Dissertação de Mestrado

Estudo comparativo sobre meta-heurísticas em GPU para clusterização de dados

Mario Diego Ferreira dos Santos
mdfs@ic.ufal.br

Orientadores:

Prof. Dr. Bruno Costa e Silva Nogueira
Prof. Dr. Rian Gabriel Santos Pinheiro

Maceió, abril de 2021

Mario Diego Ferreira dos Santos

Estudo comparativo sobre meta-heurísticas em GPU para clusterização de dados

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal de Alagoas.

Orientadores:

Prof. Dr. Bruno Costa e Silva Nogueira

Prof. Dr. Rian Gabriel Santos Pinheiro

Maceió, abril de 2021

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico

Bibliotecário: Marcelino de Carvalho Freitas Neto – CRB-4 - 1767

S237e Santos, Mario Diego Ferreira dos.
Estudo comparativo sobre meta-heurísticas em GPU para clusterização de dados / Mario Diego Ferreira dos Santos. – 2021.
58 f. : il.

Orientador: Bruno Costa e Silva Nogueira.
Co-orientador: Rian Gabriel Santos Pinheiro.
Dissertação (mestrado em Informática) - Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2021.

Bibliografia: f. 52-58.

1. Clusterização de dados. 2. Meta-heurística. 3. Unidades de processamento gráfico. 4. Otimização. I. Título.

CDU: 004.932

Folha de Aprovação

Mario Diego Ferreira dos Santos

Estudo comparativo sobre meta-heurísticas em GPU para clusterização de dados

Dissertação submetida ao corpo docente do Programa de Pós-Graduação em Informática da Universidade Federal de Alagoas e aprovada em 30 de abril de 2021.

Banca Examinadora:

Prof. Dr. Bruno Costa e Silva Nogueira (Orientador)

Prof. Dr. Rian Gabriel Santos Pinheiro (Coorientador)

Prof. Dr. ERICK DE ANDRADE BARBOZA (Examinador Interno)

Prof. Dr. ERMESON CARNEIRO DE ANDRADE (Examinador Externo)

Resumo

Clusterização é uma classe fundamental de problemas com aplicações em muitas áreas do conhecimento, incluindo: bioinformática, visão computacional, mineração de dados, mineração de texto e agrupamento de páginas na *Web*. Dado um conjunto de n objetos, a clusterização objetiva agrupar automaticamente tais objetos em k grupos, geralmente disjuntos, denominados *clusters* ou agrupamentos utilizando uma medida de similaridade preestabelecida. Problemas de clusterização em geral têm alta complexidade computacional e envolvem uma grande quantidade de dados de entrada. Dessa forma, o uso de arquiteturas paralelas como *Graphics Processing Units* (GPUs) são alternativas interessantes para acelerar o processo de clusterização. Neste trabalho, foi conduzido um estudo comparativo de meta-heurísticas aceleradas por GPU para agrupamento de dados. Três meta-heurísticas populacionais foram implementadas na GPU: *Particle Swarm Optimization* (PSO), *Differential Evolution* (DE) e *Scatter Search* (SS). A implementação dessas meta-heurísticas foi dividida em duas partes: a parte independente do problema e a parte dependente problema. A parte independente do problema se refere aos operadores de seleção, reposição e combinação de cada meta-heurística, enquanto que a parte dependente se refere a função objetivo. A parte independente foi implementada usando o *framework* `libCudaOptimize`, e a parte dependente foi criada transformando o problema de clusterização em um problema de otimização global sujeito a restrições de caixa. As meta-heurísticas propostas foram comparadas com o melhor algoritmo de clusterização da atualidade C-GRASP-Clu considerando o tempo de execução e qualidade da solução. Os resultados indicam que o PSO baseado em GPU (GPU-PSO) obteve os melhores resultados em comparação com as outras meta-heurísticas baseadas em GPU e o melhor método da atualidade. Além disso, nossa implementação em GPU da função objetivo obteve um *speedup* médio de **175x** sobre a versão sequencial. Os resultados demonstram que uma abordagem de GPU para o problema de clusterização é muito promissora.

Palavras-chave: Clusterização de dados, Meta-heurísticas, GPU, Otimização.

Abstract

Clustering is a fundamental class of problems with applications in many areas of knowledge, including: bioinformatics, computer vision, data mining, text mining, and web page grouping. Given a set of *n* objects, clustering aims to automatically group such objects in *k* groups, usually disjunct, called clusters or groupings using a pre-established similarity measure. Clustering problems in general have high computational complexity and involve a large amount of input data. Thus, the use of parallel architectures such as Graphics Processing Units (GPUs) is interesting alternatives to accelerate the clustering process. In this work, we conducted a comparative study of GPU-accelerated metaheuristics for grouping data. Three population meta-heuristics were implemented in the GPU: Particle Swarm Optimization (PSO), Differential Evolution (DE), and Scatter Search (SS). The implementation of these meta-heuristics was divided into two parts: the independent part of the problem and the dependent part of the problem. The independent part of the problem refers to the selection, replacement, and combination operators for each meta-heuristic, while the dependent part refers to the objective function. The independent part was implemented using the `libCudaOptimize` framework, and the dependent part was created by transforming the clustering problem into a global optimization problem subject to cash constraints. The proposed meta-heuristics were compared with the best current clustering algorithm C-GRASP-Clu considering the execution time and quality of the solution. The results indicate that the GPU-based PSO (GPU-PSO) obtained the best results in comparison with the other GPU-based metaheuristics and the best method today. Also, our GPU implementation of the objective function obtained an average speedup of **175x** over the sequential version. These results demonstrate that a GPU approach to the clustering problem is very promising.

Keywords: Data Clustering, Metaheuristics, GPU, Optimization.

Agradecimentos

Agradeço, primeiramente, a Deus, pois sempre me deu forças espirituais para enfrentar as dificuldades da vida. Ele é meu conforto e guia-me na minha infinita ignorância quanto à complexidade da vida.

Aos meus pais, Mario Ferreira e Glaucia Maria, pelo amor, dedicação, carinho, parceria, esforços para me ajudar em meus projetos pessoais e profissionais, além de seus ensinamentos que são essenciais para a minha construção como ser.

Aos meus irmãos, Marley Douglas, Gleice Dayana e Rouse Maria, com os quais partilho bons momentos e são tão importantes na minha vida.

A minha esposa, Suzy Kamylla, pelo amor, paciência, confiança e, principalmente, por ter aceitado estar comigo nos momentos bons e ruins. Agradeço por acreditar em mim e incentivar nos meus objetivos.

Ao meu orientador, Prof. Dr. Bruno Costa e Silva Nogueira, que durante a elaboração desta dissertação contribuiu para direcionar meus estudos, aperfeiçoar minhas pesquisas e concretizar esse trabalho. Agradeço também ao meu coorientador Prof. Dr. Rian Gabriel Santos Pinheiro por me orientar e passar todos os seus ensinamentos de bom grado na busca de realização desse projeto, que para mim é a realização de um sonho e uma conquista ímpar na minha vida.

Aos Professores Doutores Erick de Andrade Barboza e Ermeson Carneiro de Andrade por aceitarem o convite de fazer parte da banca examinadora.

Aos professores do Instituto de Computação, do Programa de Pós-Graduação em Informática, pela dedicação e conhecimentos passados nas aulas ministradas, que muito contribuíram para a minha formação acadêmica e pessoal.

À Fundação CAPES pela concessão da bolsa de mestrado, que possibilitou a realização desta pesquisa.

Aos meus colegas e amigos do mestrado que me ajudaram em muitas dificuldades que tive no caminho. Agradeço ainda a todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

Mario Diego

"Elogie em público, e corrija em particular.
Um sábio orienta sem ofender, e ensina sem humilhar!"
Mario Sérgio Cortella

À minha família e aos meus orientadores,
pelo amor e apoio.

Lista de Figuras

Lista de Tabelas

Conteúdo

Capítulo 1

INTRODUÇÃO

Clusterização é uma classe de problemas fundamentais com aplicações em muitas disciplinas (Kogan et al., 2006). Algumas áreas que abordam problemas de clusterização de dados são: bioinformática, visão computacional, mineração de dados, análise de expressão genética, mineração de texto, agrupamento de páginas na *Web* entre outras áreas. A solução de um problema de clusterização consiste em um agrupamento que é utilizado para descobrir grupos naturais em conjuntos de dados. Dado um conjunto de n objetos, a clusterização objetiva agrupar automaticamente tais objetos em k grupos, geralmente disjuntos, denominados *clusters* ou agrupamentos utilizando uma medida de similaridade preestabelecida. Deste modo, a clusterização visa identificar estruturas abstratas sem qualquer conhecimento prévio das características dos dados. Inúmeras contribuições recentes para esta área de pesquisa estão espalhadas em uma variedade de publicações em diversos campos de pesquisas.

De acordo com Cassiano & Pessanha (2014), estudos sobre clusterização começaram na Antropologia, que é a ciência que tem como objeto o estudo sobre o ser humano e a humanidade de maneira totalizante, ou seja, abrangendo todas as suas dimensões humanas. No entanto, o primeiro trabalho publicado referindo-se a um método de clusterização foi o trabalho de Sorensen (1948), onde o autor definiu o método hierárquico aglomerativo de ligação completa. A partir deste estudo, diferentes técnicas de clusterização foram desenvolvidas.

Problemas de clusterização em geral têm alta complexidade computacional e envolvem uma grande quantidade de dados de entrada. Dessa forma, o uso de arquiteturas paralelas como *Graphics Processing Units* (GPUs) são alternativas interessante para acelerar o processo de clusterização. Neste sentido, o trabalho utilizou a função objetivo para minimização das distâncias *intra-clusters*, ou seja, a distância entre elementos do mesmo *clusters* para resolver problemas de clusterização de dados.

A GPU é um processador paralelo dedicado para otimizar e acelerar cálculos gráficos. As GPUs foram especificamente projetadas para tarefas com muitos cálculos, dessa forma, as GPUs são essenciais para renderização de gráficos 3D. As GPUs modernas são massivamente paralelas e são totalmente programáveis (McClanahan, 2010).

Segundo Van Luong et al. (2011), durante anos, a paralelização por meio de GPU foi dedicado a aplicações gráficas. Recentemente, seu uso foi estendido para outros domínios de aplicação (Che et al., 2008).

Os primeiros trabalhos referentes a GPU utilizando o *Compute Unified Device Kit* que é um conjunto de ferramentas de desenvolvimento de arquitetura *Compute Unified Device Architecture* (CUDA)[1] que permite programação de GPU em linguagem C/C++.

Nas últimas décadas, houve um grande avanço nas arquiteturas das unidades de processamento gráfico (GPU). A cada ano, a presença de computadores compostos por processadores *multi-core*, e neste contexto, as GPUs têm obtidos ótimos desempenhos (Essaid et al., 2019).

Nesse sentido, as GPUs evoluíram de aceleradores gráficos dedicados e deram origem a GPUs de uso geral modernas, onde fornecem recursos paralelos para computação de alto desempenho. A tecnologia da GPU tende a adicionar ainda mais programabilidade e paralelismo, fornecendo recursos com alto desempenho e superioridade sobre um único núcleo de processamento de propósito geral. A disponibilidade de *Application Programming Interface* (API) facilitou a implementação de aplicativos paralelos, mas ainda é necessário seguir alguns princípios cruciais para obter um desempenho eficiente das GPUs.

Esta dissertação apresenta um estudo comparativo de meta-heurísticas em GPU. Três meta-heurísticas populacionais foram implementadas na GPU: *Particle Swarm Optimization* (PSO) (Kennedy, 1995), *Differential Evolution* (DE) (Storn & Price, 1997), *Scatter Search* (SS) (Glover et al., 2003).

A performance dos algoritmos é observada por meio de experimentos computacionais e estatísticos. As meta-heurísticas em GPU também foram comparadas com a abordagem mais proeminente da literatura, o algoritmo C-GRASP-Clu (Queiroga et al., 2018).

As contribuições desse trabalho podem ser listadas a seguir:

- Implementação da **paralelização da função objetivo**. A paralelização pode ser utilizada em outras meta-heurísticas utilizando a arquitetura de GPU. Com a paralelização da função objetivo busca-se alcançar um *speedup* alto em relação ao algoritmo sequencial.

- **Integração da função objetivo** nas meta-heurísticas GPU-PSO, GPU-DE e GPU-SS. Até onde os autores conhecem, esse será o primeiro trabalho que realiza um estudo profundo sobre meta-heurísticas em GPU para o problema da clusterização de dados.

- **Análise comparativa** considerando os algoritmos em GPU propostos e a abordagem mais promissora da atualidade.

[1]CUDA - (*Compute Unified Device Architecture*). É uma API destinada a computação paralela, GPGPU, e computação heterogênea, criada pela Nvidia.

1.1 Justificativa

A proposição de novos métodos e mecanismos de exploração no espaço de busca é importante para o avanço na área de meta-heurísticas, podendo gerar técnicas futuras mais eficientes em diferentes problemas com aplicações relevantes. Nesta perspectiva, o trabalho justifica-se pelos seguintes motivos:

1. A importância de trabalhos relacionados a clusterização de dados torna-se relevante com uso de métodos rápidos.
2. Problemas de clusterização são computacionalmente intensos e envolvem dados massivos.
3. As GPUs têm avançado bastante nos últimos anos e podem ser uma interessante estratégia para acelerar o processo de clusterização.
4. Por fim, ao conhecimento dos autores, não existe na literatura um estudo profundo analisando meta-heurísticas em GPU para o problema de clusterização com k fixo e a distância *intra-clusters*.

1.2 Objetivos

Nesta seção, são apresentados os objetivos gerais e específicos, bem como a hipótese do trabalho.

1.2.1 Objetivo Geral

Implementar e analisar comparativamente implementações em GPU das principais meta-heurísticas da literatura para a solução do problema de clusterização de dados.

1.2.2 Objetivos Específicos

- Implementar a paralelização da função objetivo em GPU.
- Implementar a função objetivo em GPU nas meta-heurísticas *Particle Swarm Optimization* (PSO) (Kennedy, 1995), *Differential Evolution* (DE) (Storn & Price, 1997) e *Scatter Search* (SS) (Glover et al., 2003).
- Calibrar os parâmetros das meta-heurísticas implementadas em GPU.

1.2.3 Hipótese

Com os objetivos traçados, formula-se a hipótese que o algoritmo proposto deve ter um *speedup* alto da função objetivo em GPU em comparação ao método sequencial. Desta forma, espera-se que a versão paralela das meta-heurísticas em GPU tenham um desempenho superior comparado à melhor heurística da literatura, que é o C-GRASP-Clu.

1.3 Estrutura do trabalho

A continuação deste trabalho está organizada da seguinte maneira:

- Capítulo 2: contém a fundamentação teórica referente a problemas de clusterização de dados, como também as medidas de similaridades, métodos de agrupamentos de dados, arquitetura de GPU/CPU. Por fim, otimização e meta-heurísticas e heurística utilizadas nesta pesquisa;
- Capítulo 3: apresenta uma revisão de literatura contendo os métodos de agrupamentos de dados e as meta-heurísticas e heurística utilizando as arquiteturas de GPU e CPU;
- Capítulo 4: descreve a metodologia empregada nesta pesquisa;
- Capítulo 5: exibe e discute os resultados computacionais;
- Capítulo 6: apresenta as conclusões e os trabalhos futuros.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os conceitos e métodos necessários para um melhor entendimento da pesquisa e compreensão dos próximos capítulos desta dissertação. O capítulo descreve os fundamentos relacionados a problemas de clusterização, medidas de similaridades e os métodos de agrupamentos de dados. Também é feita uma descrição sobre arquitetura de GPU/CPU e otimização. Por fim, apresentamos uma descrição das meta-heurísticas utilizadas nesta pesquisa.

2.1 Problemas de clusterização

De acordo com a definição apresentada por Hansen & Jaumard (1997), uma partição $P = \{C_1, C_2, \ldots, C_k\}$ dos dados de entrada D em k *clusters* é definida como:

$$\begin{aligned} &C_i \neq \emptyset, && i = 1, \ldots, k \\ &C_i \cap C_j = \emptyset, && i, j = 1, ..., k \text{ e } i \neq j; \\ &\bigcup_{i=1}^{k} C_i = D. \end{aligned}$$

A abordagem comumente utilizada consiste em determinar os centros de *cluster* k = $\{C_1, \ldots, C_k\}$ que minimizam a soma das distâncias *intra-clusters*. Dados m pontos de entrada (objetos), em que cada ponto p_i contém d dimensões, ou seja, $p_i = \{p_i^1, \ldots, p_i^d\}$. A função é calculada somando a distância entre cada ponto p_i e o centroide c_j mais próximo. A função objetivo do problema pode ser descrita como:

$$\min \sum_{i=1}^{m} \min_{1 \geq j \geq k} \{dist(p_i, c_j)\} \tag{2.1}$$

$$dist(p_1, p_2) = \|p_1 - p_2\| = \sqrt{\left(\sum_{i=1}^{d} (p_1^i - p_2^i)^2 \right)}. \tag{2.2}$$

Em outras palavras, o objetivo do problema tratado consiste em determinar um conjunto de k centroides que induzem um particionamento dos m pontos de entrada minimizando a função objetivo da Equação (2.1). A Figura 2.1 ilustra um exemplo de clusterização de dados definida a partir de centros (representado por círculos maiores).

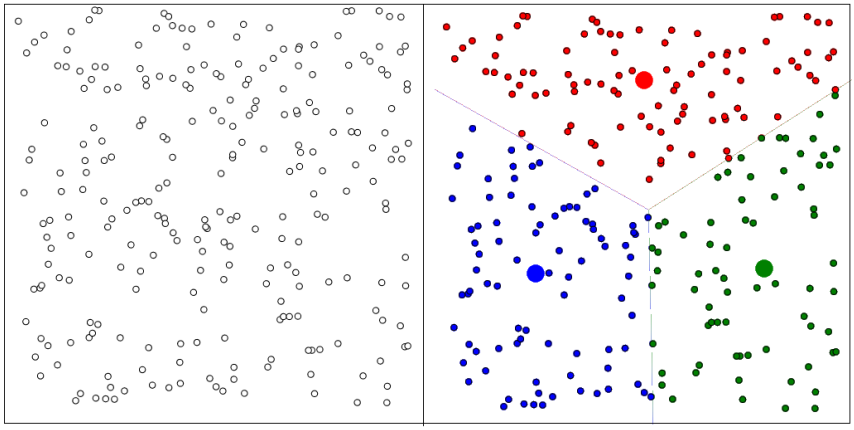


Figura 2.1: Clusterização baseada em centros de *cluster*. Diagrama de Voronoi gerando particionamento no espaço de soluções. Fonte: Autor.

Contudo, existem problemas de clusterização em que a distância não pode ser utilizada, ou não é conveniente que seja utilizada, como medida de similaridade, tendo em vista que os valores dos atributos não são escaláveis. Como exemplo, ao tratar um problema de clusterização que envolve atributos como sexo e endereço, são necessárias outras medidas que demonstrem o grau de similaridade entre as instâncias da base de dados (Ochi et al., 2004).

Conforme Ochi et al. (2004), outro exemplo em que a medida de distância não se aplica diz respeito a alguns problemas de clusterização de vértices em estruturas de grafos em que não são considerados os pesos das arestas. Nesses problemas, também referenciados como problemas de particionamento de grafos não ponderados, são necessárias, portanto, medidas que considerem apenas as conexões entre os seus vértices (Dias & Ochi, 2003; Dias, 2004; Doval et al., 1999).

Conceitos referente à medidas de similaridades podem ser vistos no trabalho de Ochi et al. (2004). Basicamente, são duas classes de Problemas de Clusterização. O caso mais estudado é onde o número de *clusters* já é previamente definido (também conhecido como o Problema de k-Clusterização ou simplesmente Problema de Clusterização (PC)). Na segunda classe do problema de clusterização, o número de *clusters* não é conhecido previamente, neste caso o

problema é denominado por Problema de Clusterização Automática (PCA) (Berkhin, 2006; Doval et al., 1999).

2.1.1 Algoritmo *k-means*

Um dos algoritmos mais conhecidos é o *k-means*, criado por Steinhaus (1956). O termo *k-means* só foi denominado no trabalho dos autores MacQueen et al. (1967). A definição do algoritmo é descrita a seguir. Dado um inteiro k representando o número de *clusters* (grupos), o *k-means* deve determinar um ponto representativo para cada *cluster*. Esse ponto será denominado *centroide*. No algoritmo, o centroide é o ponto geométrico médio obtido pela média dos valores das coordenadas dos pontos de um *cluster*. Formalmente, para cada *cluster* C_i, o centroide m_i e o ponto (objeto) P_j é obtido através da Equação (2.3).

$$m_i = \frac{1}{|C_i|} \sum_{P_j \in C_i} P_j \tag{2.3}$$

Note que o algoritmo *k-means* implicitamente busca uma clusterização que minimize a soma do erro quadrático entre os objetos e seu respectivo centroide (Queiroga et al., 2017). A seguir, é descrito o pseudocódigo do algoritmo:

Passo 1: Encontre um particionamento inicial aleatório. Um método comum é escolher pontos aleatórios no espaço para fazer o papel dos centroides e definir a clusterização por atributos a cada objeto o centroide mais próximo.

Passo 2: Calcule os centroides de cada *cluster* através da Equação (2.3) e vá para o Passo 3.

Passo 3: Defina o novo particionamento por atribuir, para cada objeto, o *cluster* representado pelo centroide mais próximo. Se houver mudança no particionamento volte para o Passo 2, caso contrário, finalize o método com a clusterização atual.

O clássico algoritmo *k-means* é um exemplo de uma heurística específica para o problema de clusterização por centroides. No entanto, o algoritmo *k-means* é sensível à inicialização e pode facilmente prover soluções em mínimos locais (Bubeck et al., 2012).

Os problemas de clusterização já são bastante estudados na literatura, principalmente na estatística e matemática. Na área de computação, esse tema ressurgiu com a popularização do conceito de mineração de dados *(Data Mining - DM)*. O que diferencia as aplicações em DM, é que nesta área, as instâncias sempre são de grande porte e cada objeto normalmente contém um número elevado de atributos ou características (Berkhin, 2006).

2.2 Medidas de similaridade em clusterização

Existem medidas de similaridades que são propostas pela literatura para solucionar problemas relacionados à clusterização de dados. Algumas medidas serão descritas ao longo desta seção.

Tendo em vista a dificuldade de se examinar todas as combinações de grupos possíveis em um grande volume de dados, desenvolveram-se diversas técnicas capazes de auxiliar na formação dos agrupamentos (Doni, 2004). Uma análise de *cluster* criteriosa exige métodos que apresentem as seguintes características (Zaïane et al., 2002):

- Ser capaz de lidar com dados de alta dimensionalidade;
- Ser "escalável" com o número de dimensões e com a quantidade de elementos a serem agrupados;
- Habilidade para lidar com diferentes tipos de dados;
- Capacidade de definir agrupamentos de diferentes tamanhos e formas;
- Exigir o mínimo de conhecimento para determinação dos parâmetros de entrada;
- Ser robusto à presença de ruído;
- Apresentar resultado consistente independente da ordem em que os dados são apresentados.

De modo geral, as métricas utilizadas nesses problemas de clusterização de dados não atendem a todos esses requisitos. Por isso, é importante entender as características de cada métrica para a escolha de um método adequado à cada tipo de dado ou problema.

2.2.1 Distância Euclidiana

A distância Euclidiana é a distância mais conhecida dentre as métricas. Essa distância é a menor distância entre dois pontos no $\mathbb{R}^n$, que pode ser representada pela hipotenusa, observada no teorema de Pitágoras visto logo abaixo. Sejam $x = (x_1 ... x_n)$ e $y = (y_1 ... y_n) \in \mathbb{R}^n$. A distância Euclidiana é definida por:

$$d_E(x,y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{2.4}$$

2.2.2 Distância de Manhattan

A distância de Manhattan, conhecida também como distância *city block* ou geometria do taxista, representa a distância entre dois pontos como a soma das diferenças absolutas de suas coordenadas. Tal distância é muito similar à distância Euclidiana. Seu nome é associado ao formato quadriculado, pois representa a maior parte das ruas da cidade de Manhattan que os taxistas tendem a tomar como rota (Krause, 1986). Dados $x, y \in \mathbb{R}^n$, a distância de Manhattan é definida por:

$$d_s(x,y) = |x_1 - y_1| + ... + |x_n - y_n| = \sum_{i=1}^{n} |x_i - y_i|. \tag{2.5}$$

2.2.3 Distância de Chebyshev

A distância de Chebyshev ou Máxima leva em consideração a maior distância entre um par de variáveis. Com isso, para cada par de amostra de dados, compara-se a diferença entre cada par de atributos, e o maior valor de diferença entre estes elementos é tomado como a distância entre as amostras de dados (Guerreiro & Breve, 2015). A fórmula que representa a distância de Chebyshev é descrita na equação abaixo:

$$d_c(x,y) = \max_i |x_i - y_i| \tag{2.6}$$

2.2.4 Distância de Mahalanobis

A aplicação da distância de Mahalanobis visa minimizar possíveis distorções que podem ocorrer pelas correlações lineares entre as características dos exemplos de dados analisados. Esta medida de distância se utiliza da correlação entre as variáveis de cada exemplo de dado, mantendo cada característica com média zero e variância um (Guerreiro & Breve, 2015).

A distância de Mahalanobis para um determinado conjunto de dados, em que x e y correspondem aos vetores de dados sendo comparados, sendo $\mathbf{S}$ a matriz de covariância do conjunto de dados, pode ser definida como:

$$d_M(x,y) = \sqrt{(x-y)^T \mathbf{S}^{-1} (x-y)} \tag{2.7}$$

2.2.5 Índice de Dunn

Inicialmente, para calcular o índice de Dunn, é necessária a distância entre dois *clusters* distintos C_k e $C_{k'}$. Para isso, é calculada a distância entre entre seus pontos mais próximos:

$$dist_{kk'} = \min_{\substack{p_i \in C_k \\ p_j \in C_{k'}}} \|p_i - p_j\|,$$

além de $dist_{min}$ que é a menor dessas distâncias $dist_{kk'}$:

$$dist_{min} = \min_{k \neq k'} dist_{kk'}.$$

Para cada *cluster* C_k, denota-se $dist_k$ a maior distância que separa dois pontos distintos no *cluster* (às vezes chamado de diâmetro do *cluster*):

$$dist_k = \max_{\substack{p_i, p_j \in C_k \\ p_i \neq p_i}} \|p_i - p_j\|.$$

Dessa forma, d_{max} é a maior dessas distâncias D_k:

$$dist_{max} = \max_{1 \leq k \leq K} dist_k.$$

Por fim, o índice de Dunn é definido como o quociente de $dist_{min}$ e $dist_{max}$:

$$Dunn = \frac{dist_{min}}{dist_{max}}.$$

2.3 Métodos de agrupamentos de dados

O processo de agrupamento de dados é responsável pelo agrupamento propriamente dito. Os métodos de agrupamento de dados podem ser divididos em duas grandes categorias, cada uma delas compreendendo diferentes tipos de algoritmos:

- *Métodos Hierárquicos* (Johnson, 1967):
  - Algoritmos Aglomerativos
  - Algoritmos Divisivos
- *Métodos Particionais* (Ruspini, 1969):
  - Algoritmos Exclusivos
  - Algoritmos Não-exclusivos

2.3.1 Métodos hierárquicos

Os métodos hierárquicos são técnicas simples onde os dados são particionados sucessivamente, produzindo uma representação hierárquica dos agrupamentos (Everitt et al., 2001). Esse tipo de agrupamento é uma representação que facilita a visualização sobre a formação dos agrupamentos em cada estágio onde ela ocorreu com um grau de semelhança entre eles.

Os métodos hierárquicos não requerem que seja definido um número *a priori* de agrupamentos. Eles requerem uma matriz contendo as métricas de distância entre os agrupamentos em cada estágio do algoritmo. Essa matriz é conhecida como matriz de similaridades entre agrupamentos.

Conforme visto nos itens acima, os métodos hierárquicos são subdivididos em: métodos aglomerativos e métodos divisivos. No método aglomerativo inicia-se com cada padrão formando seu próprio agrupamento e gradualmente os grupos são unidos até que um único agrupamento contendo todos os dados seja gerado. Desta forma, logo no início do processo, os agrupamentos são pequenos e os elementos de cada grupo possuem um alto grau de similaridade. Ao final do processo, têm-se poucos agrupamentos, cada um podendo conter muitos elementos e menos similares entre si.

Por sua vez, os métodos divisivos são os menos comuns entre os métodos hierárquicos devido a sua ineficiência e exigem uma capacidade computacional maior que os métodos hierárquicos aglomerativos. Esse método começa com um único agrupamento formado por todos os padrões e gradualmente vai dividindo os agrupamentos em grupos menores até que termine com um agrupamento por padrão.

2.3.2 Métodos particionais

Os métodos particionais são métodos baseados na minimização de uma função de custo, onde os padrões são agrupados em um número k de agrupamentos escolhido *a priori*. Cada padrão é agrupado na classe em que essa função de custo é minimizada. Uma das principais vantagens dos métodos particionais em relação aos métodos hierárquicos é a possibilidade de um padrão poder mudar de agrupamento com a evolução do algoritmo e a possibilidade de se operar com instâncias maiores. Além disso, os métodos particionais são mais rápidos que os métodos hierárquicos (Man et al., 2001).

As principais desvantagens dos métodos particionais estão no fato de que, em alguns casos, o número de agrupamentos tem que ser escolhido *a priori*. Dessa forma, poderá sugerir interpretações erradas sobre a estrutura dos dados caso o número de agrupamentos não seja o ideal e no fato de que o algoritmo é em geralmente sensível às condições iniciais, podendo gerar resultados diferentes a cada rodada (Man et al., 2001). Sendo assim, o grau de similaridade entre os agrupamentos se resume ao grau de similaridades entre os elementos. Nesse caso, pode ser calculado através das medidas de distância vistas anteriormente, por exemplo, a distância euclidiana que foi utilizada nas meta-heurísticas proposta nesta dissertação.

Nesta seção, foram descritas algumas medidas de similaridades e métodos de particionamento. A métrica da função objetivo do nosso problema é a distância Euclidiana, pois faz o cálculo a partir da medida de similaridade entre os pontos (objetos). Assim, mostrou-se ser mais adequada para o nosso problema das distâncias *intra-clusters*.

2.4 Arquitetura de GPU e CPU

Recentemente, dispositivos massivamente paralelos (*manycore*) como as GPUs destacam-se como uma alternativa de alto poder computacional. As GPUs evoluíram a partir de aceleradores

gráficos dedicados. A arquitetura da GPU é organizada como uma matriz de multiprocessadores de *streaming* (*Streaming Multiprocessors* - SM) altamente encadeados e cada um contendo diversas unidades de processamento ou *cores* (Kirk & Wen-Mei, 2016), conforme esboçado na Figura 2.2.

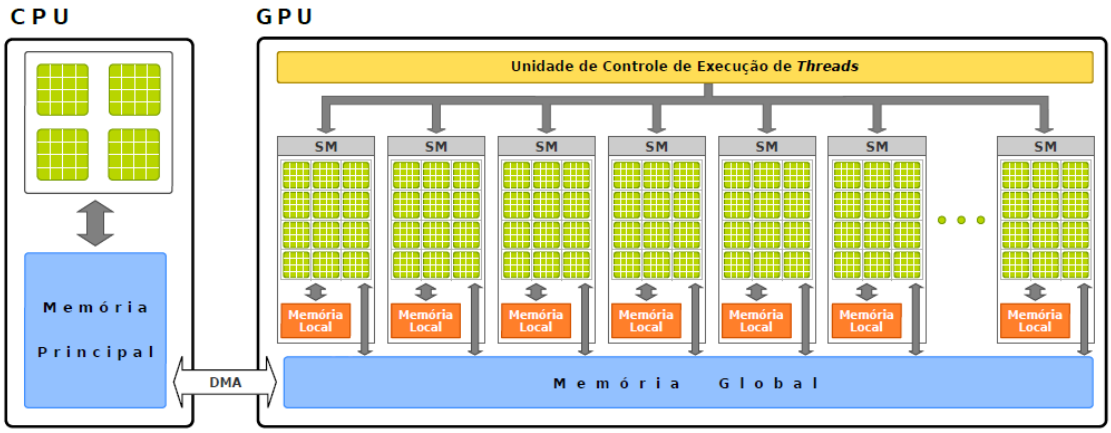


Figura 2.2: Arquitetura de CPU e GPU. Fonte: Rios (2016).

O paradigma da computação heterogênea e a computação em GPU não se destina a substituir a computação em CPU. Cada abordagem tem vantagens para certos tipos de programas. A computação da CPU é boa para tarefas que exigem muito controle, por sua vez, com a computação da GPU é possível executar tarefas intensivas em computação paralela (Cheng et al., 2014). A CPU é otimizada para cargas de trabalho dinâmicas marcadas por curtas sequências de operações computacionais e fluxo de controle imprevisível. As GPUs visam cargas de trabalho dominadas por tarefas computacionais com fluxo de controle simples. Como mostra a Figura 2.3, existem duas dimensões que diferenciam o escopo dos dispositivos para CPU e GPU:

- Nível de paralelismo, e
- Tamanho dos dados.

Deste modo, Cheng et al. (2014) exemplifica que se um problema tiver uma entrada de dados com tamanho pequeno, ou lógica de controle sofisticada e/ou paralelismo de baixo nível, a CPU é uma boa opção devido à sua capacidade de lidar com lógica complexa e paralelismo em nível de instrução. Se o problema em questão processa uma quantidade enorme de dados e exibe um paralelismo massivo de dados, a GPU é a escolha certa, pois possui um grande número de núcleos programáveis. A GPU pode suportar multi-encadeamento e possui uma largura de banda de pico maior em comparação com a CPU.

A arquitetura paralela e heterogênica composta por CPU + GPU ganharam espaço porque a CPU e a GPU possuem atributos complementares que permitem que os dispositivos tenham melhor desempenho usando os dois tipos de processadores. Portanto, para um bom desempenho é necessário definir qual dispositivo pretende utilizar a CPU ou GPU (Cheng et al., 2014).

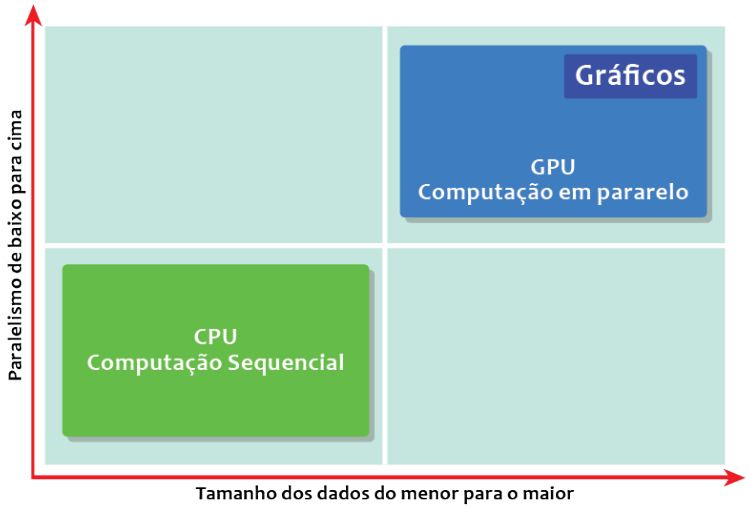


Figura 2.3: Carga de tarefas entre CPU e GPU. Fonte: Adaptado de Cheng et al. (2014).

Para executar tarefas em CPU utiliza-se partes sequenciais ou partes paralelas com os dados. Por sua vez, as tarefas intensivas podem ser executadas em paralelo nas partes da GPU, como mostra a Figura 2.4. Escrever códigos dessa maneira garante que as características da GPU e da CPU se complementem, levando à plena utilização do poder computacional do sistema combinado de CPU + GPU. A NVIDIA projetou um modelo de programação chamado CUDA para execução conjunta de CPU e GPU facilitando a programação em paralelo (Harris et al., 2007).

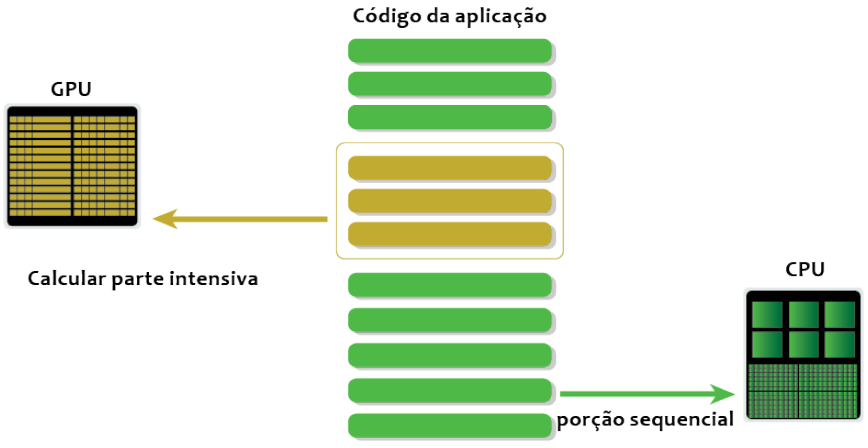


Figura 2.4: Execução sequencial em CPU e GPU. Fonte: Adaptado de Cheng et al. (2014).

Apesar das denominações *multicore* e *manycore* serem largamente empregadas para rotular, respectivamente, as arquiteturas CPU e GPU, é preciso ponderar porque os núcleos de processamento de cada plataforma possuem características distintas. Um núcleo ou (*core*) de CPU é

tipicamente mais robusto e projetado para tarefas lógicas e de controle mais complexas, sendo focado na execução de tarefas sequenciais. Um *core* GPU possui, relativamente, menor capacidade computacional sendo otimizado para o paralelismo de dados. Desta forma, um *core* GPU apresenta um esquema de controle lógico mais elementar, de modo que sua finalidade é favorecer o *throughput*[1] da aplicação paralela (Cheng et al., 2014).

Segundo Tan & Ding (2016), o *speedup* é uma métrica comumente utilizada pela comunidade de computação paralela. O *speedup* pode ser definido como a razão do tempo de execução da implementação paralela T_p e o tempo de execução da sequencial T_s, visto na Equação (2.8).

$$S = \frac{T_s}{T_p}. \tag{2.8}$$

2.5 Otimização

Otimização ou Programação Matemática é uma área da matemática caracterizada pela maximização ou minimização de uma função objetivo sobre variáveis de decisão, geralmente, modelando algumas aplicações reais existentes. Um conjunto de restrições limita os valores possíveis que as variáveis de decisão podem assumir, modelando as limitações reais da aplicação. Formalmente, é definido um problema de minimização da seguinte forma:

$$z = \min\{f(x) : x \in S \subseteq \mathbb{R}^n\}, \tag{2.9}$$

onde o conjunto S é a região viável definida pelas restrições do problema e a função objetivo $f(.)$ qualifica cada ponto da região. Uma solução x' é um *ótimo local* se $f(x') \leq f(x)$ para $\forall x \in V \subset S$, onde V é uma vizinhança de x'. Uma solução x^* é um *ótimo global* ou (solução ótima) se $f(x^*) \leq f(x), \forall x \in S$.

A natureza da função objetivo, as restrições e o domínio das variáveis de decisão caracterizam o tipo do problema tratado, bem como o grau de dificuldade e o conjunto de técnicas de resolução disponíveis. Assim, quando os problemas possuem função objetivo e restrições lineares sobre variáveis reais, temos um problema de *programação linear* e métodos como o *Algoritmo Simplex* (Dantzig, 1998) e o *Algoritmo de Pontos Interiores* (Karmarkar, 1984) possuem boa capacidade de resolução.

Porém, muitos problemas possuem natureza não-linear e as técnicas de resolução existentes geralmente conseguem tratar apenas instâncias muito pequenas. Por exemplo, em otimização contínua não-linear, os métodos de resolução geralmente realizam cálculos de gradiente em larga escala, que possuem custo computacional bastante elevado. Muitos destes problemas pertencem à classe $\mathcal{NP}$-difícil, dessa forma, não é conhecido algoritmo polinomial para resolvê-los. Esses problemas, podem ser tratados adequadamente por métodos heurísticos capazes de

[1] *Throughput* é a taxa de transferência entre CPUs e GPUs.

encontrar soluções de boa qualidade sem garantia de otimalidade (Gonzalez, 1982).

2.6 Meta-heurísticas e heurísticas

Uma meta-heurística é um algoritmo independente do problema. As meta-heurísticas fornecem um conjunto de diretrizes ou estratégias para desenvolver algoritmos de otimização heurístico Sörensen & Glover (2013). A implementação de um algoritmo para um problema específico de otimização é chamado de heurística de acordo com as diretrizes expressadas em uma meta-heurística. O termo meta-heurística foi criado por Glover (1986) e combina o prefixo grego *meta*-(metá, no sentido de *alto nível*) e heurística (do grego *heuriskein* ou *euriskein*, para busca).

Os algoritmos que não garantem que a solução é exata são chamados de algoritmos aproximados. Deste modo, basicamente, tais algoritmos são divididos em duas classes: algoritmos de aproximação e heurísticas. Os algoritmos de aproximação oferecem garantias com relação à qualidade das soluções encontradas. Para isso, eles são, geralmente, desenvolvidos especificamente para o problema tratado. No entanto, na maioria das vezes, a qualidade das aproximações fica distante da qualidade das soluções ótimas (Talbi, 2009).

Heurísticas não oferecem nenhuma garantia com relação à qualidade das soluções. Geralmente, as heurísticas se dividem em heurísticas específicas e meta-heurísticas. Como o nome diz, a primeira classe é desenvolvida especialmente para o problema abordado. Já as meta-heurísticas podem ser vistas como *frameworks* utilizados na resolução de problemas gerais (Sörensen & Glover, 2013). Nesse sentido, existe como alternativa a utilização de algoritmos que produzem soluções aproximadas, tais como heurísticas e meta-heurísticas. As vantagens de utilização desses métodos podem ser vistos na qualidade de suas soluções, na robustez dos algoritmos, e no tempo para encontrar soluções ótimas quando comparados aos métodos exatos. As heurísticas e meta-heurísticas são populares devido a sua implementação ter um baixo custo computacional para resolver problemas diversos na literatura.

Diferentemente das heurísticas específicas, as meta-heurísticas são aplicadas a problemas gerais e provêm de mecanismos de escape de ótimos locais (não globais). Dentre as meta-heurísticas mais utilizadas destacam-se: algoritmos genéticos, programação genética, *simulated annealing*, colônia de formigas, *Variable Neighborhood Search* (VNS), *terated Local Search* (ILS), *greedy randomized adaptive search procedure* (GRASP), *Tabu Search* (TS), entre outras. Cada método tem sua peculiaridade e uma forma diferente de escape de ótimos locais (Glover & Kochenberger, 2006). Embora meta-heurísticas possam produzir soluções de boa qualidade que envolvem menos recursos computacionais, o esforço empenhado na resolução de instâncias de médio e grande porte, em muitos problemas, podem ser desafiadores.

Existem duas características principais em uma meta-heurística que são a capacidade de diversificação e intensificação. A diversificação está relacionada a variabilidade de amostragem de soluções no espaço de busca e é importante na fuga de ótimos locais. A intensificação

consiste na capacidade de busca intensa do método de uma região considerada promissora do espaço de busca.

Conforme Gendreau & Potvin (2005), a ideia por trás do conceito de busca de intensificação é que se deve explorar mais profundamente os locais do espaços de busca que parecem promissores para garantir que as melhores soluções nessas áreas sejam de fato encontradas.

Em geral, a intensificação é baseada em alguma memória intermediária, como uma memória recente, na qual se registra o número de iterações consecutivas que vários componentes da solução foram presente na solução atual sem interrupção. Uma abordagem típica para intensificação é reiniciar a busca da solução mais conhecida atualmente e para corrigir os componentes que parecem mais promissores. Outra técnica que é frequentemente usada consiste em mudar a estrutura da vizinhança para outra mais promissora ou movimentos mais diversos (Gendreau & Potvin, 2005).

Atualmente, devido à presença ubíqua de paralelismo nos computadores modernos, o desenvolvimento de meta-heurísticas paralelas tornou-se corrente. Com efeito, podem ser encontradas nesse contexto diversas contribuições que buscam explorar o paralelismo presente tanto nos algoritmos quanto nas diversas plataformas de *hardware* existentes.

Neste contexto, a paralelização de meta-heurísticas surge não apenas para as estratégias para reduzir a execução do tempo computacional nas aplicações, mas também para fornecer robustez aos métodos e produzir resultados competitivos (Crainic & Toulouse, 2010). A eficácia de meta-heurísticas paralelas foi provada pela literatura em diversas aplicações do mundo real. Nesses experimentos, vários métodos exploram o paralelismo em algoritmos exatos/heurísticos e diferentes arquiteturas (Alba, 2005).

2.6.1 *Particle Swarm Optimization*

A meta-heurística *Particle Swarm Optimization* (PSO) ou otimização por nuvem de partículas, foi desenvolvida por Kennedy & Eberhart (1995). É uma técnica de otimização estocástica global inspirada no comportamento social de bandos de pássaros, peixes, formigas entre outros animais que ficam em bando. Conforme Lopes et al. (2013), este é um algoritmo de otimização modelado na teoria de enxame, onde a ideia principal de um PSO clássico é modelar o bando de pássaros voando em torno de um vale em uma região. No algoritmo PSO, as aves são substituídas por seres artificiais chamados de partículas.

No PSO, as partículas são os indivíduos da população. As partículas vão "voar" da posição atual para uma nova. A posição da melhor partícula da população será a melhor posição individual. De acordo com Talbi (2009), cada partícula possui uma influência social e uma influência com base em sua experiência, dessa forma, tradicionalmente, o PSO armazena a *melhor posição global* e, para cada partícula, sua *melhor posição individual*.

Com base nessas informações, cada partícula atualizará suas coordenadas no espaço de busca. A cada iteração, cada partícula muda sua posição de acordo com sua própria experiência

e de partículas vizinhas. As partículas do enxame estão voando através do espaço de solução de busca com uma velocidade para a formação de bandos de forma a atingir um ótimo local. A posição u e velocidade v da i-ésima partícula no espaço de solução d-dimensional de busca pode ser representada como segue as Equações (2.10) e (2.11):

$$v_i^{(t+1)} = I_n v_i^t + c_1 r_1 (p_i - u_i^t) + c_2 r_2 (p_g - u_i^t) \tag{2.10}$$

$$u_i^{(t+1)} = u_i^t + v_i^{(t+1)} \tag{2.11}$$

onde I_n é chamado de fator de ponderação de inércia; c_1 e c_2 são constantes chamados coeficientes de aceleração; r_1 e r_2 são dois números aleatórios independentes uniformemente distribuídos no intervalo $[0,1]$; p_i correspondente à melhor solução individual da partícula i obtido até o tempo t; enquanto que p_g representa, entre todas partículas, com a melhor posição encontrada até o instante t.

As constantes de aceleração dão peso ao aprendizado da partícula. O parâmetro c_1 é um fator de aprendizado cognitivo que representa a atração que uma partícula tem em direção ao seu próprio sucesso. Já o parâmetro c_2 é o aprendizado social, fator que representa a atração que uma partícula tem pelo sucesso de sua partícula vizinha. No procedimento de atualização de velocidade, um peso de inércia I_n é geralmente adicionado a velocidade anterior. O peso da inércia I_n controlará o impacto da velocidade do valor anterior para o atual. Para valores grandes no peso da inércia, o impacto das velocidades anteriores serão muito maiores que o atual. Assim, o peso da inércia representa uma troca entre a busca global e a busca local. Um grande peso de inércia estimula a busca global, ou seja, diversifica todo o espaço de busca, enquanto um peso menor de inércia incentiva a busca local, isto é, intensifica a busca na região atual.

A velocidade define a direção e a distância que a partícula devem ir, e pode ser vista na Figura 2.5. Pode-se observar os três componentes que determinam a próxima posição da partícula: velocidade anterior, aprendizado social e cognitivo.

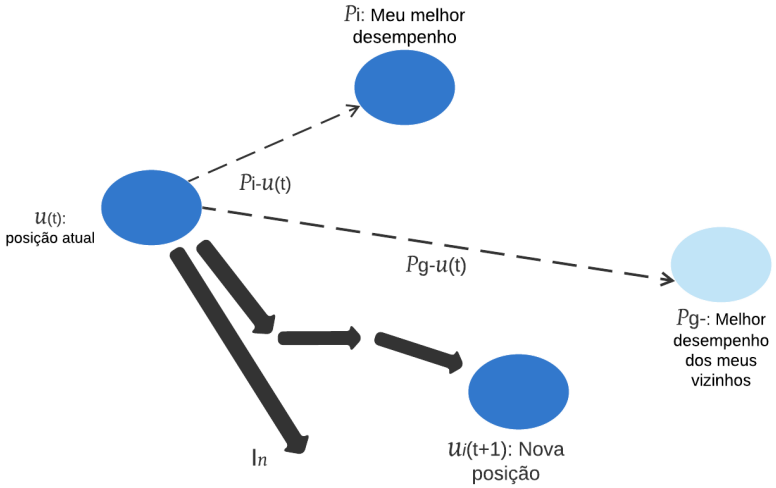


Figura 2.5: Movimento de uma partícula e atualização da velocidade. Adaptado de Talbi (2009).

Algoritmo 2.1 Modelo do algoritmo do *Particle Swarm Optimization* (PSO).

```
1:  procedure PSO
2:      Inicialização aleatória de toda população
3:      repeat
4:          for cada partícula i do
5:              if f(u_i) < f(p_i) then
6:                  p_i = u_i;
7:              end if
8:              if f(u_i) < f(p_g) then
9:                  p_g = u_i;
10:             end if
11:             Atualize a velocidade (Equação (2.10))
12:             Atualize a posição (Equação (2.11))
13:         end for
14:     until critérios de parada
15: end procedure
```

O Algoritmo 2.1 descreve a meta-heurística PSO. Na linha 2 é inicializada a população de modo aleatório. Nas linhas 5–6, para cada partícula i, é atualizada a melhor posição p_i visitada pela partícula i. Nas linhas 8–9, determina qual a melhor solução global p_g da nuvem de partículas. Nas linhas 11–12, é feita a atualização da velocidade e da melhor posição encontrada da partícula. Por fim, na linha 14 é definido o critério de parada do algoritmo PSO. Esses vários parâmetros também podem ser inicializados de forma dinâmica ou adaptativa para lidar com o *trade-off* entre intensificação e diversificação durante a busca.

2.6.2 *Differential Evolution*

A Evolução Diferencial (DE), é uma abordagem para problemas de otimização contínua que foi proposta por Storn (1995). A ideia principal do DE é usar diferenças entre vetores para perturbar a população de vetores. Essa ideia foi integrada a um novo operador de recombinação de duas ou mais soluções e um operador de mutação auto-referencial para direcionar a busca em direção a "boas" soluções. Como qualquer algoritmo evolutivo, o DE gera uma população inicial distribuída aleatoriamente P_0 de tamanho k $(k \geq 4)$. Cada indivíduo é um vetor real d-dimensional x_{ij}. Assim, cada elemento do vetor x_{ij} é gerado aleatoriamente da seguinte forma:

$$x_{ij} = x_j^l + rand_j[0,1] \cdot (x_j^h - x_j^l) \quad , i \in [1,k], j \in [1,m], \tag{2.12}$$

onde $rand_j$ é uma variável aleatória distribuída uniformemente no intervalo $[0,1]$, e os valores x_j^l e x_j^h representam o limite inferior e superior de cada variável, respectivamente.

O operador de recombinação DE fornece uma combinação linear em que o conceito de distância desempenha um papel importante. O Algoritmo 2.2 associa o operador de *crossover* ou cruzamento, dessa forma, é dado um pai i e três indivíduos selecionados aleatoriamente da população r_1, r_2 e r_3. O parâmetro F representa um fator de escala, $(F \in [0,1])$, enquanto o parâmetro CR representa uma probabilidade $(CR \in [0,1])$. Quando rand$_j$ $[0,1] < CR$ ou $j = j_{rand}$, a variável associada à prole será uma combinação linear de três soluções escolhidas aleatoriamente. Caso contrário, a variável descendente herdará o valor de seu pai. Desse modo, a condição $j = j_{rand}$ está incluída para garantir que pelo menos uma variável da prole seja diferente da mãe (para instância $CR = 0$). Contudo, o fator de escala F controla a amplificação da diferença entre os indivíduos r_1 e r_2 e é usado para evitar a estagnação do processo de busca no espaço de soluções.

A notação do algoritmo DE / x / y / z é geralmente usada para definir uma estratégia de DE, onde x especifica o vetor a sofrer mutação, que atualmente pode ser randômico (uma população escolhida aleatoriamente vetor) ou melhor (o vetor de menor custo da população atual), y é o número dos vetores de diferença usados, e z indica o esquema de cruzamento. O Algoritmo 2.2 mostra o modelo para o algoritmo *DE/rand/1/bin* básico, onde *bin* representa a variante de cruzamento binário. Uma substituição elitista é considerada, ou seja, a prole substituirá seu pai se seu função objetivo for melhor ou igual ao pai:

$$x_i(t+1) = \begin{cases} u_i(t+1), & \text{se } f(u_i(t+1)) \leq f(x_i(t)) \\ x_i(t), & \text{caso contrário,} \end{cases} \tag{2.13}$$

2.6.3 *Scatter Search*

A meta-heurística *Scatter Search (SS)*, ou denominada no português como busca de dispersão, tem sua origem no artigo de Glover (1977). A SS é uma estratégia que foi aplicada com sucesso a alguns problemas de otimização combinatória e contínua. A meta-heurística SS é evolutiva

Algoritmo 2.2 Modelo do Algoritmo DE/rand/1/bin.

```
1:  procedure DE
2:      Inicializar a população (distribuição aleatória uniforme)
3:      repeat
4:          for i ← 1 → k do
5:              Mutação/Crossover:
6:              j_rand = rand(m)
7:              for j ← 1 → m do
8:                  if rand() < CR or j == j_rand then
9:                      u_ij = x_r1j + F(x_r2j − x_r3j)
10:                 else
11:                     u_ij = x_ij
12:                 end if
13:             end for
14:             Substituição:
15:             if f(u_i(t+1)) ≤ f(x_i(t)) then
16:                 x_i(t+1) = u_i(t+1)
17:             else
18:                 x_i(t+1) = x_i(t)
19:             end if
20:         end for
21:     until Critério de parada
22: end procedure
```

e populacional que recombina soluções a partir de um conjunto de referência para criar outros conjuntos (Glover et al., 2003).

O método começa gerando um população *Pop*, onde seu passo inicial satisfaz os critérios de diversidade e qualidade na população. Considerando a referência do conjunto *(RefSet)* de tamanho moderado[2], então é construído e assim selecionando um bom representante dentro das soluções da população existente (Talbi, 2009). As soluções selecionadas são combinadas para fornecer soluções para um procedimento de melhoria baseado em uma solução na meta-heurística. De acordo com o resultado do procedimento, o conjunto de referência na população são atualizadas para incorporar as soluções de alta qualidade e bem diversificadas, tornando o processo é iterativo até que um critério de parada seja atendido. De qualquer modo, a abordagem da meta-heurística SS envolve diferentes procedimentos, permitindo gerar o valor inicial à população, construir e atualizar o conjunto de referência, combinar as soluções definidas para melhorar as soluções construídas e assim por diante. A SS usa explicitamente estratégias para intensificação de busca e diversificação de busca no espaço de soluções.

O algoritmo começa com um conjunto de diversas soluções, que representa o conjunto de referência, onde seria a (população inicial), conforme apresentado no Algoritmo 2.3. Este conjunto de soluções é desenvolvido por meio de recombinação de soluções e também por aplicação

[2]Tipicamente, o conjunto de referência é composto por mais ou menos 10 soluções.

de busca local. Cada etapa do Algoritmo 2.3 é explicada a seguir:

Algoritmo 2.3 Modelo do algoritmo *scatter search*.

```
1:  procedure SS
2:      Inicializa a população (Pop) usando um método de geração de diversificação;
3:      Aplique o método de melhoria à população;
4:      Método para atualizar conjunto de referências;
5:      repeat
6:          Método de geração de subconjuntos;
7:          repeat
8:              Método de combinação de solução;
9:              Método de melhoria;
10:         until Critério de parada 1
11:         Método de atualização do conjunto de referência;
12:     until Critério de parada
13: end procedure
```

- **Método de geração de diversificação:** esse método gera um conjunto de soluções iniciais. Em geral, procedimentos gulosos são aplicados para diversificar a busca enquanto seleciona soluções de alta qualidade.

- **Método de atualização do conjunto de referências:** nesse componente de busca, um conjunto de referências é construído e mantido. O objetivo é garantir a diversidade, mantendo soluções de alta qualidade. Por exemplo, pode-se selecionar soluções *RefSet1* com a melhor função objetivo e, em seguida, adicionar soluções *RefSet2* com a melhor diversidade *(RefSet = RefSet1 + RefSet2)*.

- **Método de geração de subconjunto:** esse método opera no conjunto de referência *RefSet*, produzindo um subconjunto de soluções como base para a criação de soluções combinadas. Geralmente, seleciona todos os subconjuntos de tamanho fixo r (em geral, $r = 2$).

- **Método de combinação de soluções:** nesse método um determinado subconjunto de soluções produzido pelo método de geração de subconjunto são recombinados. Em geral, as estruturas são ponderadas e suas combinações são usadas através de combinações lineares e arredondamentos generalizados para variáveis discretas.

Os três componentes de busca (método de geração de diversificação, método melhoria e método de combinação) são problemas específicos, enquanto as outras duas buscas por componentes (método de atualização do conjunto de referência, método de geração de subconjunto) são problemas genéricos. Esses métodos são combinados como mostrado no Algoritmo 2.3.

2.6.4 *Greedy Randomized Adaptative Search Procedure* - GRASP

O trabalho de Feo & Resende (1995) apresenta a meta-heurística denominada como *Greedy Randomized Adaptive Search Procedures* (GRASP). Essa é uma meta-heurística *multi-start* para problemas de otimização combinatória. Cada iteração consiste basicamente de duas fases: construção e busca local. A fase de construção cria uma solução viável, onde sua vizinhança é investigada até que um ótimo local seja encontrado durante a fase de busca local. Desse modo, a melhor solução geral é mantida como resultado.

A meta-heurística GRASP é utilizada em algumas áreas da computação, isso pode ser visto a partir de anotações bibliográficas descritas no trabalho de Festa & Resende (2009). Observa-se que o GRASP tem sido aplicado a uma grande variedade de problemas de otimização industrial e de pesquisa operacional. Isto inclui: problemas em escalonamento (Rajkumar et al., 2011), roteamento (Duhamel et al., 2010), lógica (Pardalos & Resende, 2002), particionamento (Pu et al., 2006), localização de *layout* (Abdinnour-Helm & Hadley, 2000), teoria dos grafos (Ribeiro et al., 2002), atribuição (Ahuja et al., 2000), manufatura (Laguna & Velarde, 1991), trasporte (Lourenço et al., 2001), telecomunicações (Resende & Ribeiro, 2003), sistemas de potência elétrico (Faria et al., 2005), entre outros.

No Algoritmo 2.4 é apresentado um pseudocódigo do algoritmo GRASP. A cada iteração é feita a construção de uma solução gulosa aleatória e, em seguida, é aplicada a busca local. Por fim, é retornado o melhor valor da solução que foi encontrada.

Algoritmo 2.4 Pseudocódigo do GRASP

```
1:  procedure GRASP(problema P, instância i)
2:      f(s*) ← ∞;
3:      while Condição de parada não for satisfeita do
4:          s ← solução_gulosa_aleatória(α);
5:          s ←busca_local(s);
6:          if f(s) < f(s*) then
7:              s* ← s;
8:          end if
9:      end while
10:     return s*;
11: end procedure
```

Por sua vez, no Algoritmo 2.5 é feita a fase de construção das soluções. Nessa etapa, é criada uma Lista Restrita de Candidatos (LRC) e atribui-se os melhores candidatos (elemento/parte da solução) à LRC. Em seguida, um candidato é escolhido aleatoriamente da LRC e adicionado na solução corrente. Tal processo construtivo é denominado como algoritmo semi-guloso e foi proposto por Hart & Shogan (1987).

Algoritmo 2.5 Método construtivo da meta-heurística GRASP

```
1: procedure SOLUCAO_GULOSA_ALEATORIA(PROBLEMA P, INSTÂNCIA i);
2:     solução ← ∅;
3:     while Construção da Solução não Concluída do
4:         Crie a RCL;
5:         s ← SelecioneElementoAleatoriamente(LRC);
6:         solução ← solução ∪{s};
7:     end while
8:     return solução;
9: end procedure
```

Capítulo 3

REVISÃO DA LITERATURA

Neste capítulo, são apresentados os trabalhos mais proeminentes relacionados ao trabalho proposto. O capítulo está dividido em duas partes. A primeira parte descreve alguns métodos heurísticos para clusterização de dados. A segunda parte mostra alguns trabalhos relacionados com métodos heurísticos para clusterização em GPU.

3.1 Métodos heurísticos para clusterização de dados

Nesta seção, são descrito alguns artigos que fazem uso de heurísticas e meta-heurísticas para problemas de clusterização de dados. Alguns trabalhos são meta-heurísticas híbridas, desse modo, são feitas algumas adaptações para combinar técnicas existentes a fim de encontrar a melhor solução.

Nos trabalhos dos autores Queiroga et al. (2017, 2018) é proposto um algoritmo de clusterização de dados denominado como *C-GRASP-Clu* baseado na heurística *Continuous Greedy Randomized Adaptive Search Procedure* (C-GRASP). Neste sentido, é proposto uma abordagem para resolver problemas de clusterização que visa minimizar a distâncias *intra-clusters*. A cada iteração, a sub-rotina solução inicial gera uma solução inicial x. Esta etapa é baseada no esquema de inicialização do *k-means*. Atualmente a meta-heurística C-GRASP-Clu é a melhor meta-heurística referente a problema de clusterização de dados com objetivo de minimizar a soma das distâncias intra-*clusters*.

Wang & Sun (2016) apresentam um algoritmo híbrido de *k-means* baseado em SA-PSO. Tal algoritmo apresenta os princípios relativos do algoritmo *k-means*, do algoritmo de *Simulated Annealing* (SA) e algoritmo de *Particle Swarm Optimization* (PSO). Em seguida, é vista a influência do valor inicial do algoritmo *k-means* na solução ótima do algoritmo. Assim, com a otimização global, o novo algoritmo pode superar a deficiência do algoritmo *k-means* que fica preso em mínimos locais.

Krishnasamy et al. (2014) apresentam um algoritmo de agrupamento de dados evolucionário híbrido eficiente referido como K-MCI, por meio do qual é combinado *k-means* com inteligên-

cia de grupo modificada. O algoritmo proposto foi testado em vários conjuntos dados do UCI *Machine Learning Repository* e seu desempenho é comparado com outros algoritmos bem conhecidos como *k-means*, *k-means* ++, *Cohort Intelligence* (CI), *Modified Cohort Intelligence* (MCI), *Genetic Algorithm* (GA), *Simulated Annealing* (SA), *Tabu Search* (TS), *Ant Colony Optimization* (ACO), *Honey Bee Mating Optimization* (HBMO) e *Particle Swarm Optimization* (PSO). Os resultados experimentais mostram que a abordagem é promissora em termos de qualidade da solução e velocidade de convergência do algoritmo.

Em Tsai & Kao (2011), os autores apresentam uma otimização da regeneração seletiva por enxame de partículas (SRPSO). Um novo algoritmo foi desenvolvido com base na *Particle Swarm Optimization* (PSO). Ele contém dois novos recursos, parâmetro não balanceado, configuração e operação de regeneração de partículas. A configuração de parâmetro desequilibrado permite uma convergência rápida do algoritmo e a operação de regeneração de partículas permite a busca para escapar de ótimos locais e explorar melhores soluções. O algoritmo é aplicado para o problemas de agrupamento de dados com o desempenho e um algoritmo híbrido (KSRPSO), o método de agrupamento é baseado no algoritmo *k-means* e SRPSO. Dentro dos experimentos conduzidos, SRPSO e KSRPSO são comparados com o algoritmo PSO original, *k-means*, bem como, outros métodos propostos por outros estudos. Os resultados demonstram que SRPSO e KSRPSO são métodos eficientes, precisos e robustos para problemas de agrupamento de dados.

Xu et al. (2010), utilizam um PSO e DE híbrido, conhecido como o *Particle Swarm Optimization* e *Differential Evolution* (DEPSO), a fim de melhorar ainda mais a capacidade de busca e obter maior flexibilidade na exploração de estruturas nas instâncias. Os resultados empíricos mostram que o algoritmo de agrupamento baseado em DEPSO atinge um melhor desempenho com a função de aptidão do que em outras abordagens usadas. Outros estudos experimentais em conjuntos de dados sintéticos e reais demonstram a eficácia do método proposto em encontrar soluções de agrupamento significativas.

3.2 Métodos heurísticos para clusterização em GPU

Nesta seção serão descritos alguns trabalhos referentes a a clusterização de dados em GPU.

No artigo de Krömer et al. (2012), é apresentado uma implementação de um algoritmo genético para clusterização em densidade. O trabalho mostra um método de clusterização baseado em GPU usando a plataforma (NVIDIA CUDA). É feito um agrupamento baseado em densidade utilizando o índice de Dunn. O índice foi implementado no algoritmo para ser executado em GPU, onde mostrou ser 5x superior no tempo de processamento comparado ao algoritmo sequencial (CPU).

No trabalho de Kohlhoff et al. (2011) são implementados algoritmos de clusterização para GPU denominando a *library* de *CAMPAIGN*. A *library* é implementada especificamente para

execução massivamente paralelo em GPU. Dessa forma, foram implementado 5 variações do algoritmo *k-means*, cada versão é executada em CPU e as versões aceleradas em GPU. Por fim, concluiu que o algoritmo *k-centers* teve *speedup* médio de 177.9x comparado com a versão sequencial.

No trabalho Kazakovtsev et al. (2019), os autores propõem um algoritmos heurístico de agrupamento *k-means*. O estudo faz implementação de uma heurística gulosa randomizada em (GPU). Os experimentos computacionais ilustram o alto desempenho das GPUs em comparação com a execução dos algoritmos heurísticos gulosos em (CPU). Nos casos com grandes conjuntos de dados e número de *clusters*, a GPU teve os melhores desempenho. Tal trabalho concluiu que o uso de GPU mostra um vantagem na velocidade alcançada em comparação com os cálculos na CPU, a qual aumenta com grandes conjuntos de dados e um grande número de *clusters*.

No trabalho de Farivar et al. (2008), os autores apresentam uma implementação paralela eficiente em GPU. Foi desenvolvido um algoritmo de clusterização *k-means* usando as extensões de processamento CUDA. A implementação do algoritmo *k-means* acelerada por CUDA foi divida em três estágios distintos de operações. O primeiro estágio inicializa o *hardware CUDA*. Segundo, aloca o *host* apropriado na memória do dispositivo. Por fim, armazena, o conjunto inicial de centroides e carrega o conjunto de dados na memória *on-board* da GPU. Os resultados computacionais mostram que dispositivos como a GPU pode usar os recursos ociosos oferecidos pelo sistema por meio da arquitetura CUDA.

O trabalho de Zechner & Granitzer (2009) faz uma implementação otimizada do algoritmo *k-means* em GPU utilizando o NVIDIA's (CUDA). O algoritmo foi desenvolvido de maneira híbrida, paralelizando cálculos de distância na GPU enquanto atualiza sequencialmente centroides de *cluster* na CPU com base nos resultados do cálculos de GPU. Para observar a influência do *k-means*, foram criadas instâncias como o número de pontos de dados (500, 5000, 50.000 e 500.000) para comparar o desempenho em GPU e CPU. Por fim, o trabalho concluiu que o desempenho com os dados fornecidos, demonstram um *speedup* médio de 14x em comparação a implementação de CPU.

O trabalho de Sirotković et al. (2012) apresenta uma implementação do algoritmo *k-means* para o problema de segmentação de imagem na plataforma GPU com CUDA. A segmentação paralela do *k-means* é realizada de maneira híbrida, ou seja, a abordagem proposta distribui a carga de computação entre (CPU) e a (GPU). A ênfase é colocada na adaptação do algoritmo para processar de forma eficiente as características da segmentação de imagem no conjuntos de dados para explora os benefícios da arquitetura de GPU. Os resultados experimentais envolvendo a segmentação de imagens, mostraram 2.3x melhor e em 600x melhor no tempo de execução em comparação com o versão sequencial. Os experimentos demonstraram que a execução é consideravelmente mais rápida com a abordagem proposta em GPU em comparação com CPU.

Embora existam alguns métodos heurísticos que usam GPU para clusterização de dados.

vale ressaltar que na literatura referente ao problema foram encontrados poucos trabalhos. Até o conhecimento dos autores, não foi encontrado nenhuma meta-heurística em GPU com a função objetivo aqui utilizada.

Capítulo 4

METODOLOGIA

Neste capítulo, são descritos os passos metodológicos e as ferramentas utilizadas para obter os resultados deste trabalho. Esta pesquisa foi fundamentada nos preceitos da pesquisa descritiva e aplicada. É descritiva, pois relata a situação prática encontrada comparando às mudanças geradas pelas futuras aplicações de um método de otimização. É aplicada, pois gera resultados que possam ser aplicados, visando melhorias. Na Figura 4.1 é descrito o Fluxograma do processo metodológico utilizado nesta dissertação. As seções seguintes detalham as etapas desse fluxograma.

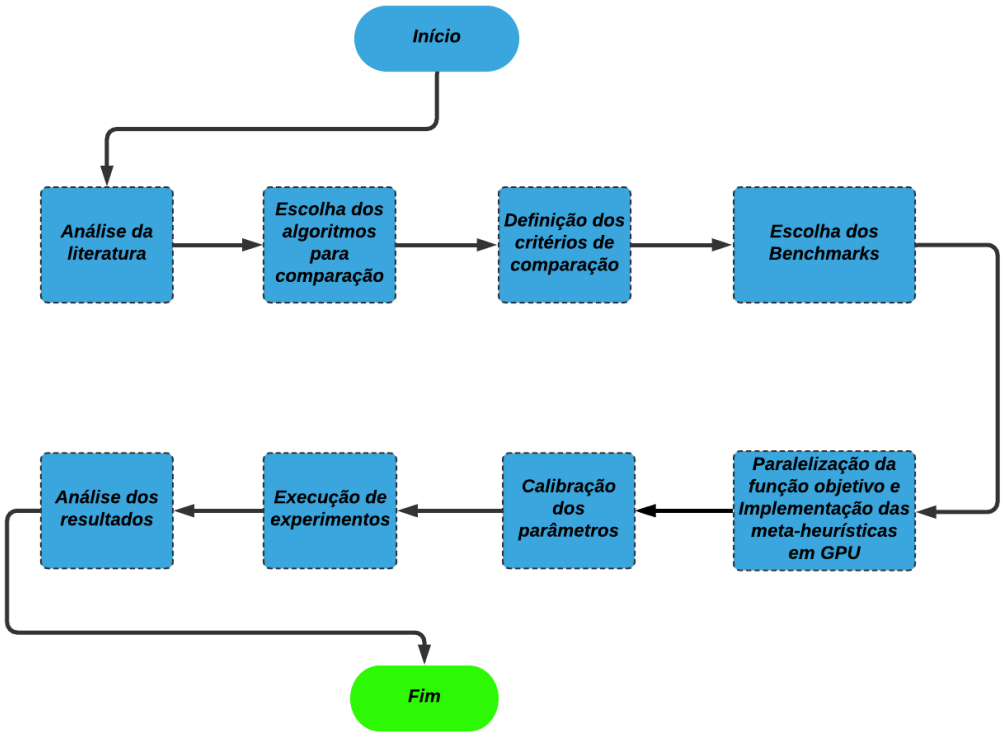


Figura 4.1: Fluxograma do processo metodológico. Fonte: Autor.

4.1 Análise da literatura

Após a definição dos objetivos da pesquisa, foram definidas buscas de trabalhos relacionados à clusterização de dados na literatura. O próximo passo foi estabelecer os critérios dos trabalhos que tiveram mais semelhanças com a pesquisa, tais trabalhos são relacionados a métodos de clusterização de dados, meta-heurísticas em GPU, estudos comparativos entre meta-heurísticas e otimização de sistemas computacionais. Os trabalhos podem ser vistos no Capítulo 3. A análise da literatura levou em consideração as seguintes bases de dados: *Springer ACM Digital Library*, *IEEEXplore Digital Library* e *Google Scholar*.

Também foram conduzidas buscas na literatura com o propósito de encontrar meta-heurísticas em GPU para o problema de clusterização de dados. Ao longo das buscas de trabalhos relacionados sobre meta-heurísticas em GPU, vale ressaltar que ainda existem poucos trabalhos de pesquisa relacionados à evolução de algoritmos para essa arquitetura.

4.2 Escolha dos algoritmos para comparação

Foram escolhidas as meta-heurísticas GPU-PSO, GPU-DE e GPU-SS e a meta-heurística C-GRASP-Clu que é o algoritmo sequencial mais competitiva para o problema de agrupamento usando a distância *intra-clusters*, o C-GRASP-Clu é estado da arte para problemas de clusterização. O código-fonte do C-GRASP-Clu e as instâncias podem ser encontrados no *link*[1].

Foram implementadas adaptações nas meta-heurísticas em GPU após um estudo comparativo entre as meta-heurísticas mais proeminentes da literatura para o problema de clusterização de dados. Foi adaptada a função objetivo para as três seguintes meta-heurísticas: (PSO) (Kennedy, 1995), (DE) (Storn & Price, 1997) e (SS) (Glover et al., 2003). A implementação de versões paralelas dessas meta-heurísticas são conhecidas pela literatura e, existe o *framework* `libCudaOptimize`[2] que fornece uma biblioteca em GPU para implementá-las (Nashed et al., 2012).

4.3 Definição dos critérios de comparação e escolha do *benchemarks*

Foi comparado o desempenho das meta-heurísticas em GPU com C-GRASP-Clu (Queiroga et al., 2018). Ficou definido como critério de parada os tempos de 1s, 10s, 30 e 60s. Consideramos esse critério de tempos conforme a literatura em Queiroga et al. (2018). Cada algoritmo foi executado 50 vezes em cada um dos tempos, pois os algoritmo possuem elementos estocásticos. Após isso foi identificada a melhor solução *(Best)* sendo a melhor solução geradas em cada

[1] Acesso em: 12 de Fevereiro de 2019. https://doi.org/10.1016/j.asoc.2018.07.031
[2] Acesso em: 05 de março de 2019. https://sourceforge.net/projects/libcudaoptimize/

limite de tempo, após isso calculam-se as médias do melhor caso de todas as soluções com os casos obtidas nas execuções.

Foi utilizado para os experimentos o conjunto de 20 instâncias de *benchmark* para o problema de clusterização. As principais características das instâncias são descritas na Tabela 4.1, contendo o número de Atributos, número de *Clusters* e o número de Objetos. As instâncias *iris*, *wine*, *cancer*, *cmc*, *glass*, *ionosphere*, *wdbc*, *yeast*, *abalone* foram obtidas do repositório de instâncias e podem ser encontradas em *University of California, Irvine(UCI)*[3]. Por sua vez, as instâncias *vowel* e *crudeoil* são originadas respectivamente em S.K. Pal (1977) e Gerrild & Lantz (1969), respectivamente. Dois conjuntos de dados artificiais, *artset1* e *artset2* são instâncias artificias da literatura (Niknam & Amiri, 2010; Ahmadi, 2012; Wang et al., 2016). As instâncias *a1*, *D31*, *s1* e *unbalance* foram obtidas em Fränti & Sieranoja (2015). Por fim, as instâncias *artset3* e *artset4* são instâncias artificias descrita na literatura por Pedregosa et al. (2011).

Tabela 4.1: Principais características dos conjuntos de dados.

Instâncias:	Nº de Atributos (d)	Nº de *Clusters* (k)	Nº de Objetos (m)
Fisher's Iris (iris)	4	3	150
Wine (wine)	13	3	178
Indian Telugu Vowel Sounds (vowel)	3	6	871
Contraceptive Method Choice (cmc)	9	3	1473
Breast Cancer Wisconsin 1 (cancer)	9	2	683
Glass Identification (glass)	9	6	214
Crude Oil (crudeoil)	5	3	56
Thyroid Disease (thyroid)	5	2	215
Artificial Set One (artset1)	3	5	250
Artificial Set Two (artset2)	2	4	600
Artificial Set Three (artset3)	2	3	1500
Artificial Set Four (artset4)	3	4	1750
Ionosphere (ionosphere)	34	2	351
Breast Cancer Wisconsin 2 (wdbc)	30	2	569
Yeast (yeast)	8	10	1484
A1 (a1)	2	20	3000
D31 (D31)	2	31	3100
S1 (s1)	2	15	5000
Unbalance (unbalance)	2	8	6500
Abalone (abalone)	3	8	4177

[3]http://archive.ics.uci.edu/ml/index.php

4.4 Paralelização da função objetivo e implementação das meta-heurísticas em GPU

Esta seção descreve a implementação da função objetivo assim como a implementação das meta-heurísticas mais proeminentes da literatura em GPU. Para implementar essas meta-heurísticas, a implementação foi dividida em duas partes: a parte independente do problema e a parte dependente do problema. A parte independente do problema se refere aos operadores de seleção, reposição e combinação de cada meta-heurística, enquanto a parte dependente se refere à avaliação da função objetivo.

4.4.1 Parte independente

Na parte independente do problema, os operadores independentes de (PSO), (DE) e (SS), foram implementados usando o *framework* `libCudaOptimize` desenvolvido por Nashed et al. (2012). Esta estrutura fornece uma biblioteca de código aberto de meta-heurísticas baseadas em GPU para problemas de otimização global contínua (CGOP) sujeitos a restrições de caixa.

Cada uma das meta-heurísticas implementadas possui parâmetros específicos que precisam ser configurados, como taxa de cruzamento, número de população, etc. Como será mostrado na Seção 4.5, os parâmetros foram configurados usando a ferramenta *irace* (López-Ibáñez et al., 2016).

4.4.2 Parte dependente

Na parte dependente do problema, a função objetivo das meta-heurísticas foi criada transformando o problema de agrupamento em um CGOP da seguinte forma:

(i) Cada solução do CGOP corresponde à posição de k centroides no espaço de solução;

(ii) As restrições da caixa são dadas pelos valores máximo e mínimo para cada dimensão de d;

(iii) A função objetivo do CGOP corresponde à soma das distâncias intra-*cluster*.

Como a adaptação dos itens (i) e (ii) é trivial, a seguir é apresentado um algoritmo em GPU para calcular o item (iii). O Algoritmo 4.1 mostra a implementação do kernel da GPU proposta para a avaliação da distância *intra-cluster*. Este *kernel* é chamado com N blocos de B *threads*, onde N é o número de indivíduos na população, e B é o valor mais alto que satisfaz as seguintes condições: (i) é uma potência de 2, e (ii) é menor ou igual a $\min(m, 1024)$ (m é o número de pontos). Cada grupo de *threads* é responsável por realizar uma única avaliação de função de *fitness*, e cada *thread* em um bloco é responsável por calcular a distância *intra-cluster* de m/B pontos.

Algoritmo 4.1 Kernel *GPUCUDA* para cálculo da função objetivo.

```
1:  procedure FO_KERNEL(d_solutions, d_fitness, k, d_points, m, d)
2:      declare int n = k * d;
3:      declare float x[n] in shared memory;
4:      declare int B                                   ▷ tamanho do bloco consultado no tempo de execução CUDA
5:      declare int solutionId                          ▷ ID de bloco de thread consultado no tempo de execução CUDA
6:      declare int alleleId                            ▷ ID do thread consultado no tempo de execução CUDA
7:      declare int allelePos = (solutionId * n);
8:      if (alleleId < n) then
9:          x[alleleId] = d_solutions[allelePos + alleleId];
10:     end if
11:
12:     __syncthreads();
13:     declare float min_dist_total = 0;
14:     for (int point=alleleId; point < m; point=point + B) do
15:         declare float min_dist = MAX_FLOAT;
16:         for (int m = 0; m < k ; ++m) do
17:             declare float distance = 0.0;
18:             for (int dim = 0; dim < d ; ++dim) do
19:                 i = m * dim + point;
20:                 j = k * m + dim;
21:                 distance += powf(fabs(d_points[i] - x[j]), 2);
22:             end for
23:             distance = sqrtf(distance);
24:             if (distance < min_dist) then
25:                 min_dist = distance;
26:             end if
27:         end for
28:         min_dist_total += min_dist;
29:     end for
30:     __syncthreads();
31:     float result = BlockSumReduce(min_dist_total);
32:     if (alleleId ==0) then
33:         fitness[solutionId] = result;
34:     end if
35: end procedure
```

O *kernel* recebe como entrada o vetor com as coordenadas de cada candidato, a solução ($d\_solutions$), onde o vetor onde é armazenado cada valor de aptidão ($d\_fitness$), o número de *clusters* (k), o vetor com as coordenadas dos pontos (d_points), o número de pontos (m) e a dimensão dos pontos (d). O vetor de solução $d\_solutions$ tem tamanho $k*d$, tal que $d\_solutions[i]$ é a coordenada $(i\%d)+1$ do *cluster* $\lfloor i/d \rfloor + 1$. O vetor de aptidão $d\_fitness$ tem o tamanho N e $d\_fitness[i]$ armazena o valor de aptidão do indivíduo $i+1$. Finalmente, d_points tem o tamanho D e, para uma melhor localidade no acesso à memória, consideramos que $d\_points[i]$ é a coordenada $\lfloor i/m \rfloor + 1$ do ponto $(i\%m)+1$.

O Algoritmo 4.1 referente ao cálculo da função objetivo. Após inicializar algumas variáveis (linhas 2–7), o *kernel* carrega a solução codificada na na memória *scratchpad* da GPU (linhas 8–9). Em seguida, ele calcula a distância *intra-cluster* de m/B pontos (linhas 12–30). Depois que cada *thread* calcula suas distâncias *intra-cluster* associadas, na linha 31, o *kernel* usa um operador $BlockReduceSum$ para somar todas as distâncias *intra-cluster*. Finalmente, ele escreve o cálculo resultante no vetor de *fitness* (linhas 32–34).

4.5 Calibração dos parâmetros

Para encontrar uma configuração adequada dos parâmetros nas meta-heurísticas GPU-PSO, GPU-DE e GPU-SS, foi utilizado o pacote *irace* [4]. Seu principal objetivo é configurar automaticamente os algoritmos de otimização, localizando as configurações mais apropriadas (López-Ibáñez et al., 2016). Na busca para o melhor parâmetro nas meta-heurísticas foram traçadas as estratégias de executar todas as instâncias para cada meta-heurística e logo após a execução de todas as instâncias ficando assim apenas um único parâmetro final.

A Tabela 4.2 indica a faixa de valores testados pelo *irace* para cada meta-heurísticas. Foi utilizado o valor máximo de experimentos considerado em *maxExperiments* = 20000. O procedimento de calibração é descrito na Tabela 4.2.

[4] *Package irace: https://cran.r-project.org/web/packages/irace/index.html*

Tabela 4.2: Procedimento de busca dos melhores parâmetros no *Package Irace*.

Meta-heurísticas:	Parâmetros:	Valores:
Particle Swarm Optimization:	I_n	**[0,1;5,0]**
	$C1$	**[0,1;5,0]**
	$C2$	**[0,1;5,0]**
Differential Evolution:	CR	**[0,0;3,0]**
	F	**[0,0;3,0]**
	tipo de *crossover*	**{Binomial, Exponencial Crossover_num}**
	tipo de mutação	**{DE_Random, DE_Best, DE_Target_To_Best, DE_Mutation_Num}**
Scatter Search:	b_1	**[0,1;10]**
	b_2	**[0,1;10]**
	λ	**[0,1;1]**
	interações de busca local	**{1,2,...,10}**
	usar diversificação	**{falso, verdadeiro}**
	divisão da grade	**[0,1;10]**

Os melhores parâmetros encontrados pelo *irace* para cada meta-heurística GPU-PSO, GPU-DE e GPU-SS são apresentados na Tabela 4.2. O valor do parâmetro N determina o número de indivíduos da população. Desta forma, o valor do N foi configurado manualmente sem o auxílio do *framework irace*. A escolha para o valor do N em cada meta-heurística foi dado a partir de análise de desempenho nos seguintes valores de indivíduos (128, 256, 512, 1024). Os parâmetros utilizados no *C-GRASP-Clu*, os mesmos foram retirados do trabalho de Queiroga et al. (2018).

Tabela 4.3: Parâmetros utilizados nas meta-heurísticas.

Meta-heurísticas	Parâmetros
Particle Swarm Optimization:	I_n = **0,1**
	$C1$ = **0,1**
	$C2$ = **1,7**
	N = **1024**
Differential Evolution:	CR = **0,4**
	F = **0,9**
	crossover exponencial = 1
	(tipo de mutação) melhor alvo = 2
	N = **256**
Scatter Search:	b_1 = **1**
	b_2 = **1**
	λ = **0,5**
	interações de busca local = **1**
	usar diversificação = false
	divisão da grade = **1**
	N = **256**

4.6 Execução dos experimentos

Durante a execução dos experimentos, buscou-se responder as seguintes perguntas elencadas abaixo segundo Barr et al. (1995). Conforme será relatado no próximo Capítulo 5.

1. Qual é a qualidade da melhor solução encontrada?
2. Quanto tempo leva para determinar a melhor solução?
3. Com que rapidez o algoritmo encontra boas soluções?
4. Quão robusto é o método?
5. Quão "distante" está a melhor solução daquelas mais facilmente encontradas?
6. Qual é a compensação entre viabilidade e qualidade da solução?

4.7 Análise dos resultados

Conforme Barr et al. (1995) esta fase do experimento converte os dados coletados em informações por meio de análise e interpretação. A análise de dados refere-se à avaliação dos dados empíricos registrados com técnicas estatísticas e técnicas não estatísticas com relação ao propósito e aos objetivos do experimento. Neste ponto, os dados necessários devem estar disponíveis

para testar as hipóteses, estimar a população parâmetros, descobrir relações pertinentes, verificar suposições e medir a variabilidade.

Dessa forma, devem ser abordados os objetivos experimentais e as questões que os pesquisadores colocaram. Em particular, deve-se considerar as principais compensações (como qualidade da solução versus tempo, velocidade versus robustez) e tentar identificar os fatores ou combinações de fatores que parecem contribuir e que estão relacionados aos desempenhos dos algoritmos.

4.8 Ambiente experimental

As meta-heurísticas utilizadas nos experimentos foram implementadas em C++ utilizando a biblioteca NVIDIA CUDA versão 11.1 compiladas com o NVCC (NVIDIA C *Compiler*) e g++ 9.3.0. Com sistema operacional de 64-bits executando Linux Ubuntu com a versão 20.4.1. e processador Intel Corei7-4790 CPU @ 3.60GHz e 16GB de memória RAM. Essa máquina está equipada com uma placa de GPU NVIDIA GeForce GTX 780 Ti de 3GB.

Os experimentos seguiram o protocolo experimental adotado na literatura (Queiroga et al., 2018). Dessa forma, cada meta-heurística foi executada 50 vezes em cada instância, considerando os seguintes tempos limites:1s, 10s, 30s e 60s. Os resultados experimentais dos algoritmos foram avaliados de acordo com a metodologia apresentada em Barr et al. (1995). Isto é, a avaliação foi feita em termos de robustez, qualidade das soluções encontradas e esforços computacionais.

Capítulo 5

RESULTADOS COMPUTACIONAIS

Neste capítulo, são apresentados os resultados da comparação entre as meta-heurísticas propostas e C-GRASP-Clu. Neste capítulo também são reportados os resultados refentes ao *speedup* da função objetivo. Por fim, são apresentados os resultados dos testes estatísticos de *Friedman*.

As Tabelas 5.1 e 5.2 mostram os resultados referente ao tempo limite de 60s segundos, enquanto nas Tabelas 5.3 e 5.4 refere-se ao tempo de 30s segundos. Os resultados mostram para cada instância de dados o melhor *(Best)*, a média e os valores de *gap* relativos a cada meta-heurística. A coluna relativa ao *Gap* é calculada como $gap = 100 \times (s - s^*)/s^*$, onde s^* é a distância *intra-cluster* da melhor solução e s é o *intra-cluster* distância da melhor solução encontrada pelo método. A parte inferior das tabelas mostra um resumo que inclui: a coluna média e a coluna máxima dos *Gaps*. As demais, Tabelas 5.5, 5.6, 5.7 e 5.8 referem aos tempos limites de: 1s e 10s.

Tabela 5.1: Resultados obtidos com o tempo limite de 60 segundos.

Instâncias:	Critério	C-GRASP-Clu	GPU-PSO	GPU-DE	GPU-SS
iris:	Best	96,65548	**96,65547**	**96,65547**	**96,65547**
	Média	96,65548	**96,65547**	**96,65547**	109,22327
	Gap.	0,00%	0,00%	0,00%	0,00%
wine:	Best	16292,18464	**16292,18359**	**16292,18359**	16292,18750
	Média	**16292,18464**	16292,18730	**16292,18464**	16293,21054
	Gap.	0,00%	0,00%	0,00%	0,00%
vowel:	Best	148967,24099	148967,20312	**148967,18750**	148967,25000
	Média	149012,72062	148975,94000	**148967,19281**	150655,89781
	Gap.	0,00%	0,00%	0,00%	0,00%
cmc:	Best	5532,18472	**5532,18261**	**5532,18261**	5532,18505
	Média	5532,18472	5532,18363	**5532,18332**	5810,20918
	Gap.	0,00%	0,00%	0,00%	0,00%
cancer:	Best	2964,38697	**2964,38598**	**2964,38598**	2964,38696
	Média	2964,38697	2964,38623	**2964,38621**	3028,50237
	Gap.	0,00%	0,00%	0,00%	0,00%
glass:	Best	210,00104	210,42863	**210,00103**	210,42890
	Média	210,58446	211,30036	**210,03859**	247,15263
	Gap.	0,00%	0,20%	0,00%	0,20%
crudeoil:	Best	277,21072	**277,21069**	294,69293	277,21075
	Média	277,27518	**277,21072**	299,43907	279,23954
	Gap.	0,00%	0,00%	6,31%	0,00%
thyroid:	Best	2167,12919	**2155,61767**	**2155,61767**	2155,61792
	Média	2123,78661	**2155,61767**	**2155,61767**	2156,00289
	Gap.	0,53%	0,00%	0,00%	0,00%
artset1:	Best	1831,11532	**1831,11499**	**1831,11499**	1831,11523
	Média	1831,11533	**1831,11511**	1831,11507	1977,31633
	Gap.	0,00%	0,00%	0,00%	0,00%
artset2:	Best	545,03768	**545,03753**	**545,03753**	545,03765
	Média	545,03768	**545,03758**	**545,03753**	554,03852
	Gap.	0,00%	0,00%	0,00%	0,00%

Tabela 5.2: Continuação dos resultados obtidos com o tempo limite de 60 segundos.

Instâncias:	Critério	C-GRASP-Clu	GPU-PSO	GPU-DE	GPU-SS
artset3:	Best	1742,17983	**1742,17895**	**1742,17895**	1742,17944
	Média	1707,33624	1742,17911	**1742,17905**	1773,67709
	Gap.	0,00%	0,00%	0,00%	0,00%
artset4:	Best	5198,72783	**5198,72558**	**5198,72558**	5198,72656
	Média	5214,72724	5198,72606	**5198,72577**	5243,57537
	Gap.	0,00%	0,00%	0,00%	0,00%
abalone:	Best	124,23001	124,24639	**124,22472**	125,53461
	Média	124,26792	130,62587	**124,22928**	134,86320
	Gap.	0,00%	0,02%	0,00%	1,05%
ionosphere:	Best	**793,71228**	**793,71228**	793,71240	793,71319
	Média	793,71251	793,71482	**793,71247**	1236,00684
	Gap.	0,00%	0,00%	0,00%	0,00%
wdbc:	Best	149473,85499	**149473,84375**	149473,90625	149473,93750
	Média	**149473,85499**	149473,87000	178341,99937	149620,45906
	Gap.	0,00%	0,00%	0,00%	0,00%
yeast:	Best	233,67349	**233,64415**	305,12670	249,88955
	Média	**234,63230**	239,57667	307,78897	450,94266
	Gap.	0,01%	0,00%	30,59%	6,95%
D31:	Best	**2882,02182**	2882,02246	2884,08911	3019,02758
	Média	3065,06908	3008,07833	**2887,40044**	4371,11014
	Gap.	0,00%	0,00%	0,07%	4,75%
a1:	Best	5,37263E+06	**5,37262E+06**	5,37264E+06	5,37263E+06
	Média	5,66870E+06	5,48697E+06	**5,37266E+06**	6,37480E+06
	Gap.	0,00%	0,00%	0,00%	0,00%
s1:	Best	1,69056E+08	**1,69055E+08**	2,84003E+08	1,69056E+08
	Média	1,77330E+08	**1,69056E+08**	2,84004E+08	2,31931E+08
	Gap.	0,00%	0,00%	67,99%	0,00%
unbalance:	Best	**2,95803E+07**	**2,95803E+07**	**2,95803E+07**	**2,95803E+07**
	Média	3,21676E+07	**2,95803E+07**	**2,95803E+07**	3,97271E+07
	Gap.	0,00%	0,00%	0,00%	0,00%
Média total:		0,03%	**0,02%**	1,76%	1,48%
GAP Máximo:		0,53%	**0,20%**	67,99%	6,95%

Tabela 5.3: Resultados obtidos com o tempo limite de 30 segundos.

Instâncias:	Critério	C-GRASP-Clu	GPU-PSO	GPU-DE	GPU-SS
iris:	Best	96,65548	**96,65547**	**96,65547**	**96,65547**
	Média	96,65548	**96,65547**	**96,65547**	120,33233
	Gap.	0,00%	0,00%	0,00%	0,00%
wine:	Best	16292,18464	**16292,18359**	**16292,18359**	16292,18945
	Média	**16292,18464**	16292,19818	16292,18478	16293,26609
	Gap.	0,00%	0,00%	0,00%	0,00%
vowel:	Best	148967,24096	148967,20312	**148967,18750**	148984,51562
	Média	149023,72720	148977,93843	**148967,19906**	152400,59437
	Gap.	0,00%	0,00%	0,00%	0,01%
cmc:	Best	5532,18472	**5532,18261**	5532,18310	5532,18554
	Média	5532,18472	5532,18352	**5532,18347**	5986,35945
	Gap.	0,00%	0,00%	0,00%	0,00%
cancer:	Best	2964,38697	**2964,38598**	**2964,38598**	2964,38720
	Média	2964,38697	2964,38626	**2964,38622**	3159,34363
	Gap.	0,00%	0,00%	0,00%	0,00%
glass:	Best	**210,00104**	210,42863	210,00105	210,44578
	Média	210,81424	211,20853	**210,03751**	239,53434
	Gap.	0,00%	**0,20%**	0,00%	0,21%
crudeoil:	Best	277,21072	**277,21069**	294,69293	277,21075
	Média	277,27501	**277,21072**	300,24464	279,22578
	Gap.	0,00%	0,00%	6,31%	0,00%
thyroid:	Best	2167,12919	**2155,61767**	**2155,61767**	2155,61767
	Média	2167,12919	**2155,61767**	**2155,61767**	2163,41161
	Gap.	**0,53%**	0,00%	0,00%	0,00%
artset1:	Best	1831,11532	1831,11511	**1831,11499**	1831,11523
	Média	1831,11533	1831,11513	**1831,11509**	2094,30431
	Gap.	0,00%	0,00%	0,00%	0,00%
artset2:	Best	545,03768	**545,03753**	**545,03753**	545,03765
	Média	545,03768	545,03759	**545,03754**	553,58550
	Gap.	0,00%	0,00%	0,00%	0,00%

Tabela 5.4: Continuação dos resultados obtidos com o tempo limite de 30 segundos.

artset3:	Best	1742,17983	1742,17907	**1742,17895**	1742,17944
	Média	1742,17983	1742,17914	**1742,17907**	1790,97409
	Gap.	0,00%	0,00%	0,00%	0,00%
artset4:	Best	5198,72783	**5198,72558**	**5198,72558**	5198,72656
	Média	5228,67346	5198,72603	**5198,72590**	5480,45958
	Gap.	0,00%	0,00%	0,00%	0,00%
abalone:	Best	**124,22461**	124,24639	124,22739	125,53921
	Média	124,29271	130,17718	**124,23983**	138,01019
	Gap.	0,00%	0,02%	0,00%	1,06%
ionosphere:	Best	**793,71228**	793,71234	793,71240	793,71337
	Média	**793,71228**	793,71333	793,71250	1221,79507
	Gap.	0,00%	0,00%	0,00%	0,00%
wdbc:	Best	149473,85499	**149473,84375**	149473,87500	149473,93750
	Média	**149473,85499**	149473,87062	184904,15406	156982,28625
	Gap.	0,00%	0,00%	0,00%	0,00%
yeast:	Best	**233,64570**	233,64588	305,12670	259,85763
	Média	**234,84678**	239,73554	306,90341	365,20688
	Gap.	0,00%	0,00%	30,59%	**11,22%**
D31:	Best	2882,33697	**2882,02197**	2892.78710	3034,06469
	Média	3018,38260	3029,22541	**2915,04469**	4214,82870
	Gap.	0,01%	0,00%	0,37%	5,28%
a1:	Best	**5,37263E+06**	**5,37263E+06**	5,37275E+06	**5,37263E+06**
	Média	5,74500E+06	5,55246E+06	**5,37341E+06**	6,25218E+06
	Gap.	0,00%	0,00%	0,00%	0,00%
s1:	Best	1,69056E+08	**1,69055E+08**	2,84010E+08	1,69056E+08
	Média	1,81010E+08	**1,69056E+08**	2,84065E+08	2,25228E+08
	Gap.	0,00%	0,00%	**68,00%**	0,00%
unbalance:	Best	**2,95803E+07**	**2,95803E+07**	**2,95803E+07**	**2,95803E+07**
	Média	3,25303E+07	**2,95803E+07**	**2,95803E+07**	3,86849E+07
	Gap.	0,00%	0,00%	0,00%	0,00%
Média total		0,03%	**0,01%**	5,26%	0,89%
GAP Máximo		0,53%	**0,20%**	68,00%	11,22%

Tabela 5.5: Resultados obtidos com o tempo limite de 10 segundos.

Instâncias:	Critério	C-GRASP-Clu	GPU-PSO	GPU-DE	GPU-SS
iris:	*Best*	96.65548	**96,65547**	**96,65547**	**96,65547**
	Média	96,65548	**96,65547**	**96,65547**	124,06470
	Gap.	0,00%	0,00%	0,00%	0,00%
wine:	*Best*	16292,18464	16292,18457	**16292,18359**	16292,19238
	Média	**16292,18464**	16292,18767	16292,18521	16398,26705
	Gap.	0,00%	0,00%	0,00%	0,00%
vowel:	*Best*	148967,24102	148967,20312	**148967,18750**	148967,31250
	Média	149073,26461	148984,42656	**148967,20500**	158828,33906
	Gap.	0,00%	0,00%	0,00%	0,00%
cmc:	*Best*	5532,18472	**5532,18261**	5532,18310	5532,18505
	Média	5532,18472	5532,18377	**5532,18372**	6384,68904
	Gap.	0,00%	0,00%	0,00%	0,00%
cancer:	*Best*	2964,38697	2964,38623	**2964,38598**	2964,38696
	Média	2964,38697	2964.38632	**2964,38622**	3188.95074
	Gap.	0,00%	0,00%	0,00%	0,00%
glass:	*Best*	210,00104	210,00103	**210,00108**	210,43222
	Média	211,52460	210,93681	**210,07757**	273,41456
	Gap.	0,00%	0,00%	0,00%	0,21%
crudeoil:	*Best*	277.21072	**277.21069**	294,69293	277,21075
	Média	277,27674	**277,21073**	301,91995	283,41081
	Gap.	0,00%	0,00%	6,31%	0,00%
thyroid:	*Best*	2167,12919	**2155,61767**	**2155,61767**	2155,61792
	Média	2168,62042	**2155,61767**	**2155,61767**	2176,84702
	Gap.	0,53%	0,00%	0,00%	0,00%
artset1:	*Best*	1831,11532	1831,11511	**1831,11499**	1831,11523
	Média	1831,11534	1831,11513	**1831,11510**	2212,04721
	Gap.	0,00%	0,00%	0,00%	0,00%
artset2:	*Best*	545,03768	**545,03753**	**545,03753**	545,03765
	Média	545,03768	545,03759	**545,03754**	667,16653
	Gap.	0,00%	0,00%	0,00%	0,00%

Tabela 5.6: Continuação dos resultados obtidos com o tempo limite de 10 segundos.

artset3:	*Best*	1742,17983	1742,17907	**1742,17895**	1742,17956
	Média	1742,17983	1742,17916	**1742,17911**	1949,89409
	Gap.	0,00%	0,00%	0,00%	0,00%
artset4:	*Best*	5198,72783	**5198,72558**	**5198,72558**	5198,72705
	Média	5220,83526	5198,72623	**5198,72604**	5730,39611
	Gap.	0,00%	0,00%	0,00%	0,00%
abalone:	*Best*	**124,22529**	124,28488	124,25612	140,01962
	Média	124,343714	130,90984	**124,31908**	155,71074
	Gap.	0,00%	0,05%	0,02%	12,71%
ionosphere:	*Best*	**793,71228**	793,71240	793,71246	793,71331
	Média	**793,71344**	793,71508	793,71254	884,60472
	Gap.	0,00%	0,00%	0,00%	0,00%
wdbc:	*Best*	**149473,85499**	149473,85937	149473,95312	149474,04687
	Média	**149473,85499**	149473,87812	200789,28437	157198,54812
	Gap.	0,00%	0,00%	0,00%	0,00%
yeast:	*Best*	**233,86790**	236,63305	305,12674	297,96902
	Média	**235,25767**	243,86391	307,79679	326,07281
	Gap.	0,00%	**1,18%**	30,47%	27,41%
D31:	*Best*	2978,05598	**2884,13574**	3047,59301	3697,12695
	Média	3206,30744	**3071,83749**	3143,54287	4873,65580
	Gap.	**3,26%**	0,00%	5,67%	**28,19%**
a1:	*Best*	5,37279E+06	**5,37262E+06**	5,38316E+06	5,70889E+06
	Média	5,75591E+06	5,47556E+06	**5,40894E+06**	6,60947E+06
	Gap.	0,00%	0,00%	0,20%	6,26%
s1:	*Best*	**1,69056E+08**	**1,69056E+08**	2,84333E+08	**1,69056E+08**
	Média	1,86547E+08	**1,69056E+08**	2,85552E+08	2,23652E+08
	Gap.	0,00%	0,00%	**68,19%**	0,00%
unbalance:	*Best*	**2,95803E+07**	**2,95803E+07**	**2,95803E+07**	**2,95803E+07**
	Média	3,43498E+07	**2,95803E+07**	**2,95803E+07**	4,00604E+07
	Gap.	0,00%	0,00%	0,00%	0,00%
Média total		0,19%	**0,06%**	5,54%	3,74%
GAP Máximo		3,26%	**1,18%**	68,19%	28,19%

Tabela 5.7: Resultados obtidos com o tempo limite de 1 segundo.

Instâncias:	Critério	C-GRASP-Clu	GPU-PSO	GPU-DE	GPU-SS
iris:	*Best*	96,65548	**96,65547**	**96,65547**	96,65550
	Média	96,65548	**96,65547**	**96,65547**	122,67504
	Gap.	0,00%	0,00%	0,00%	0,00%
wine:	*Best*	**16292,18464**	16292,18554	16292,18554	16296,49511
	Média	16292,76936	16292,21679	**16292,18853**	16526,78925
	Gap.	0,00%	0,00%	0,00%	0,03%
vowel:	*Best*	148967,24111	**148967,21875**	148967,23437	149054,84375
	Média	150079,26525	148989,35968	**148969,14281**	154052,18468
	Gap.	0,00%	0,00%	0,00%	0,06%
cmc:	*Best*	5532,19039	5532,20166	**5532,18457**	5565,521973
	Média	5532,23048	5532,27338	**5532,20962**	6227,53694
	Gap.	0,00%	0,00%	0,00%	0,60%
cancer:	*Best*	2964,38697	**2964,38623**	**2964,38623**	2964,40429
	Média	2964,38697	**2964,38654**	2964,38643	3185,07504
	Gap.	0,00%	0,00%	0,00%	0,00%
glass:	*Best*	**210,00178**	210,42869	210,45043	248,90499
	Média	213,25004	211,48114	**210,55025**	270,37343
	Gap.	0,00%	0,20%	0,21%	18,53%
crudeoil:	*Best*	277,21072	**277,21069**	294,69293	277,21340
	Média	277,26964	**277,21074**	300,83604	284,29367
	Gap.	0,00%	0,00%	6,31%	0,00%
thyroid:	*Best*	2167,12919	**2155,61767**	**2155,61767**	2155,61865
	Média	2179,19292	**2155,61767**	**2155,61767**	2203,89523
	Gap.	0,53%	0,00%	0,00%	0,00%
artset1:	*Best*	1831,11531	**1831,11511**	**1831,11511**	1831,28247
	Média	1831,11533	1831,11519	**1831,11512**	2261,70314
	Gap.	0,00%	0,00%	0,00%	0,01%
artset2:	*Best*	545,03768	545,03753	545,03753	545,03772
	Média	545,03768	545,03759	545,03758	676,33307
	Gap.	0,00%	0,00%	0,00%	0,00%

Tabela 5.8: Continuação dos resultados obtidos com o tempo limite de 1 segundo.

artset3:	*Best*	1742,17983	**1742,17907**	**1742,17907**	1742,17944
	Média	1742,179836	**1742,17921**	**1742,17921**	1790,12040
	Gap.	0,00%	0,00%	0,00%	0,00%
artset4:	*Best*	5198,72783	**5198,72607**	**5198,72607**	5198,97705
	Média	5214,77628	**5198,72661**	5198,72678	5410,89592
	Gap.	0,00%	0,00%	0,00%	0,00%
abalone:	*Best*	**124,83028**	139,71267	129,24919	194,53974
	Média	**128,41288**	164,46001	133,86502	245,80730
	Gap.	0,00%	11,92%	3,54%	55,84%
ionosphere:	*Best*	793,77068	793,86401	**793,72113**	801,97985
	Média	793,98150	794,03633	**793,75793**	825,84193
	Gap.	0,01%	0,02%	0,00%	1,04%
wdbc:	*Best*	**149473,85792**	149474,03125	149921,10937	149502,14062
	Média	**149473,87015**	149474,18062	227260,34562	152112,75906
	Gap.	0,00%	0,00%	0,30%	0,02%
yeast:	*Best*	**240,02970**	293,95266	310,46255	375,90560
	Média	**244,49200**	307,92191	322,36799	380,51034
	Gap.	0,00%	**22,47%**	29,34%	**56,61%**
D31:	*Best*	**3571,81856**	4182,79931	5154,99804	5328,12939
	Média	**3860,70716**	4377,98669	5707,61848	6106,18001
	Gap.	0,00%	17,11%	44,32%	49,17%
a1:	*Best*	6,37007E+06	6,14073E+06	6,90209E+06	**6,12254E+06**
	Média	7,14020E+06	**6,78070E+06**	7,25574E+06	7,20274E+06
	Gap.	4,04%	0,30%	12,73%	0,00%
s1:	*Best*	1,93960E+08	2,00679E+08	3,18485E+08	**1,72393E+08**
	Média	2,33727E+08	**2,23609E+08**	3,35558E+08	2,37226E+08
	Gap.	**12,51%**	16,41%	**84,74%**	0,00%
unbalance:	*Best*	2,98931E+07	2,97606E+07	**2,95978E+07**	3,25388E+07
	Média	3,65395E+07	3,12525E+07	**2,98263E+07**	4,28768E+07
	Gap.	1,00%	0,55%	0,00%	9,94%
Média total		**0.,90%**	3,45%	9,08%	9,59%
GAP Máximo		12,51%	**22,47%**	84,74%	56,61%

Na Tabela 5.9 mostra os *Gaps* percentuais. Foi utilizado as 20 melhores soluções *(Best)* de cada algoritmo. Consideramos todos os instantes de tempos 1s, 10s,30s e 60s segundos.

Tabela 5.9: Valor do *Gap* das meta-heurísticas com os tempos limites:

Tempo:	C-GRASP-Clu	GPU-PSO	GPU-DE	GPU-SS
1s	**0,90%**	3,45%	9,08%	9,59%
10s	0,19%	**0,06%**	5,54%	3,74%
30s	0,03%	**0,01%**	5,36%	0,89%
60s	0,03%	**0,02%**	1,76%	1,48%

Conforme a análise comparativa entre as soluções propostas o GPU-PSO atingiu resultados de melhor qualidade em todas as instâncias e em quase todos os critérios considerados. Destaca-se a robustez da meta-heurística proposta GPU-PSO, uma vez que o *Gap* do melhor resultado obtido é quase zero para todas as instâncias em contraste com as demais meta-heurísticas. A meta-heurística GPU-SS gerou os piores resultados para alguns dos critérios, com destaque para as instâncias *glass*, *yeast*, *abalone* e *D31*. Apesar disso, a meta-heurística conseguiu valores razoáveis nas demais instâncias.

A meta-heurística GPU-PSO conseguiu as melhores soluções *(Best)* em 16 das 20 instâncias no tempo de 60s segundos, já no tempo de 30s segundos 13 das 20 instâncias, contabilizando os empates com as outras meta-heurísticas e heurística. Desta forma, o GPU-PSO obteve o melhor desempenho em quase todas as instâncias, além de produzir soluções de alta qualidade no melhor caso (Best) com quase todos os *Gaps* igual a zero, com exceção das instâncias *glass* e *abalone*. Contudo o GPU-DE também obteve valores significativos, contabilizando 13 instâncias com o melhor caso *Best* em 60s segundos, por sua vez, no tempo limite de 30s segundos obteve 10 das 20 instâncias. Com relação ao valor das qualidades de soluções (Best) o GPU-DE, contabilizou 13 das 20 instâncias nos tempos tanto de 30s e 60s segundos.

A meta-heurística GPU-SS não conseguiu os melhores resultados para nenhum critério e empatou em 2 instâncias. Finalmente, o GPU-PSO conseguiu os melhores resultados comparados ao C-GRASP-Clu em quase todas as instâncias. O C-GRASP-Clu, nas instâncias *Ionosphere* e *D31*, teve o seu desempenho superior em qualidade e robustez, evidenciando a contribuição dos mecanismos incorporados na abordagem proposta em Queiroga et al. (2018).

Os resultados mostram que das quatro configurações de tempo, o GPU-PSO obteve vantagens significativas nos tempos de 10s, 30s e 60s sobre as demais. Os resultados mostram que o C-GRASP-Clu superou no tempo de 1s. Os resultados desta seção sugerem que uma abordagem utilizando GPU para clusterização de dados é bastante promissora em termos de qualidade e robustez das soluções.

5.1 *Speedup* da função objetivo

Nesta seção é apresentada uma análise do *speedup* obtida com a paralelização da função objetivo em GPU. A Equação (2.8) a qual foi apresentada anteriormente, detalha o cálculo do *speedup*.

Foi analisado o *speedup* da função objetivo em paralelo (Algoritmo 4.1). Para este experimento, foram realizadas 1.280 avaliações independentes do nosso procedimento de função objetivo escrita com a linguagem de programação C++ baseado em GPU e sua versão serial correspondente em cada instância de *benchmark*. Cada execução independente corresponde a uma solução gerada aleatoriamente. A Tabela 5.10 apresenta os resultados deste experimento. O speedup variou de 53x a 248x, com uma média de 175x.

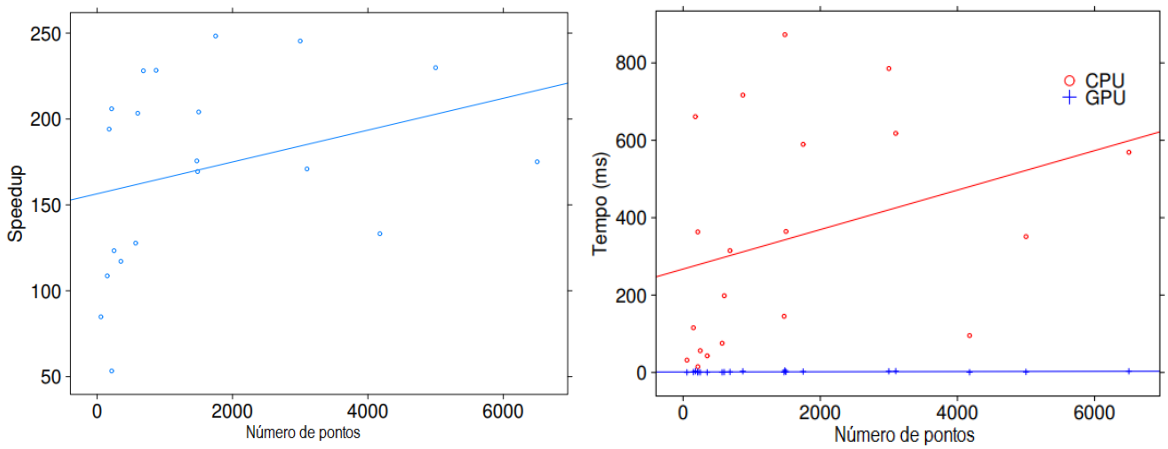
Tabela 5.10: *Speedup* da função objetivo com tempos em milissegundos.

Instâncias:	Tempo: CPU (ms)	Tempo: GPU (ms)	*Speedup*
iris	31,974	0,376928	84,82787
wine	115,622	1,06394	108,67342
cmc	661,116	3,40579	194,11531
cancer	363,291	1,76387	205,96245
crudeoil	14,9075	0,279648	53,30808
thyroid	56,3548	0,456864	123,35136
artset1	43,1293	0,368224	117,12788
artset2	75,6439	0,592128	127,74923
artset3	198,353	0,97536	203,36388
artset4	314,976	1,38099	228,07985
abalone	716,847	3,1393	228,34612
vowel	145,258	0,827168	175,60882
wdbc	873,017	5,15382	169,39221
D31	364,453	1,7856	204,10674
S1	589,588	2,37434	248,31658
unbalance	785,565	3,20093	245,41773
ionosphere	618,008	3,61546	170,93481
glass	95,4511	0,716512	133,21633
a1	351,216	1,52781	229,88198
yeast	569,091	3,24989	175,11084
Média *Speedup*:			**175,35983**

As Figuras 5.2(a) e 5.2(b) mostram a aceleração e o tempo de execução em função do número de pontos no conjunto de dados. Os resultados demonstram a escalabilidade da implementação e seu potencial para resolver com eficiência instâncias maiores. Na Figura 5.2(a) o eixo X é representado o número de pontos, por sua vez, no eixo Y temos o *speedup* da função objetivo, na medida em que o número de pontos aumenta o *speedup* também aumenta, isso é demonstrado pela linha reta em azul referente a aproximação linear. Por sua vez, na Figura 5.2(b) no eixo X temos o número de pontos e, no eixo Y o tempo total em milissegundos, os círculos em vermelha representa CPU e as cruzes representa em azul representa os pontos em GPU, dessa forma, na medida em que o número de pontos aumenta o tempo em CPU aumenta também podendo ser visto na linha vermelha referente a aproximação liner, por sua vez, a medina em que o número de pontos em GPU aumenta, a linha em azul permanece numa constante

indicando a escalabilidade do algoritmo da função objetivo em GPU.

Figura 5.1: *Speedup* da função objetivo (a) e (b).



(a) Número de pontos vs. *speedup*. Fonte: Autor.

(b) Número de pontos vs. tempo total. Fonte: Autor.

5.2 Testes estatísticos

Para aprimorar a análise de desempenho, foram realizados alguns testes estatísticos. A hipótese nula pressupõe que os algoritmos têm o mesmo desempenho médio e que as diferenças observadas são aleatórias. Desta forma, foram conduzidos testes utilizando uma análise dos valores *Best* de cada meta-heurística no tempo de 60 segundos. A partir do teste de *Friedman* visto na Tabela 5.11 ele e uma análise da classificação média do valor mínimo e valor máximo de cada meta-heurística sendo o valor mínimo do *ranking* indica a melhor meta-heurística.

O teste de *Friedman* é um procedimento não paramétrico que visa detectar se em um conjunto de amostras pelo menos duas delas representam populações com médias diferentes (Kaplan & Meier, 1958). Esta estatística é baseada nas classificações definidas pela qualidade das amostras, que em nosso caso corresponde à qualidade das soluções obtidas a partir das melhores soluções *Best* de cada meta-heurísticas em cada uma das instância do problema. O procedimento do *Friedman* foi utilizado para com o ajuste do p-valor $\alpha = 0,05$.

A classificação média de cada algoritmo é feita em classificações ascendentes da média obtida pela aplicação do procedimento de *Friedman* visto na Tabela 5.11.

Tabela 5.11: Classificação média dos algoritmos.

Algoritmos	*Ranking*
GPU-PSO	1,525
GPU-DE	2,175
C-GRASP-Clu	2,95
GPU-SS	3,35

Foi realizada uma comparação pareada para identificar as diferenças entre os algoritmos a partir do teste de *Friedman*. O valor menor que 0,05 indica que a hipótese nula pode ser rejeitada, e que dessa forma existe uma diferença significativa entre os algoritmos comparados. Os resultados obtidos em comparações para $\alpha = 0,05$ e p-valor ajustados são descritos na Tabela 5.12. Os resultados mostram que o GPU-PSO comparado com C-GRASP-Clu possui o valor $0,000482$ mostrando que existe diferença significava entre os algoritmos.

Tabela 5.12: p-valor para limite de $\alpha = 0.05$.

Algoritmos	p-valor
C-GRASP-Clu vs. GPU-SS	0,327187
GPU-PSO vs. GPU-DE	0,111347
C-GRASP-Clu vs. GPU-DE	0,057649
GPU-DE vs. GPU-SS	0,004
C-GRASP-Clu vs. GPU-PSO	0,000482
GPU-PSO vs. GPU-SS	0,000008

Capítulo 6

CONSIDERAÇÕES FINAIS

Clusterização de dados é uma classe de problemas fundamentais e bastante estudada pela comunidade de Inteligência Artificial. A clusterização possui inúmeras aplicações em muitas disciplinas, incluindo: bioinformática, visão computacional, mineração de dados, análise de expressão genética, mineração de texto, agrupamento de páginas na *Web*, entre outras áreas.

Existem inúmeros algoritmos para resolução de problemas de clusterização, e o problema pode ser atacando considerando algumas medidas de similaridade como a distância Euclidiana, distância de Manhattan, distância de Chebyshev, distância de Mahalanobis, coeficiente de Dunn, índice de Silhouette entre outras medidas ou variações do problema. A solução de um problema de clusterização consiste em um agrupamento que é utilizado para descobrir grupos naturais em conjuntos de dados. Deste modo, a clusterização visa identificar estruturas abstratas sem qualquer conhecimento prévio das características dos dados.

Neste trabalho, foram propostos estudos de técnicas para o problema de clusterização de dados em GPU baseadas em meta-heurísticas. Com o problema de clusterização particional com mínima distância *intra-clusters*, foi tratado através de uma nova abordagem onde consistiu a adaptação das meta-heurísticas que são PSO-GPU, DE-GPU e SS-GPU contidas no *framework* `libCudaOptimize` (Nashed et al., 2012).

Deste modo, extensivos experimentos computacionais em instâncias de dados conhecidas mostraram que os métodos heurísticos propostos foram capazes bons resultados. Os *gap* dos melhores resultados mostraram-se em *0%* em quase todos os casos considerando o melhor resultado da literatura comparado com C-GRASP-Clu visto em Queiroga et al. (2018). Desta forma, os resultados referente ao problema de clusterização de dados que foram comparados com as meta-heurísticas em GPU e, mostram resultados superiores com relação a heurística *C-GRASP-Clu*.

6.1 Contribuições

Conforme visto no Capítulo 3, vale ressaltar que existem poucos trabalhos relacionados à meta-heurísticas em GPU para clusterização de dados. As principais contribuições do trabalho são descritas a seguir:

1. **Paralelização em GPU da função objetivo** a paralelização pode ser utilizada em outras meta-heurísticas utilizando a arquitetura de GPU. Vale ressaltar que a implementação da paralelização da função objetivo, alcançamos um *speedup* **175x** em relação a versão sequencial da função objetivo.

2. **Integração da função objetivo** nas meta-heurísticas GPU-PSO, GPU-DE e GPU-SS. Até onde os autores conhecem este é o primeiro trabalho que faz uma comparação de diversas meta-heurísticas em GPU para o problema de clusterização.

3. **Análise comparativa** considerando as meta-heurísticas mais preeminentes da literatura GPU-PSO, GPU-DE e GPU-SS. Deste modo, a meta-heurística GPU-PSO foi capaz de ter os melhores resultados comparados ao melhor algoritmo conhecido na literatura que o C-GRASP-Clu visto em Queiroga et al. (2018).

6.2 Trabalhos futuros

Durante a elaboração deste trabalho surgiram várias possibilidades de pesquisa que podem agregar uma maior qualidade ao trabalho proposto nesta dissertação. Entretanto, devido à impossibilidade de tratá-los em prazo adequado, é descrito algumas possibilidades como trabalhos futuros. Algumas possibilidades de continuidade como trabalhos futuros desta pesquisa, são descritos a seguir:

- Estudar outras meta-heurísticas ou heurísticas em GPU como o *Biased Random-Key Genetic Algorithm* (BRKGA) (Goncalves & Resende, 2011) ou *Ant Colony Optimization* (ACO) (Dorigo, 1992) para problemas de clusterização de dados.

- Para o problemas de clusterização adaptamos meta-heurísticas em GPU utilizando a distância *intra-clusters*. Dessa forma, poderia ser aplicada outras variantes do problema. Essas variantes, podem ser implementadas na função objetivo utilizando outras métricas tais como a distância de Manhattan, distância de Chebyshev, distância de Mahalanobis, coeficiente de Dunn ou índice de Silhouette.

- As meta-heurísticas GPU-PSO, GPU-DE e GPU-SS que são utilizadas neste trabalho são genéricas, para isso, como trabalho futuro pode ser analisado uma meta-heurística com adaptações específicas para o problema de clusterização como é feito no C-GRASP-CLu.

Referências bibliográficas

Abdinnour-Helm, S. & Hadley, S. W. (2000), 'Tabu search based heuristics for multi-floor facility layout', *International Journal of Production Research* **38**(2), 365–383.

Ahmadi, A. F. Karray, M. K. (2012), 'Model order selection for multiple cooperative swarms clustering using stability analysis', *Information Sciences* **182**(1), 169–183. Nature-Inspired Collective Intelligence in Theory and Practice.

Ahuja, R. K., Orlin, J. B. & Tiwari, A. (2000), 'A greedy genetic algorithm for the quadratic assignment problem', *Computers & Operations Research* **27**(10), 917–934.

Alba, E. (2005), *Parallel metaheuristics: a new class of algorithms*, Vol. 47, John Wiley & Sons.

Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C. & Stewart, W. R. (1995), 'Designing and reporting on computational experiments with heuristic methods', *Journal of Heuristics* **1**(1), 9–32.

Berkhin, P. (2006), A survey of clustering data mining techniques, *in* 'Grouping multidimensional data', Springer, pp. 25–71.

Bubeck, S., Meilă, M. & von Luxburg, U. (2012), 'How the initialization affects the stability of the -means algorithm', *ESAIM: Probability and Statistics* **16**, 436–452.

Cassiano, K. M. & Pessanha, J. F. M. (2014), 'Análise espectral singular com clusterização baseada em densidade na modelagem de séries temporais.', *Anais do XLVI Simpósio Brasileiro de Pesquisa Operacional. Salvador-BA*.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W. & Skadron, K. (2008), 'A performance study of general-purpose applications on graphics processors using cuda', *Journal of parallel and distributed computing* **68**(10), 1370–1380.

Cheng, J., Grossman, M. & McKercher, T. (2014), *Professional Cuda C Programming*, John Wiley & Sons.

Crainic, T. G. & Toulouse, M. (2010), Parallel meta-heuristics, *in* 'Handbook of metaheuristics', Springer, pp. 497–541.

Dantzig, G. B. (1998), *Linear programming and extensions*, Princeton university press.

Dias, C. R. (2004), 'Algoritmos evolutivos para o problema de clusterização de grafos orientados: Desenvolvimento e análise experimental', *Dissertação de Mestrado. Rio de Janeiro*.

Dias, C. R. & Ochi, L. S. (2003), Efficient evolutionary algorithms for the clustering problem in directed graphs, *in* 'The 2003 Congress on Evolutionary Computation, 2003. CEC'03.', Vol. 2, IEEE, pp. 983–990.

Doni, M. V. (2004), 'Análise de cluster: métodos hierárquicos e de particionamento', *Trabalho de Conclusão de Curso. São Paulo: Universidade Presbiteriana Mackenzie, São Paulo*.

Dorigo, M. (1992), 'Optimization, learning and natural algorithms', *Ph. D. Thesis, Politecnico di Milano*.

Doval, D., Mancoridis, S. & Mitchell, B. S. (1999), Automatic clustering of software systems using a genetic algorithm, *in* 'STEP'99. Proceedings Ninth International Workshop Software Technology and Engineering Practice', IEEE, pp. 73–81.

Duhamel, C., Lacomme, P., Prins, C. & Prodhon, C. (2010), 'A grasp× els approach for the capacitated location-routing problem', *Computers & Operations Research* **37**(11), 1912–1923.

Essaid, M., Idoumghar, L., Lepagnot, J. & Brévilliers, M. (2019), 'Gpu parallelization strategies for metaheuristics: a survey', *International Journal of Parallel, Emergent and Distributed Systems* **34**(5), 497–522.

Everitt, B. S., Landau, S. & Leese, M. (2001), 'Cluster analysis arnold', *A member of the Hodder Headline Group, London* pp. 429–438.

Faria, H., Binato, S., Resende, M. G. & Falcão, D. M. (2005), 'Power transmission network design by greedy randomized adaptive path relinking', *IEEE Transactions on Power Systems* **20**(1), 43–49.

Farivar, R., Rebolledo, D., Chan, E. & Campbell, R. H. (2008), A parallel implementation of k-means clustering on gpus., *in* 'Pdpta', Vol. 13, pp. 212–312.

Feo, T. A. & Resende, M. G. (1995), 'Greedy randomized adaptive search procedures', *Journal of global optimization* **6**(2), 109–133.

Festa, P. & Resende, M. G. (2009), 'An annotated bibliography of grasp–part i: Algorithms', *International Transactions in Operational Research* **16**(1), 1–24.

Fränti, P. & Sieranoja, S. (2015), 'Clustering datasets', *URL:http://cs.uef.fi/sipu/datasets/* **4**, 2017.

Gendreau, M. & Potvin, J.-Y. (2005), Tabu search, *in* 'Search methodologies', Springer, pp. 165–186.

Gerrild, P. M. & Lantz, R. J. (1969), Chemical analysis of 75 crude oil samples from pliocene sand units, elk hills oil field, california, Technical report, US Geological.

Glover, F. (1977), 'Heuristics for integer programming using surrogate constraints', *Decision sciences* **8**(1), 156–166.

Glover, F. (1986), 'Future paths for integer programming and links to artificial intelligence', *Computers & operations research* **13**(5), 533–549.

Glover, F., Laguna, M. & Martí, R. (2003), Scatter search, *in* 'Advances in evolutionary computing', Springer, pp. 519–537.

Glover, F. W. & Kochenberger, G. A. (2006), *Handbook of metaheuristics*, Vol. 57, Springer Science & Business Media.

Goncalves, J. F. & Resende, M. G. (2011), 'Biased random-key genetic algorithms for combinatorial optimization', *Journal of Heuristics* **17**(5), 487–525.

Gonzalez, T. F. (1982), On the computational complexity of clustering and related problems, *in* 'System modeling and optimization', Springer, pp. 174–182.

Guerreiro, L. & Breve, F. A. (2015), Analysis of the influence of distance metrics on the semi-supervised algorithm of particle competition and cooperation, *in* 'Proceedings-XI Workshop de Visao Computacional WVC', pp. 77–82.

Hansen, P. & Jaumard, B. (1997), 'Cluster analysis and mathematical programming', *Mathematical programming* **79**(1-3), 191–215.

Harris, M. et al. (2007), 'Optimizing parallel reduction in cuda', *Nvidia developer technology* **2**(4), 1–39.

Hart, J. P. & Shogan, A. W. (1987), 'Semi-greedy heuristics: An empirical study', *Operations Research Letters* **6**(3), 107–114.

Johnson, S. C. (1967), 'Hierarchical clustering schemes', *Psychometrika* **32**(3), 241–254.

Kaplan, E. L. & Meier, P. (1958), 'Nonparametric estimation from incomplete observations', *Journal of the American statistical association* **53**(282), 457–481.

Karmarkar, N. (1984), A new polynomial-time algorithm for linear programming, *in* 'Proceedings of the sixteenth annual ACM symposium on Theory of computing', pp. 302–311.

Kazakovtsev, L., Rozhnov, I., Popov, E., Karaseva, M. & Stupina, A. (2019), Parallel implementation of the greedy heuristic clustering algorithms, *in* 'IOP Conference Series: Materials Science and Engineering', Vol. 537, IOP Publishing.

Kennedy, J. & Eberhart, R. (1995), Particle swarm optimization, *in* 'Proceedings of ICNN'95-International Conference on Neural Networks', Vol. 4, IEEE, pp. 1942–1948.

Kennedy, R. (1995), J. and eberhart, particle swarm optimization, *in* 'Proceedings of IEEE International Conference on Neural Networks IV, pages', Vol. 1000, p. 33.

Kirk, D. B. & Wen-Mei, W. H. (2016), *Programming massively parallel processors: a hands-on approach*, Morgan kaufmann.

Kogan, J., Nicholas, C., Teboulle, M. et al. (2006), *Grouping multidimensional data*, Springer.

Kohlhoff, K. J., Sosnick, M. H., Hsu, W. T., Pande, V. S. & Altman, R. B. (2011), 'Campaign: an open-source library of gpu-accelerated data clustering algorithms', *Bioinformatics* **27**(16), 2321–2322.

Krause, E. F. (1986), *Taxicab geometry: An adventure in non-Euclidean geometry*, Courier Corporation.

Krishnasamy, G., Kulkarni, A. J. & Paramesran, R. (2014), 'A hybrid approach for data clustering based on modified cohort intelligence and k-means', *Expert Systems with Applications* **41**(13), 6009 – 6016.

Krömer, P., Platoš, J. & Snášel, V. (2012), Genetic algorithm for clustering accelerated by the cuda platform, *in* '2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)', IEEE, pp. 1005–1010.

Laguna, M. & Velarde, J. L. G. (1991), 'A search heuristic for just-in-time scheduling in parallel machines', *Journal of Intelligent manufacturing* **2**(4), 253–260.

Lopes, H. S., Rodrigues, L. & Steiner, M. T. A. (2013), 'Meta-heurísticas em pesquisa operacional', *Editora Omnipax, Curitiba, PR*.

Lourenço, H. R., Paixão, J. P. & Portugal, R. (2001), 'Multiobjective metaheuristics for the bus driver scheduling problem', *Transportation science* **35**(3), 331–343.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M. & Stützle, T. (2016), 'The irace package: Iterated racing for automatic algorithm configuration', *Operations Research Perspectives* **3**, 43–58.

MacQueen, J. et al. (1967), Some methods for classification and analysis of multivariate observations, *in* 'Proceedings of the fifth Berkeley symposium on mathematical statistics and probability', Vol. 1, Oakland, CA, USA, pp. 281–297.

Man, K.-F., Tang, K. S. & Kwong, S. (2001), *Genetic algorithms: concepts and designs*, Springer Science & Business Media.

McClanahan, C. (2010), 'History and evolution of gpu architecture', *A Paper Survey*.

Nashed, Y. S., Ugolotti, R., Mesejo, P. & Cagnoni, S. (2012), libcudaoptimize: an open source library of gpu-based metaheuristics, *in* 'Proceedings of the 14th annual conference companion on Genetic and evolutionary computation', ACM, pp. 117–124.

Niknam, T. & Amiri, B. (2010), 'An efficient hybrid approach based on pso, aco and k-means for cluster analysis', *Applied soft computing* **10**(1), 183–197.

Ochi, L. S., Dias, C. R. & Soares, S. S. F. (2004), 'Minicurso de clusterização em mineração de dados', *Programa de Pós Graduação em Computação. Instituto de Computação-Universidade Federal Fluminense-Niterói* **1**, 46.

Pardalos, P. M. & Resende, M. G. (2002), 'Handbook of applied optimization'.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. et al. (2011), 'Scikit-learn: Machine learning in python', *the Journal of machine Learning research* **12**, 2825–2830.

Pu, G., Chong, Z., Qiu, Z., Lin, Z. & Jifeng, H. (2006), A hybrid heuristic algorithm for hw-sw partitioning within timed automata, *in* 'International Conference on Knowledge-Based and Intelligent Information and Engineering Systems', Springer, pp. 459–466.

Queiroga, E., Subramanian, A. & Lucídio dos Anjos, F. C. (2018), 'Continuous greedy randomized adaptive search procedure for data clustering', *Applied Soft Computing* **72**, 43–55.

Queiroga, E. V. et al. (2017), 'Abordagens meta-heurísticas para clusterização de dados e segmentação de imagens.', *Dissertação de Mestrado. Programa de Pós-Graduação em Informática – UFPB/CI*.

Rajkumar, M., Asokan, P., Anilkumar, N. & Page, T. (2011), 'A grasp algorithm for flexible job-shop scheduling problem with limited resource constraints', *International Journal of Production Research* **49**(8), 2409–2423.

Resende, M. G. & Ribeiro, C. C. (2003), 'A grasp with path-relinking for private virtual circuit routing', *Networks: An International Journal* **41**(2), 104–114.

Ribeiro, C. C., Uchoa, E. & Werneck, R. F. (2002), 'A hybrid grasp with perturbations for the steiner problem in graphs', *INFORMS Journal on Computing* **14**(3), 228–246.

Rios, E. (2016), Exploração de Estratégias de Busca Local em Ambientes CPU/GPU, PhD thesis, Tese de Doutorado, Programa de Pós-Graduação em Computação, Instituto de Computação.

Ruspini, E. H. (1969), 'A new approach to clustering', *Information and control* **15**(1), 22–32.

Sirotković, J., Dujmić, H. & Papić, V. (2012), K-means image segmentation on massively parallel gpu architecture, *in* '2012 Proceedings of the 35th International Convention MIPRO', pp. 489–494.

S.K. Pal, D. M. (1977), 'Fuzzy sets and decisionmaking approaches in vowel and speaker recognition', *IEEE Transactions on Systems, Man, and Cybernetics* **7**(8), 625–629.

Sörensen, K. & Glover, F. (2013), 'Metaheuristics', *Encyclopedia of operations research and management science* **62**, 960–970.

Sorensen, T. A. (1948), 'A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on danish commons', *Biol. Skar.* **5**, 1–34.

Steinhaus, H. (1956), 'Sur la division des corps matériels en parties', *Bull. Acad. Polon. Sci* **1**(804), 801.

Storn, R. (1995), 'Differrential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces', *Technical report, International Computer Science Institute*.

Storn, R. & Price, K. (1997), 'Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces', *Journal of global optimization* **11**(4), 341–359.

Talbi, E.-G. (2009), *Metaheuristics: from design to implementation*, Vol. 74, John Wiley & Sons.

Tan, Y. & Ding, K. (2016), 'A survey on gpu-based implementation of swarm intelligence algorithms', *IEEE Transactions on Cybernetics* **46**(9), 2028–2041.

Tsai, C.-Y. & Kao, I.-W. (2011), 'Particle swarm optimization with selective particle regeneration for data clustering', *Expert Systems with Applications* **38**(6), 6565 – 6576.

Van Luong, T., Melab, N. & Talbi, E.-G. (2011), 'Gpu computing for parallel local search metaheuristic algorithms', *IEEE Transactions on Computers* **62**(1), 173–185.

Wang, R., Zhou, Y., Qiao, S. & Huang, K. (2016), 'Flower pollination algorithm with bee pollinator for cluster analysis', *Information Processing Letters* **116**(1), 1–14.

Wang, X. & Sun, Q. (2016), The study of k-means based on hybrid sa-pso algorithm, *in* '2016 9th International Symposium on Computational Intelligence and Design (ISCID)', Vol. 2, pp. 211–214.

Xu, R., Xu, J. & Wunsch, D. C. (2010), Clustering with differential evolution particle swarm optimization, *in* 'IEEE Congress on Evolutionary Computation', pp. 1–8.

Zaïane, O. R., Foss, A., Lee, C.-H. & Wang, W. (2002), On data clustering analysis: Scalability, constraints, and validation, *in* 'Pacific-Asia Conference on Knowledge Discovery and Data Mining', Springer, pp. 28–39.

Zechner, M. & Granitzer, M. (2009), Accelerating k-means on the graphics processor via cuda, *in* '2009 First International Conference on Intensive Applications and Services', IEEE, pp. 7–15.