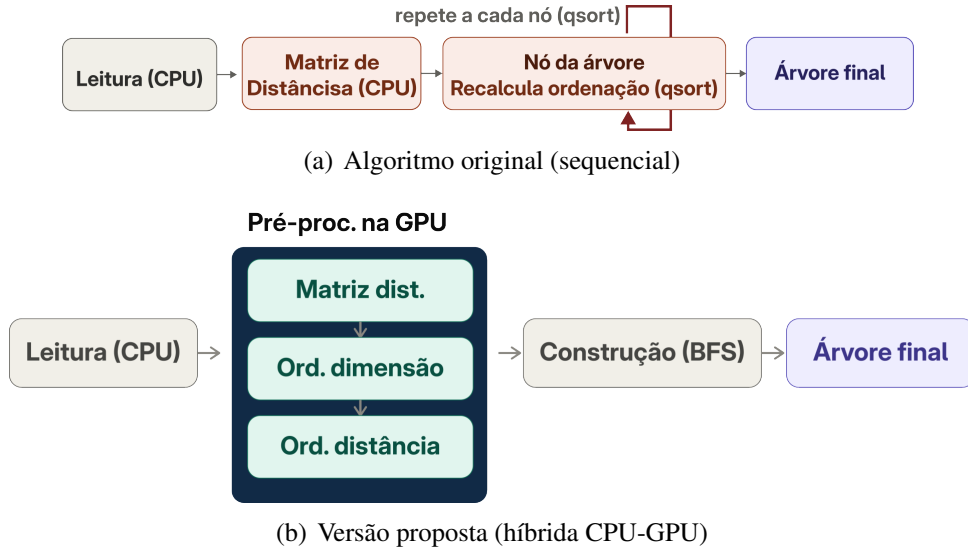


Figura 1. Comparação entre o fluxo do algoritmo original e o fluxo da versão híbrida proposta.

A Figura 1 compara os fluxos. Na versão original (Figura 1(a)), as ordenações são refeitas via `qsort` a cada nó, com custo $O(D(N \log N + K \log K))$, o principal gargalo. Na híbrida (Figura 1(b)), elas são feitas uma única vez na GPU e cada nó filho apenas filtra as do pai.

O pré-processamento na GPU ocorre em duas etapas, detalhadas na Figura 1(b). Na primeira, o cálculo da matriz de distâncias entre os N pontos e os K centroides é feito por um *kernel* CUDA em que cada *thread* processa um único par (ponto, centroide). Na segunda, os K centroides de cada ponto são ordenados por distância em uma única chamada de *segmented sort* da CUB [Merrill and NVIDIA Corporation 2025], e os N pontos de cada dimensão são ordenados com `sort_by_key` da Thrust, uma dimensão por vez. A construção da árvore em si, incluindo a avaliação sequencial dos cortes, permanece idêntica entre as versões, alterando-se apenas a origem (CPU ou GPU) das ordenações que ela consome.

O Algoritmo 1 resume a versão híbrida. A versão sequencial otimizada difere apenas nas linhas 2 a 6, que são executadas na CPU com `qsort`; a construção da árvore (linhas 7 a 24) é a mesma. Nos filhos, a operação `FILTRA` apenas descarta, na ordem já existente, os elementos que não pertencem ao filho, sem reordenar.

Seja N o número de amostras, D o de dimensões e K o de centroides. A varredura de cortes custa $O(N + K)$ por dimensão em cada nó. Na versão traduzida, as ordenações são refeitas em cada nó, com custo $O(D(N \log N + K \log K))$ por nó. Na otimizada, elas são feitas uma única vez, com custo total $O(DN \log N + NK \log K)$, e cada nó apenas filtra as ordenações do pai em $O(N)$. Na híbrida, esse pré-processamento é executado na GPU. Essa caracterização é informal, pois o desempenho real da GPU depende de fatores como ocupação e latência de memória.

Por fim, a avaliação experimental foi conduzida no ambiente Google Colab, versão gratuita de Jupyter Notebooks [Perkel 2018] disponibilizada pela Google, utilizando a CPU do ambiente para as versões sequenciais e na GPU NVIDIA Tesla T4 para a versão

Algorithm 1 ExGreedy híbrido CPU-GPU: pré-processamento e construção da árvore.

Entrada: Amostras X ($N \times D$), centroides S ($K \times D$)

Saída: Árvore de decisão T

```

1: // Pré-processamento (executado uma única vez)
2:  $dist \leftarrow \text{kernel CUDA: uma thread por par (amostra, centroide)}$  [GPU]
3:  $ord\_dist \leftarrow \text{segmented sort (CUB) de } dist, \text{ um segmento por amostra}$  [GPU]
4: para cada dimensão  $d = 1, \dots, D$  faça
5:    $ord\_dim[d] \leftarrow \text{sort\_by\_key (Thrust) de } X[:, d]$  [GPU]
6: fim para
7: // Construção da árvore (BFS) [CPU]
8:  $fila \leftarrow \{(raiz, X, S, ord\_dim, ord\_dist)\}$ 
9: enquanto  $fila$  não vazia faça
10:   $retira (v, P, C, ord\_dim_v, ord\_dist_v)$  de  $fila$ 
11:  se  $|C| = 1$  então
12:     $v$  é folha com o único centroide de  $C$ ; continue
13:  fim se
14:  para cada dimensão  $d$  faça
15:     $(t_d, custo_d) \leftarrow \text{VARREDURADECORTES}(d, ord\_dim_v[d], ord\_dist_v)$  (ExGreedy original)
16:  fim para
17:   $(d^*, t^*) \leftarrow \text{corte de menor } custo_d$ 
18:  se nenhum corte válido então
19:     $v$  é folha; continue
20:  fim se
21:  para cada filho  $f \in \{\text{esq, dir}\}$  ( $x_{d^*} \leq t^*$  ou  $x_{d^*} > t^*$ ) faça
22:     $P_f, C_f \leftarrow \text{amostras e centroides de } P, C \text{ que ficam do lado de } f$ 
23:     $ord\_dim_f[d] \leftarrow \text{FILTRA}(ord\_dim_v[d], P_f)$  para todo  $d$ 
24:     $ord\_dist_f \leftarrow \text{FILTRA}(ord\_dist_v, P_f, C_f)$ 
25:     $\text{insere } (f, P_f, C_f, ord\_dim_f, ord\_dist_f)$  em  $fila$ 
26:  fim para
27: fim enquanto
28: return  $T$ 

```

híbrida, também disponibilizada pelo Google Colab. A avaliação dividiu-se em duas etapas. Uma consiste na verificação de corretude das árvores geradas pelas implementações sequencial otimizada e híbrida, comparadas nó a nó com a estrutura produzida pela implementação sequencial traduzida, a partir de um percurso em pré-ordem que registra a dimensão e o valor de cada corte, ou, caso seja uma folha, o *cluster* a que foi atribuída. A outra etapa é a avaliação de desempenho: o tempo de execução das três implementações foi medido utilizando o tempo de relógio real (*wall-clock time*). Adicionalmente, realizou-se uma execução prévia de *warm-up* na GPU antes do início da cronometragem, para excluir da análise o custo fixo de inicialização do *driver* CUDA.

Os experimentos foram realizados para sete *datasets* de tamanhos e dimensões diferentes. Os conjuntos de dados utilizados são alguns dos utilizados por [Murtinho 2025] na avaliação dos algoritmos propostos em seu trabalho, como o ExGreedy. Para reprodutibilidade, disponibilizamos os conjuntos no repositório público [Freitas 2025a].

5. Resultados

Para efeito de contextualização e estabelecimento de uma linha de base, a Tabela 1 apresenta o tempo médio de execução do algoritmo original, na implementação em Python com chamadas a uma biblioteca em C, tal como disponibilizado por [Murtinho 2025], antes de qualquer tradução ou otimização realizada neste trabalho.

Tabela 1. Tempo médio de execução do algoritmo original (Python + C).

Dataset	Original Python+C (ms)
<i>Iris</i>	2,742
<i>Digits</i>	1.049,239
<i>Pendigits</i>	535,630
<i>Avila</i>	1.741,386
<i>Sensorless</i>	19.372,807
<i>Covtype</i>	62.594,833
<i>Poker</i>	30.233,002

Esses valores servem apenas de contexto: como o tempo depende da linguagem, todas as comparações usam a tradução para C. Eles mostram, como em [Leiserson et al. 2020], que há muito espaço na pilha de *software* para acelerar mesmo em CPU apenas reescrevendo o código.

Assim, para avaliar de forma justa o impacto da paralelização em GPU de parte do algoritmo ExGreedy, comparamos três implementações: a versão sequencial traduzida para C diretamente do algoritmo original de [Murtinho 2025], nossa versão sequencial com a otimização de reaproveitamento de ordenações entre nós (Sequencial otimizado) e a versão híbrida CPU-GPU, na qual o cálculo da matriz de distâncias e as ordenações iniciais são realizados em paralelo na GPU, antes da construção da árvore. Para cada *dataset*, executamos 5 vezes as três versões do algoritmo, medindo o tempo de construção da árvore (sem incluir o tempo de leitura dos *datasets* e de compilação dos códigos).

Essa separação em três versões permite isolar o ganho obtido exclusivamente pela reestruturação algorítmica sequencial do ganho obtido pela paralelização em GPU, conforme discutido na seção de metodologia. Além do tempo total de execução (que inclui o pré-processamento e a construção da árvore), medimos separadamente o tempo gasto apenas na etapa de pré-processamento (cálculo de distâncias e ordenações), por ser essa a única parte do algoritmo atualmente delegada à GPU na versão híbrida.

A Tabela 2 apresenta o tempo médio total de execução das três versões, em milissegundos, e dois fatores de aceleração: o ganho da otimização sequencial isolada (Sequencial / Seq. Otimizado) e o ganho da versão híbrida em relação à sequencial já otimizada (Seq. Otimizado / Híbrido), retratando a comparação mais justa para avaliar isoladamente a contribuição da utilização de GPU.

A otimização sequencial reduziu consideravelmente o tempo de execução em todos os *datasets*, com ganhos de aceleração entre $1.28\times$ (*Iris*) e $4.45\times$ (*Avila*), confirmando que parte relevante do ganho de desempenho final é atribuível também à reestruturação algorítmica realizada.

Tabela 2. Tempo médio total de construção da árvore (ms) e *speedups*.

<i>Dataset</i>	N	D	K	Seq. (ms)	Seq. Otim. (ms)	Híbrido (ms)	Seq/SeqOtim	SeqOtim/Híbrido
<i>Iris</i>	150	4	3	0,185	0,145	3,072	1,28×	0,05×
<i>Digits</i>	5.620	64	10	483,890	131,586	142,043	3,68×	0,93×
<i>Pendigits</i>	10.992	16	10	339,301	116,232	76,061	2,92×	1,53×
<i>Avila</i>	20.866	10	12	1.216,022	273,311	208,588	4,45×	1,31×
<i>Sensorless</i>	58.508	48	11	13.489,924	4.980,236	2.435,206	2,71×	2,05×
<i>Covtype</i>	581.012	54	7	42.988,197	14.945,856	8.406,658	2,88×	1,78×
<i>Poker</i>	1.025.009	10	10	16.971,374	7.100,683	4.566,410	2,39×	1,55×

Quando comparada à versão sequencial otimizada, a versão híbrida apresentou ganhos de desempenho para os *datasets* *Pendigits*, *Avila*, *Sensorless*, *Covtype* e *Poker*, com ganhos de aceleração entre $1.31\times$ e $2.05\times$. Observa-se que o *Sensorless*, *dataset* que possui tanto dimensionalidade, quanto número de amostras significativos ($D = 48$ e $N = 58508$), apresentou a maior aceleração total entre todos os testados. Ressalta-se que, para os três últimos *datasets*, onde o número de amostras é da ordem de centenas de milhares ou milhões, passa a ser interessante as otimizações.

Os *datasets* *Iris* e *Digits* são pequenos e servem mais à validação de corretude: nesses casos a GPU fica subutilizada e o custo fixo domina o tempo total. Contudo, esses conjuntos de dados menores permanecem úteis para identificar gargalos iniciais e direcionar otimizações no código.

Para investigar mais a fundo o comportamento observado em *Digits*, além de possibilitar avaliações mais justas e focadas no efeito de paralelização, isolamos os tempos gastos por cada *dataset* exclusivamente na etapa de pré-processamento, a única parte do algoritmo efetivamente delegada à GPU. Os tempos medidos são apresentados na Tabela 3.

Tabela 3. Tempo médio da etapa de pré-processamento (ms) e *speedup*.

<i>Dataset</i>	Seq. Otim. (ms)	Híbrido (ms)	<i>Speedup</i>
<i>Iris</i>	0,065	2,960	0,02×
<i>Digits</i>	27,384	24,173	1,13×
<i>Pendigits</i>	34,511	10,851	3,18×
<i>Avila</i>	42,307	12,079	3,50×
<i>Sensorless</i>	695,191	81,047	8,58×
<i>Covtype</i>	4.267,381	873,473	4,89×
<i>Poker</i>	2.079,052	367,498	5,66×

A Tabela 3 mostra que a paralelização em GPU acelera de forma significativa a etapa de pré-processamento, com *speedups* entre $3,18\times$ e $8,58\times$ para todos os *datasets* que demonstraram diminuição no tempo total do algoritmo na Tabela 2. Para o *Digits*, embora tenha havido ganho nesta etapa ($1,13\times$), ele foi substancialmente inferior ao observado nos demais conjuntos. Essa diferença explica o motivo de o *Digits* ter sido o único *dataset* de escala relevante sem ganho líquido no tempo total, pois como o pré-processamento representa apenas cerca de 20,8% do seu tempo total de execução no algoritmo sequencial otimizado, o ganho obtido nessa fase não foi suficiente para compensar o *overhead* fixo de transferência de dados e inicialização da GPU.

Por fim, visando garantir que a paralelização em GPU e as otimizações sequenciais introduzidas não alteraram substancialmente o resultado do algoritmo, validamos a corretude comparando, para cada *dataset*, as estruturas das árvores geradas pelas versões Sequencial original (Python + C), Sequencial traduzida, Sequencial otimizada e Híbrida. Em seis dos sete *datasets* avaliados, as quatro versões produziram árvores estruturalmente idênticas. No *dataset Sensorless*, observamos uma divergência pontual em um único nó interno (de nove nós de decisão): as versões sequenciais selecionaram a dimensão 10 (limiar 0.124570) enquanto a versão híbrida selecionou a dimensão 11 (limiar 0.124460), valores extremamente próximos entre si, sugerindo um custo quase empatado entre as duas dimensões naquele nó. Todos os demais cortes e todas as folhas (atribuições de *cluster*) permaneceram idênticos entre as versões nesse *dataset*, indicando que a divergência não se propagou para o restante da árvore.

Atribuímos essa divergência pontual a diferenças de precisão numérica entre CPU e GPU, e não a um erro de tradução do algoritmo. Isso porque o cálculo da matriz de distâncias na GPU é realizado por meio de operações de multiplicação-soma fundidas (FMA), que arredondam de forma diferente das operações sequenciais equivalentes em CPU, além disso, a ordenação dos centros por distância utiliza implementações distintas nas duas versões (`qsort` sequencial vs. `cub::DeviceSegmentedSort` paralelo), nenhuma delas garante estabilidade em casos de empate. Como para este trabalho utilizamos o fator de penalização por profundidade desativado ($depth_factor = 0$), visto que nosso foco era o ExGreedy, o custo de cada corte depende inteiramente da soma das distâncias aos centroides, tornando o algoritmo sensível a esse tipo de ruído numérico quando duas dimensões apresentam custo muito próximo. Ressaltamos que essa divergência ocorreu em apenas 1 nó de decisão e afetou um único *dataset*, além de não alterar o número de nós nem a atribuição final de *clusters* da árvore, de modo que os ganhos de desempenho reportados nesta seção não são comprometidos por diferenças no resultado final do algoritmo.

A Figura 2 apresenta as estruturas simplificadas das árvores geradas pelas quatro versões do ExGreedy (Sequencial original, Sequencial traduzida, Sequencial otimizada e Híbrida) para os conjuntos *Poker* e *Pendigits*. Para facilitar a visualização, os nós internos exibem apenas a *feature* selecionada para o corte, enquanto as folhas indicam o *cluster* atribuído. Vale ressaltar que, como o foco deste trabalho é a paralelização do algoritmo puramente guloso, a penalização por profundidade manteve-se desativada ($depth_factor = 0$), resultando no forte desbalanceamento observado em parte das árvores.

Outro ponto interessante de análise a partir das árvores geradas é que o algoritmo ExGreedy [Laber and Murtinho 2021, Murtinho 2025], cuja lógica de construção foi preservada em todas as versões, não precisa utilizar todas as variáveis (*features*) do conjunto de dados para conseguir separar os grupos obtidos pelo K-means. Isso é relevante do ponto de vista da explicabilidade, pois como discutido na Seção 2.1, a interpretação direta de um agrupamento se torna mais difícil à medida que a dimensionalidade dos dados aumenta, de modo que uma árvore capaz de reproduzir a partição do K-means utilizando apenas um subconjunto reduzido das dimensões originais oferece uma explicação mais compacta e compreensível do que a análise direta de todas as D variáveis. Os dados da Tabela 4 mostram que esse descarte de variáveis é mais forte em conjuntos de da-

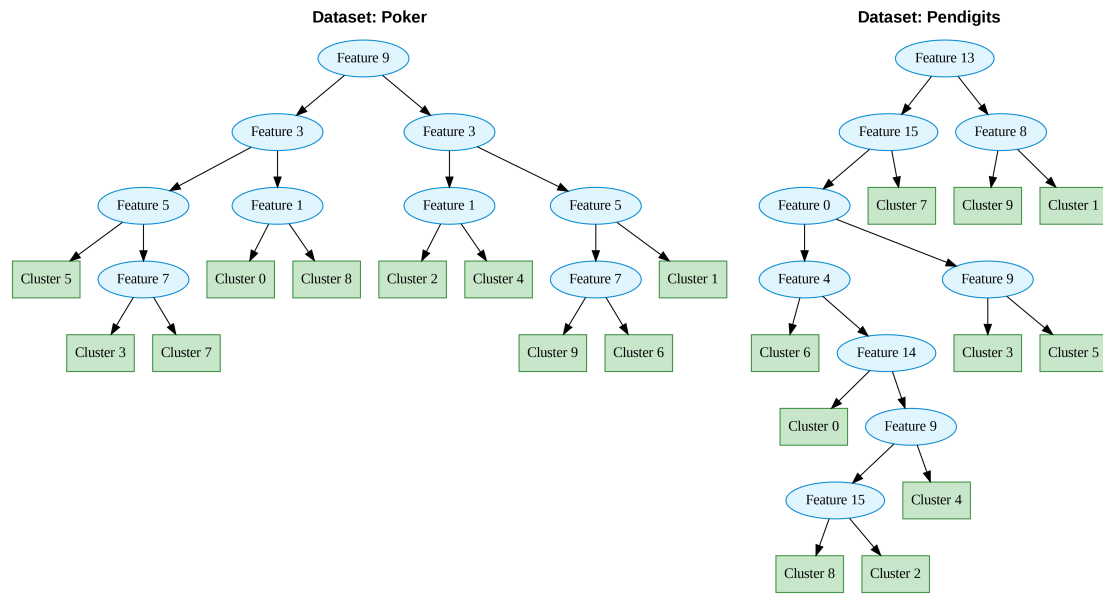


Figura 2. Estruturas simplificadas das árvores de decisão geradas para os conjuntos de dados *Poker* e *Pendigits*, destacando *features* de corte nos nós internos e *clusters* nas folhas.

dos com muitas dimensões. Nos *datasets Covtype*, *Digits* e *Sensorless* (todos com 48 ou mais *features* originais), o algoritmo utilizou apenas entre 11,1% e 16,7% das variáveis disponíveis para isolar todos os *clusters*. Por outro lado, em conjuntos com menos atributos, como *Avila* e *Poker* (ambos com 10 dimensões), a taxa de utilização foi bem maior, chegando a 80,0% e 50,0%, respectivamente.

Sem a penalização por profundidade, as árvores ficam desbalanceadas (Tabela 4): *Poker* gerou uma árvore rasa (profundidade máxima 4), enquanto *Avila* e *Digits* chegaram a 11 e 9, evidenciando que alguns *clusters* exigem muito mais cortes para serem separados.

Tabela 4. Resumo da estrutura das árvores.

Dataset	Dimensões (<i>D</i>)	Features Usadas	% Features Usadas	Prof. Média	Prof. Máx.
<i>Iris</i>	4	2	50,0%	1,67	3
<i>Digits</i>	64	9	14,1%	6,10	9
<i>Pendigits</i>	16	7	43,8%	4,30	7
<i>Avila</i>	10	8	80,0%	6,42	11
<i>Sensorless</i>	48	8	16,7%	5,45	9
<i>Covtype</i>	54	6	11,1%	3,86	6
<i>Poker</i>	10	5	50,0%	3,40	4

6. Conclusões

Os resultados obtidos indicam que a reestruturação algorítmica sequencial e a paralelização em GPU contribuem de forma complementar para acelerar a construção da árvore de decisão explicável para o K-means. A otimização sequencial isolada produziu ganhos expressivos em todos os *datasets* avaliados, com *speedups* entre $1.28\times$ (*Iris*) e

$4.45\times$ (*Avila*) em relação à versão sequencial traduzida diretamente do algoritmo original. Partindo dessa versão já otimizada, a versão híbrida CPU-GPU apresentou ganhos adicionais consistentes para os *datasets* *Pendigits*, *Avila*, *Sensorless*, *Covtype* e *Poker*, com *speedups* entre $1.31\times$ e $2.05\times$, demonstrando que a contribuição específica da GPU permanece relevante mesmo após a reestruturação algorítmica. As exceções observadas (*Iris* e *Digits*) foram discutidas na Seção 5 e se devem, respectivamente, ao pequeno volume de trabalho e ao ganho discreto de *speedup* na etapa de pré-processamento.

Um tópico observado que se destaca é que o ganho da paralelização não se relaciona apenas ao número de amostras (N) dos *datasets* avaliados. O conjunto *Sensorless*, com maior dimensionalidade entre os testados ($D = 48$), apresentou o maior *speedup* total, superando os que apresentavam N consideravelmente maior, como *Covtype* e *Poker*. Isso sugere que a dimensionalidade, ao escalar diretamente o custo do cálculo de distâncias e das ordenações, pode ser tão determinante quanto o número de amostras para prever o ganho obtido pela paralelização. A investigação dessa relação de forma controlada, variando N , D e K independentemente é deixada como trabalho futuro.

Além disso, a avaliação de corretude e a análise estrutural confirmaram que as otimizações e a paralelização preservaram o resultado e as propriedades explicativas inerentes ao algoritmo ExGreedy original, com uma única exceção pontual e sem impacto na estrutura final da árvore (a divergência, já discutida, atribuída a diferenças de precisão numérica entre CPU e GPU, não a um erro de adaptação do algoritmo). Observou-se que a versão híbrida reproduz fielmente a capacidade do método de realizar uma seleção implícita de variáveis, utilizando uma fração reduzida do espaço de atributos original. Adicionalmente, a desativação da penalização por profundidade resulta, como esperado do comportamento guloso original, em topologias mais desbalanceadas, sem que a aceleração por hardware feita através da paralelização e a mudança estrutural do código comprometessem a interpretabilidade do modelo final. Como trabalho futuro, mecanismos de desempate explícitos entre dimensões de custo equivalente podem tornar esse comportamento mais previsível e reduzir a chance de divergências.

Por fim, este trabalho restringiu a paralelização ao pré-processamento, e a construção da árvore permanece sequencial. Paralelizar essa etapa, ou aplicar abordagens semelhantes a outros algoritmos de explicabilidade para clusterização, são direções promissoras.

Agradecimentos

Aos programas institucionais de bolsa de iniciação científica da FAPEMIG e CNPq, ao FAPEMIG (APQ-01577-22), Funarbe e ao Curso de Especialização em Robótica. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES) Código de Financiamento 001.

Disponibilidade de Código

O código-fonte e todo o material necessário para reproduzir os experimentos estão disponíveis publicamente: códigos [Freitas 2025b] e *datasets* [Freitas 2025a].

Referências

- 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.
- Cheng, J., Grossman, M., and McKercher, T. (2014). *Professional CUDA c programming*. John Wiley & Sons.
- Costa, V. G. and Pedreira, C. E. (2023). Recent advances in decision trees: an updated survey. *Artificial Intelligence Review*, 56(5):4765–4800.
- Ezugwu, A. E., Ikotun, A. M., Oyelade, O. O., Abualigah, L., Agushaka, J. O., Eke, C. I., and Akinyelu, A. A. (2022). A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, 110:104743.
- Ferreira, R., Magalhães, S. V. G. d., and Nacif, J. A. M. (2019). Métricas e números: Desmistificando a programação de alto desempenho em GPU. In Menotti, R. and Galante, G., editors, *Minicursos do WSCAD 2019*, chapter 1. Sociedade Brasileira de Computação, Porto Alegre.
- Ferreira, R. d. S. (2025). Mais produtividade com LLMs, engenharia de prompt, aprendizado de máquina e GPUs. In Cordeiro, D. and Bianchini, C. P., editors, *Minicursos do SSCAD 2025*, chapter 1. Sociedade Brasileira de Computação, Porto Alegre.
- Freitas, I. C. (2025a). Datasets k-means explicável. <https://github.com/isabela-cfreitas/datasets-kmeans-explicavel>. Acessado em: 7 out. 2026.
- Freitas, I. C. (2025b). Explicabilidade k-means utilizando cuda. https://github.com/isabela-cfreitas/Explicabilidade_K-means_utilizando_CUDA. Acessado em: 7 out. 2026.
- Frost, N., Moshkovitz, M., and Rashtchian, C. (2020). Exkmc: Expanding explainable k-means clustering. *arXiv preprint arXiv:2006.02399*.
- Gabidolla, M. and Carreira-Perpiñán, M. Á. (2022). Optimal interpretable clustering using oblique decision trees. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 400–410.
- He, Q., Zhang, F., Bian, G., and Zhang, W. (2025). Optimization of maximum parallel k-means algorithm based on gpu cluster. *Cluster Computing*, 28(8):489.
- Hu, L., Jiang, M., Dong, J., Liu, X., and He, Z. (2026). Interpretable clustering: A survey. *ACM Computing Surveys*, 58(8):1–21.
- Hu, L., Jiang, M., Liu, X., and He, Z. (2025). Significance-based decision tree for interpretable categorical data clustering. *Information Sciences*, 690:121588.
- Laber, E., Murtinho, L., and Oliveira, F. (2023). Shallow decision trees for explainable k-means clustering. *Pattern Recognition*, 137:109239.
- Laber, E. S. and Murtinho, L. (2021). On the price of explainability for some clustering problems. In *International Conference on Machine Learning*, pages 5915–5925. PMLR.

- Leiserson, C. E., Thompson, N. C., Emer, J. S., Kuszmaul, B. C., Lampson, B. W., Sanchez, D., and Schardl, T. B. (2020). There’s plenty of room at the top: What will drive computer performance after moore’s law? *Science*, 368(6495):eaam9744.
- Li, Y., Zhao, K., Chu, X., and Liu, J. (2013). Speeding up k-means algorithm by gpus. *Journal of Computer and System Sciences*, 79(2):216–229.
- MacQueen, J. (1967). Multivariate observations. In *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. University of California press Oakland, CA, USA.
- Merrill, D. and NVIDIA Corporation (2025). CUB: CUDA UnBound. <https://nvidia.github.io/cccl/cub/>. Biblioteca de primitivas paralelas de alto desempenho para CUDA. Acessado em: 7 out. 2026.
- Moshkovitz, M., Dasgupta, S., Rashtchian, C., and Frost, N. (2020). Explainable k-means and k-medians clustering. In *International conference on machine learning*, pages 7055–7065. PMLR.
- Murtinho, L. S. (2025). *Clustering under constraints: explainability via decision trees and separability with minimum size*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brasil.
- Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., and Phillips, J. C. (2008). Gpu computing. *Proceedings of the IEEE*, 96(5):879–899.
- Pan, Z. and Mishra, P. (2022). Hardware acceleration of explainable machine learning. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1127–1130. IEEE.
- Penha, J., Silva, A., Barros, O., Moreira, I., Nacif, J., and Ferreira, R. (2023). Avaliação de estilos de código para Árvores de decisão em gpu com microbenchmarks. In *Anais do XXIV Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 277–288, Porto Alegre, RS, Brasil. SBC.
- Penha, J. C., Bragança, L., Coelho, K., Canesche, M., Silva, J., Comarela, G., Nacif, J. A. M., and Ferreira, R. (2018). A gpu/fpga-based k-means clustering using a parameterized code generator. In *2018 Symposium on High Performance Computing Systems (WSCAD)*, pages 61–69. IEEE.
- Perkel, J. M. (2018). Why jupyter is data scientists’ computational notebook of choice. *Nature*, 563(7732):145–147.
- Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215.