UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação

William Gabriel da Paz Rosendo

Algoritmos Tabu Search em GPU para Problemas de Maximização de Diversidade

Maceió - AL
2024

William Gabriel da Paz Rosendo

Algoritmos Tabu Search em GPU para Problemas de Maximização de Diversidade

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientador: Prof. Dr. Bruno Costa e Silva Nogueira

Maceió - AL
2024

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

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

R813a Rosendo, William Gabriel da Paz.
Algoritmos Tabu Search em GPU para problemas de maximização de diversidade / William Gabriel da Paz Rosendo. – 2024.
75 f. : il.

Orientador: Bruno Costa e Silva Nogueira.
Monografia (Trabalho de conclusão de curso em Engenharia de Computação) - Universidade Federal de Alagoas, Instituto de Computação. Maceió, 2024.
Texto em inglês.

Bibliografia: f. 71-74.
Apêndice: f. 75.

1. Otimização combinatória. 2. Processamento paralelo (Computadores). 3. Unidade de processamento gráfico. 4. Busca tabu. 5. Algoritmos híbridos. I. Título.

CDU: 004.421

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação do Instituto de Computação da Universidade Federal de Alagoas, aprovada pela banca examinadora que está abaixo.

Prof. Dr. Bruno Costa e Silva Nogueira - Orientador
Universidade Federal de Alagoas
Instituto de Computação

Prof. Dr. Rian Gabriel Santos Pinheiro - Examinador
Universidade Federal de Alagoas
Instituto de Computação

Prof. Dr. Ermeson Carneiro de Andrade - Examinador
Universidade Federal Rural de Pernambuco
Instituto de Computação

Maceió, 06 de Dezembro de 2024

Agradecimentos

Primeiramente, agradeço a Deus por me conceder forças, sabedoria e bênçãos, permitindo que eu continuasse essa jornada e superasse os desafios que surgiram ao longo do caminho.

Agradeço profundamente à minha mãe, Carmen Silva da Paz, e à minha namorada, Annanda Lopes Jacobs, pelo apoio constante, carinho e incentivo. Sem a força de ambas, não teria sido possível enfrentar as dificuldades dessa trajetória com a mesma coragem e determinação.

Sou grato à Universidade Federal de Alagoas (UFAL), ao corpo administrativo do Instituto de Computação e aos professores do curso de Engenharia de Computação, que me proporcionaram um ótimo ambiente de aprendizado e me ofereceram oportunidades de crescimento acadêmico e profissional.

Agradeço ao meu orientador, Prof. Dr. Bruno Costa e Silva Nogueira, pelo apoio e orientação tanto nas iniciações científicas quanto neste trabalho. Também sou muito grato aos professores Rian Gabriel Santos Pinheiro e Ermeson Carneiro de Andrade por aceitarem participar da banca examinadora deste trabalho, contribuindo com suas avaliações e sugestões.

E aos meus colegas do Instituto de Computação, que, com suas conversas, conselhos e apoio, tornaram essa jornada mais leve e agradável. Em especial, agradeço a Artur Cavalcante de Jesus, João Victor Ribeiro Ferro e Matheus Ferreira Gêda, pela amizade e colaboração ao longo dessa caminhada.

Se pude enxergar mais longe, foi porque me apoiei em ombros de gigantes

(Isaac Newton)

Resumo

Os problemas de otimização combinatória, como o Problema da Máxima Diversidade (MDP) e o Problema de Dispersão Maxima Média Ponderada (GMaxMeanDP), têm sido amplamente estudados devido à sua relevância em diversas aplicações práticas, como classificação de páginas web, mineração de comunidades, redes de confiança, formação de grupos, alocação de recursos e análise de redes sociais. Contudo, esses problemas enfrentam desafios significativos quando aplicados a instâncias de grande escala, frequentemente encontradas em cenários reais. Tanto o MDP quanto o GMaxMeanDP apresentam limitações em instâncias massivas, principalmente por utilizarem representações baseadas em matrizes, que se mostram ineficientes para dados esparsos. Além disso, a execução da busca local acaba demorando muito à medida que o tamanho do problema cresce, dificultando a obtenção de soluções em tempo viável. Essa combinação de ineficiência estrutural e aumento da complexidade computacional compromete a eficácia das estratégias de busca convencionais para problemas de grande escala. Para superar essas limitações, este estudo propõe algoritmos de busca tabu paralelizados, projetados especificamente para lidar com instâncias massivas de ambos os problemas. Os algoritmos exploram o poder do processamento paralelo em GPUs para acelerar significativamente o processo de busca local, permitindo a exploração eficiente de grandes espaços de soluções em menos tempo. Além disso, a implementação de estruturas de dados otimizadas para representar instâncias esparsas reduz a sobrecarga de memória e aumenta a escalabilidade do método. Os resultados experimentais demonstram que os algoritmos de busca tabu em GPU oferecem desempenho superior em relação aos algoritmos da literatura, tanto em termos de qualidade das soluções quanto de tempo de execução. Essa eficiência é ainda mais evidente em instâncias de grande escala, onde os algoritmos se destacam como uma alternativa eficaz e escalável para resolver os desafios impostos por problemas de otimização combinatória em cenários massivos.

Palavras-chaves: Otimização Combinatória, Processamento Paralelo, GPU, Busca Tabu, Algoritmos Híbridos.

Abstract

Combinatorial optimization problems, such as the Maximum Diversity Problem (MDP) and the Generalized Max-Mean Dispersion Problem (GMaxMeanDP), have been extensively studied due to their relevance in various practical applications, including web page classification, community mining, trust networks, group formation, resource allocation, and social network analysis. However, these problems face significant challenges when applied to large-scale instances commonly encountered in real-world scenarios. Both MDP and GMaxMeanDP exhibit limitations with massive instances, primarily due to their reliance on matrix-based representations, which are inefficient for sparse data. Additionally, the execution of local search becomes increasingly time-consuming as the problem size grows, hindering the ability to obtain solutions within a feasible timeframe. This combination of structural inefficiency and growth in computational complexity compromises the effectiveness of conventional search strategies for large-scale problems. To overcome these limitations, this study proposes parallelized tabu search algorithms specifically designed to handle massive instances of both problems. The algorithms leverage the power of parallel processing on GPUs to significantly accelerate the local search process, enabling the efficient exploration of large solution spaces in less time. Moreover, the implementation of optimized data structures for representing sparse instances reduces memory overhead and enhances the scalability of the method. Experimental results demonstrate that GPU-based tabu search algorithms outperform state-of-the-art methods in the literature, both in terms of solution quality and execution time. This efficiency is particularly evident in large-scale instances, where the algorithms stand out as an effective and scalable alternative to address the challenges posed by combinatorial optimization problems in massive scenarios.

key-words: Combinatorial Optimization, Parallel Processing, GPU, Tabu Search, Hybrid Algorithms

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A Pesquisa Operacional (PO) é uma área interdisciplinar que estuda, desenvolve e aplica métodos avançados para otimizar a tomada de decisões em uma vasta gama de setores. Sua aplicação expandiu-se com o avanço das tecnologias computacionais, abrangendo áreas como logística, saúde, finanças, entre outras. A PO utiliza técnicas de modelagem matemática e algoritmos computacionais para auxiliar a compreender e resolver problemas complexos, promovendo decisões mais eficientes e a criação de sistemas produtivos mais eficazes (SOBRAPO, 2024).

A PO se baseia em métodos matemáticos, como programação linear, simulação, teoria das filas, teoria dos jogos e otimização combinatória, para resolver problemas específicos de otimização, alocação de recursos e análise de redes (VOITTO, 2020). Essas técnicas são essenciais para o planejamento e a coordenação de operações, sendo amplamente aplicadas em setores como transporte, produção, finanças e administração pública. A PO permite transformar problemas complexos em modelos matemáticos de fácil análise, viabilizando decisões mais precisas e a redução de custos operacionais.

A aplicabilidade da PO varia conforme o tipo de problema a ser resolvido. Entre os métodos mais comuns estão a programação linear e inteira, utilizadas para otimizar a alocação de recursos escassos, além de técnicas de simulação, que auxiliam na previsão de cenários antes da implementação de soluções (UNISOMA, 2024). Essas abordagens contribuem significativamente para melhorar a eficiência operacional, reduzir riscos e aumentar a confiabilidade dos processos de decisão.

Um problema que a PO aborda é a maximização de diversidade (ZHOU; HAO, J. K., 2017), (LAI, X.; HAO, J.-K.; GLOVER, 2020). Esse problema envolve selecionar um subconjunto de elementos de um conjunto maior de forma que a distância entre os elementos selecionados

seja maximizada. O conceito de distância é definido de maneira personalizada, dependendo das necessidades da aplicação, o que resulta em diferentes modelos matemáticos para calcular a diversidade geral. Maximizar a diversidade tornou-se uma área de destaque na PO, especialmente nas últimas duas décadas, consolidando-se como um tema importante no campo da otimização combinatória (MARTÍ; MARTÍNEZ-GAVARA et al., 2022).

Os conceitos de dispersão e representatividade, fundamentais nesse tipo de problema, envolvem selecionar elementos que representem bem um conjunto maior, enquanto apresentam o maior grau possível de conexão entre si. Essa abordagem contribui para soluções mais robustas e eficientes em diversos cenários, como *design* de portfólio, seleção de amostras e problemas logísticos.

Com o crescente volume de dados gerados, resolver esses problemas torna-se cada vez mais complexo devido à alta demanda computacional. Para lidar com esses desafios, é essencial implementar algoritmos eficientes e explorar técnicas de programação paralela, possibilitando o processamento rápido de grandes volumes de dados.

1.1 Motivação e Justificativa

O Problema da Diversidade Máxima (MDP) se enquadra em uma ampla categoria de problemas de diversidade ou dispersão. Esse problema envolve a escolha de um subconjunto de elementos com tamanho fixo de forma que a diversidade dos elementos selecionados seja maximizada. Um exemplo desse problema seria na escolha de um subconjunto de cidades para abrir filiais de uma empresa, de forma que as filiais sejam o mais distante entre elas. Se tiver um conjunto de n cidades, o objetivo é escolher m cidades cuja soma das distâncias entre elas seja a maior possível, garantindo uma diversidade máxima.

O MDP possui alta complexidade computacional e uma ampla gama de aplicações práticas. Na redistribuição de terras, o MDP é utilizado para particionar áreas de forma a maximizar a diversidade entre elas (BORGWARDT; SCHMIEDL, 2014). Esse tipo de aplicação é útil em planejamento agrícola e urbano. Outro exemplo de aplicação ocorre na formação de grupos, o MDP auxilia na criação de equipes diversificadas, maximizando a diferença de habilidades, conhecimentos ou experiências entre os membros (KOVALEV; CHALAMON; PETANI, 2023). Essa abordagem é aplicada em ambientes corporativos e acadêmicos para incentivar inovação e produtividade.

Além disso, o MDP também pode ser aplicado em problemas de localização de instalações, onde ele busca posicionar elementos, como centros de distribuição ou sensores, de modo a maximizar a diversidade de suas localizações (PISINGER, 2006). Isso pode ser aplicado para minimizar riscos ou melhorar a cobertura espacial. Na análise de redes sociais, o MDP identifica subconjuntos de nós com alta diversidade, permitindo estudos sobre interações e comunidades ajudando a entender dinâmicas sociais e padrões de comportamento (ANDERSEN; CHELLAPILLA, 2009; LETSIOS et al., 2016).

O Problema de Dispersão Max-Média Ponderada (GMaxMeanDP) compartilha algumas semelhanças com o MDP, mas se diferencia principalmente na flexibilidade do tamanho do subconjunto e na forma como as soluções são avaliadas. Enquanto no MDP o objetivo é encontrar um subconjunto de tamanho fixo, no GMaxMeanDP o tamanho do subconjunto pode variar dinamicamente ao longo da busca pela solução ótima. Além disso, no GMaxMeanDP, cada elemento do subconjunto possui um peso associado. Assim, a solução ótima no GMaxMeanDP é obtida ao maximizar a média ponderada entre os elementos.

Um exemplo clássico do GMaxMeanDP também seria no planejamento da expansão de filiais de uma empresa em diferentes cidades. Neste caso, além de buscar cidades distantes entre si, como no MDP, também é importante considerar o peso de cada cidade, representando a importância da economia ou o tamanho da população. Diferentemente do MDP, o GMaxMeanDP não exige que o número de cidades seja fixo, mas sim que o número de cidades possa variar dinamicamente durante a busca para encontrar a melhor combinação que gere uma diversidade máxima

O GMaxMeanDP também apresenta aplicações relevantes. Na classificação de páginas na web, o GMaxMeanDP é usado para ranquear páginas de forma a maximizar a diversidade de resultados apresentados ao usuário (KERCHOVE; DOOREN, 2008). Isso é fundamental para melhorar a experiência de busca e evitar redundâncias.

Por outro lado, o GMaxMeanDP pode ser aplicado na mineração de comunidades, onde auxilia na detecção de grupos diversificados em redes sociais, permitindo a identificação de padrões e comunidades com diferentes interesses (YANG; CHEUNG; LIU, J., 2007). Na construção de redes de confiança, o GMaxMeanDP é aplicado para selecionar nós ou conexões que maximizem a dispersão de atributos, aumentando a robustez e a confiabilidade da rede (CARRASCO et al., 2015).

1.2 Objetivo Geral

O objetivo geral deste trabalho é desenvolver duas meta-heurísticas *Tabu Search* em GPU para resolver instâncias massivas dos problemas de Diversidade Máxima e Máxima Média Ponderada, buscando alcançar resultados superiores em comparação às abordagens existentes na literatura.

1.2.1 Objetivos Específicos

Para atingir o objetivo geral, este trabalho se propôs a:

- Entendimento do uso da Busca Tabu na otimização de problemas combinatórios e de algoritmos implementados em GPUs.
- Análise do desempenho da Busca Tabu com outros métodos de busca, tanto locais quanto globais.
- Caracterização e entendimento dos problemas de Diversidade Máxima e de Dispersão Máxima Média Ponderada.
- Desenvolvimento de meta-heurísticas Busca Tabu utilizando paralelismo para resolver os dois problemas.
- Avaliação da performance das heurísticas implementadas em GPUs.

1.3 Contribuições do Trabalho

Este trabalho propõe dois algoritmos de busca tabu baseados em GPU projetados para lidarem com instâncias massivas do MDP e do GMaxMeanDP, superando efetivamente os principais desafios enfrentados pelas heurísticas de ponta ao lidar com tais instâncias. As principais contribuições deste trabalho são descritas a seguir:

- Propor duas buscas tabu baseadas em GPU que aproveitam o poder das capacidades paralelas da GPU para acelerar a exploração das vizinhanças dos problemas. Até onde se sabe, este é o primeiro trabalho a estudar aplicações de GPU no MDP e no GMaxMeanDP.

- Os algoritmos em GPU propostos utilizam estruturas de dados mais eficientes em comparação com representações baseadas em matrizes, que se mostram menos eficazes para instâncias esparsas comumente encontradas em instâncias massivas.
- A busca tabu em GPU proposta para o MDP é integrada a um algoritmo memético de ponta (ZHOU; HAO, J.-K.; DUVAL, 2017), e o algoritmo resultante é testado considerando instâncias pequenas e médias, com até 5.000 vértices, bem como instâncias massivas com até 45.000 vértices.
- A busca tabu em GPU proposta para o GMaxMeanDP é integrada a um algoritmo evolutivo baseado em pertubação (LAI, X.; HAO, J.-K.; GLOVER, 2020), e o algoritmo resultante é testado considerando instâncias pequenas e médias, com até 5.000 vértices, bem como instâncias massivas com até 45.000 vértices.
- Os resultados experimentais demonstram a capacidade de aceleração das abordagens propostas e suas eficácias em instâncias massivas.

1.4 Estrutura do Trabalho

Este trabalho está organizado em sete capítulos, conforme descrito a seguir:

- **Capítulo 2**: Apresenta a fundamentação teórica essencial para compreender os conceitos e as técnicas de otimização combinatória aplicadas na Busca Tabu, além de abordar algoritmos de otimização.
- **Capítulo 3**: Realiza uma revisão bibliográfica dos métodos de otimização, com ênfase na Busca Tabu.
- **Capítulo 4**: Descreve o Problema de Diversidade Máxima, abordando suas características, aplicações e implementação proposta.
- **Capítulo 5**: Apresenta o Problema de Dispersão Máxima Média Ponderada, abordando suas características, aplicações e implementação proposta.
- **Capítulo 6**: Apresenta os resultados obtidos, incluindo uma análise detalhada do desempenho dos algoritmos.

- **Capítulo 7**: Conclui o trabalho, apresentando as conclusões gerais e sugestões para futuras pesquisas.

Capítulo 2

Fundamentação Teórica

Neste capítulo são abordados os conceitos necessários para o entendimento do problema alvo, que é resolução do problema de diversidade máxima e do problema de dispersão máxima média ponderada utilizando busca tabu paralelizada.

2.1 Otimização Combinatória

A otimização combinatória é uma área que busca encontrar a melhor solução em um conjunto finito de possibilidades, sendo fundamental para a resolução de inúmeros problemas práticos. Alguns desses problemas são classificados como NP-difíceis, ou seja, até o momento, não existe algoritmo de tempo polinomial conhecido capaz de resolvê-los. Exemplos clássicos incluem o problema do caixeiro viajante, empacotamento de itens, escalonamento de tarefas e satisfação booleana (BLUM; ROLI, 2003).

Um problema de otimização combinatória é formalmente definido por um par (S, f), onde $X = \{x_1, x_2, \ldots, x_n\}$ é um conjunto de variáveis, D_i é o domínio de valores possíveis para cada variável x_i, e S é o espaço de busca, composto por todas as combinações viáveis de variáveis que satisfazem as restrições impostas. A função objetivo $f : D_1 \times D_2 \times \cdots \times D_n \to \mathbb{R}^+$ é definida para ser minimizada ou maximizada. A solução ótima, $s^* \in S^*$, minimiza a função objetivo em problemas de minimização, ou seja, $f(s^*) \leq f(s)$ para todo $s \in S$. Por outro lado, em problemas de maximização, a solução ótima maximiza a função objetivo, ou seja, $f(s^*) \geq f(s)$ para todo $s \in S$. O conjunto S^* é composto por todas as soluções ótimas (PAPADIMITRIOU; STEIGLITZ, 1982).

Os problemas de otimização combinatória são desafiadores, pois o número de combinações possíveis cresce muito de acordo com o tamanho do problema. Métodos que examinam todas as possibilidades se tornam rapidamente inviáveis em instâncias de grande escala, demandando o desenvolvimento de abordagens mais eficientes (SCHRIJVER, 2005).

Os problemas de otimização combinatória são profundamente enraizados na prática. Eles surgem em tarefas cotidianas, como planejamento de rotas, alocação de recursos e organização de cronogramas. Esses desafios não são apenas modernos, mas possuem relevância histórica e até biológica, estando presentes desde sociedades primitivas e no comportamento de animais. O problema do caixeiro viajante, por exemplo, reflete necessidades básicas como otimizar trajetos para busca de recursos, transporte ou entrega de mercadorias (AARDAL et al., 2005). A Figura 2.1 apresenta um exemplo de situação do problema do caixeiro viajante. A ampla aplicação desses problemas, aliada à sua complexidade, faz da otimização combinatória um campo essencial na interseção entre matemática, ciência da computação e diversas outras áreas.

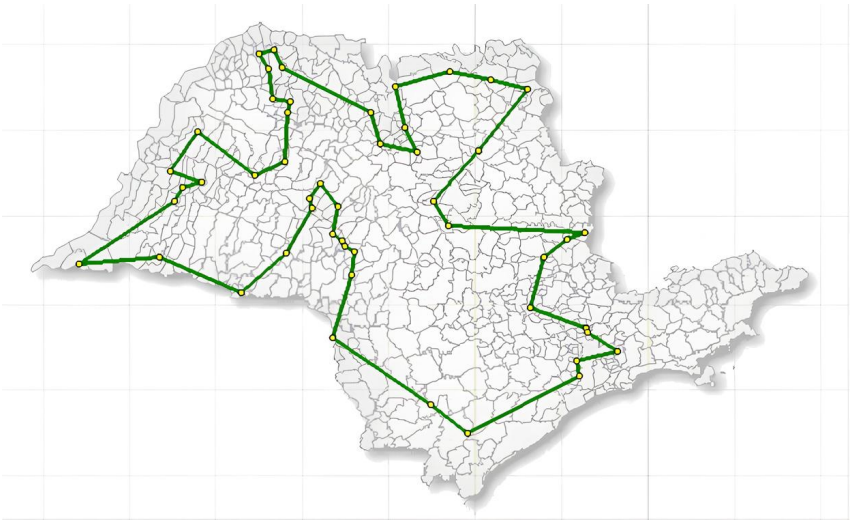


Figura 2.1: Rota conectando cidades de São Paulo, sem repetição. Fonte: (MIYAZAWA, 2024)

2.2 NP-Completude

A NP-completude é um conceito fundamental na teoria da computação, que classifica problemas de decisão com base em sua complexidade computacional. Os problemas dessa classe possuem a característica onde suas soluções ("sim") podem ser verificadas em tempo polinomial. Por outro lado, a classe P contém problemas que podem ser resolvidos diretamente em tempo polinomial por um algoritmo (SHAH, 2023; GAREY; JOHNSON, 1979).

A área de complexidade computacional é essencial dentro da ciência da computação teórica, pois busca classificar problemas de acordo com a dificuldade de sua resolução. Enquanto os problemas de P podem ser resolvidos rapidamente (em tempo polinomial), os problemas em NP podem ser mais difíceis de resolver, pois, embora suas soluções possam ser verificadas rapidamente, encontrar uma solução pode ser uma tarefa extremamente desafiadora. Um ponto importante é que todos os problemas em P estão também em NP, já que se podemos resolver um problema em tempo polinomial, também podemos verificar a solução rapidamente (SHAH, 2023).

Os problemas NP-completos são um subconjunto dentro da classe NP. Eles possuem uma característica intrigante porque, embora ainda não se tenha encontrado uma solução eficiente para esses problemas, ninguém provou que não é possível existir tal solução. Isso é conhecido como o Problema P vs NP, o maior desafio não resolvido na ciência da computação. Se for encontrado um algoritmo eficiente para resolver qualquer problema NP-completo, isso implicaria que todos os problemas em NP poderiam ser resolvidos de forma eficiente. Isso torna a busca por algoritmos eficientes para problemas NP-completos um dos maiores objetivos da computação teórica (CORMEN et al., 2002; GAREY; JOHNSON, 1979).

A redução de problemas permite transformar um problema em outro de forma que uma solução para o segundo problema também resolva o primeiro. Isso é crucial, pois permite classificar a dificuldade de diferentes problemas e entender a relação entre eles. Stephen Cook, em 1971, introduziu o conceito de redução em tempo polinomial e mostrou que o problema da satisfatibilidade (SAT) é NP-completo. Esse trabalho foi expandido por Richard Karp, em 1972, que identificou uma série de problemas combinatórios, como o Problema do Caixeiro Viajante, que são tão difíceis quanto o problema de satisfatibilidade, ou seja, são NP-completos (KARP, 1972; COOK, 1971).

Além disso, existem conceitos adicionais fundamentais para categorizar a complexidade dos problemas. O termo NP-difícil refere-se a problemas que, pelo menos, são tão difíceis quanto os problemas NP-completos. Contudo, um problema NP-difícil não precisa necessariamente pertencer à classe NP nem ser um problema de decisão. Um problema é classificado como NP-difícil se satisfizer a condição de que todo problema em NP pode ser reduzido a ele em tempo polinomial. Isso implica que, se uma solução eficiente fosse encontrada para um problema NP-difícil, tal solução permitiria resolver todos os problemas da classe NP de ma-

neira eficiente (SHAH, 2023; GAREY; JOHNSON, 1979). A Figura 2.2 mostra um diagrama das classes abordadas nesta seção.

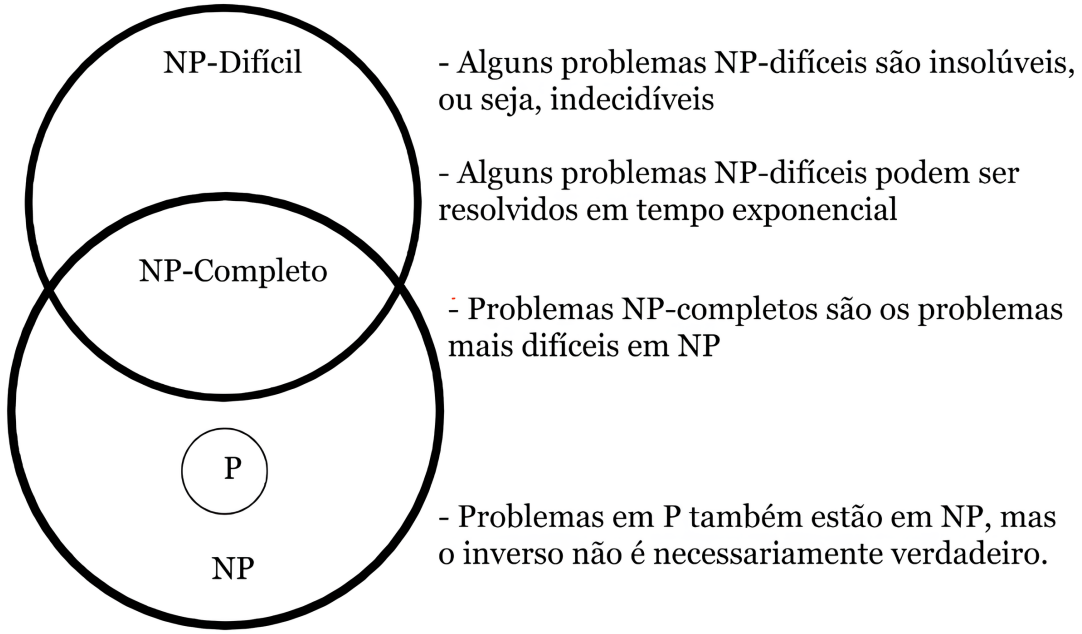


Figura 2.2: Diagrama das diferentes classes de complexidade. Fonte: (SHAH, 2023)

O estudo da NP-completude não se limita apenas à teoria, mas também tem implicações práticas significativas, especialmente em áreas como otimização e inteligência artificial. Muitos problemas reais podem ser formulados como problemas NP-completos, e, embora a busca por soluções exatas possa ser inviável, o desenvolvimento de algoritmos aproximativos e soluções heurísticas oferece alternativas para encontrar boas soluções de maneira mais eficiente. Além disso, em alguns casos, o uso de tecnologias como GPUs pode ser explorado para acelerar a resolução de problemas, embora, para problemas de grande escala, a dificuldade de resolver problemas NP-completos de forma eficiente ainda permaneça um desafio importante (GAREY; JOHNSON, 1979).

2.3 Algoritmos Exatos e Algoritmos Aproximados

Os algoritmos de otimização podem ser classificados em duas grandes categorias: exatos e aproximados, cada uma com características, vantagens e limitações distintas. Os algoritmos exatos são projetados para encontrar uma solução ótima de um problema, explorando o espaço de soluções de forma completa ou utilizando métodos matemáticos que garantem a precisão do

resultado. Eles são essenciais em aplicações que demandam alta confiabilidade, mas, tratando de problemas NP-difíceis, sua complexidade os tornam inadequados.

Por outro lado, os algoritmos aproximados são métodos que buscam soluções viáveis, frequentemente próximas do ótimo, dentro de um tempo computacional reduzido. Eles são especialmente úteis para problemas complexos ou instâncias massivas onde os métodos exatos se tornam impraticáveis. Embora não garantam encontrar a solução ótima, sua flexibilidade e eficiência os tornam uma escolha popular em diversas aplicações práticas.

2.4 Algoritmos Exatos

Algoritmos exatos são algoritmos rigorosos que garantem encontrar uma solução ótima de um problema de otimização ou provam que essa solução não existe. Geralmente, essas técnicas baseiam-se em métodos como enumeração, ramificações ou planos de corte, explorando sistematicamente a região viável do problema e eliminando soluções subótimas. A principal vantagem dos algoritmos exatos é a garantia de alcançar a melhor solução possível, além de fornecer uma métrica que indica a proximidade das outras soluções em relação à ideal (HAAS; NAMBIAR, 2024).

No entanto, em problemas NP-completos ou NP-difíceis, esses métodos apresentam limitações significativas, como a alta demanda por recursos computacionais, memória ou tempo de execução. Isso ocorre especialmente quando o tamanho do problema aumenta, tornando esses algoritmos impraticáveis para instâncias grandes (FULBER-GARCIA; AIBIN, 2023). Assim, em muitos casos, é necessário recorrer a métodos heurísticos, que trocam a garantia de otimalidade por maior eficiência computacional.

Os algoritmos exatos se destacam por sua precisão: garantem (com 100% de probabilidade) a solução ótima de um problema. Estratégias como força bruta são exemplos de métodos sempre exatos, mas que podem demandar um custo computacional elevado. Exemplos de algoritmos exatos amplamente usados incluem:

- Branch-and-Bound (B&B)
- Programação Dinâmica
- Métodos baseados em relaxação Lagrangiana

- Programação Linear e Inteira, com variações como branch-and-cut, branch-and-price e branch-and-cut-and-price.

Esses métodos têm sido amplamente aplicados a problemas combinatórios difíceis, embora sua viabilidade prática esteja limitada a instâncias pequenas ou de tamanho moderado (PUCHINGER; RAIDL, 2005).

2.5 Algoritmos Aproximados

Os algoritmos de aproximados podem ser classificados em duas categorias principais: aproximativos e heurísticas, cada uma com suas características, vantagens e limitações. Os algoritmos aproximativos são projetados para fornecer soluções viáveis que estão próximas do ótimo, garantindo uma margem de erro aceitável. Embora esses algoritmos não garantam a solução ótima, eles são eficazes em encontrar boas soluções em um tempo computacional reduzido.

Por outro lado, as heurísticas são abordagens que buscam encontrar soluções boas, mas sem qualquer garantia de proximidade com a solução ótima ou de qualidade mínima. Elas geralmente utilizam regras baseadas em experiências anteriores para encontrar soluções rápidas, mas não fornecem um critério claro de quão boas são as soluções encontradas.

2.5.1 Algoritmos Aproximativos

Um algoritmo aproximativo é um algoritmo que retorna um valor não necessariamente ótimo, mas um garantia de proximidade do ótimo. Ele é uma abordagem de resolução de problemas que retorna uma solução que, embora não seja necessariamente a solução ótima, o algoritmo garante que ela está próxima da ótima. Esse tipo de algoritmo busca balancear eficiência computacional e qualidade da solução, oferecendo uma resposta que, em muitos casos, é suficientemente boa para ser útil em aplicações práticas, mesmo que não atinja o valor ideal. A principal vantagem dos algoritmos aproximativos é a garantia de que a solução encontrada estará dentro de um intervalo controlado de proximidade da solução ótima.

Por exemplo, em um algoritmo de minimização (ou maximização), um algoritmo é considerado 2-aproximativo se, no pior caso, a solução retornada for no máximo duas vezes maior (ou menor) do que a solução ótima. Ou seja, a solução obtida é garantidamente dentro de uma margem de erro de 100% em relação ao valor ótimo.

2.5.2 Heurísticas

Algoritmos heurísticos são amplamente utilizados em aplicações práticas devido à sua flexibilidade e eficiência em fornecer soluções aceitáveis em tempos razoáveis, mesmo para problemas complexos. Embora não garantam soluções ótimas, são eficazes na resolução de problemas em grande escala ou intratáveis por métodos exatos (SHAH, 2023; HAAS; NAMBIAR, 2024).

De maneira geral, algoritmos heurísticos utilizam regras práticas, intuição ou experiência para encontrar soluções boas o suficiente, mesmo sem a garantia de serem as melhores possíveis. Eles são frequentemente mais rápidos que métodos exatos, mas podem apresentar limitações, como sensibilidade às condições iniciais ou tendência a ficarem presos em ótimos locais (FULBER-GARCIA; AIBIN, 2023).

Além disso, heurísticas apresentam características específicas:

- **Design baseado em problemas**: São adaptadas para resolver problemas específicos.
- **Sucesso não mensurável**: Não indicam quão próximo o resultado obtido está da solução ótima, diferentemente de métodos exatos.
- **Execução eficiente**: O tempo ou recursos computacionais necessários nunca devem exceder os de métodos exatos para o mesmo problema (FULBER-GARCIA; AIBIN, 2023).

Essas características tornam as heurísticas ferramentas valiosas em cenários onde métodos exatos são inviáveis, especialmente ao resolver problemas complexos.

2.5.3 Meta-Heurísticas

Meta-heurísticas, como resfriamento simulado (SA) e busca tabu (TS), são amplamente utilizadas para resolver problemas de otimização combinatória complexos. Essas técnicas são eficazes em encontrar soluções de alta qualidade para problemas onde métodos exatos são inviáveis devido à complexidade computacional (SHAH, 2023).

Assim como as heurísticas, metaheurísticas buscam soluções promissoras, mas apresentam um design genérico e independente de problemas, o que as torna aplicáveis a uma ampla gama de cenários. Isso contrasta com o design específico de problema característico das heurísticas (FULBER-GARCIA; AIBIN, 2023). As principais categorias de meta-heurísticas incluem:

- **Busca local**: Melhoram iterativamente uma única solução inicial até alcançar um ótimo local.

- **Construtivas**: Dividem o problema em subproblemas menores, resolvendo-os separadamente e combinando as soluções parciais.

- **Baseadas em populações**: Trabalham com um conjunto de soluções potenciais, aprimorando-as por meio de combinação e modificação, como exemplificado em algoritmos genéticos (FULBER-GARCIA; AIBIN, 2023).

Além disso, meta-heurísticas podem incluir algoritmos como busca local iterada, busca com vizinhança variável, otimização por enxame de partículas, algoritmos meméticos e de estimativa de distribuição. Recentemente, esforços têm sido direcionados para combinar metaheurísticas com métodos exatos, como descrito por Dumitrescu e Stützle, que exploraram abordagens híbridas de busca local e algoritmos exatos (DUMITRESCU; STUETZLE, 2003).

Tabu-Search

A busca tabu (TS) é uma meta-heurística amplamente utilizada para otimização combinatória, conforme discutido por Glover e Laguna (1997) (GLOVER; LAGUNA, 1997). A principal característica dessa abordagem é a utilização de uma estrutura de vizinhança N e uma função de avaliação f para realizar diversas iterações com o objetivo de aprimorar a solução atual.

Dada uma solução inicial s, pertencente ao espaço de soluções S, é possível realizar uma transição saindo dessa solução s e alcançando uma solução vizinha s'. Essa transição é conhecida como movimento. A vizinhança de uma solução pode ser definida por um conjunto de movimentos e consiste em um mapeamento que atribui a cada solução $s \in S$ um conjunto de soluções vizinhas $N(s)$. Já a função f define o número que indica por quanto tempo, ou quantas iterações, um movimento ou elemento permanece na lista tabu, de modo que esse movimento, ou qualquer movimento envolvendo os elementos associados, não possa ser realizado novamente, pois já foi visitado anteriormente.

Durante cada iteração, o algoritmo substitui a solução atual s por uma melhor solução vizinha elegível $s' \in N(s)$, registrando o movimento (ou os elementos associados) na lista tabu. Isso impede que o movimento inverso seja executado nas próximas tt iterações, onde tt é denominado tabu tenure, que é ajustado de acordo com a estratégia de gerenciamento da lista tabu definida na função f (LAI, X.; HAO, J.-K.; GLOVER, 2020).

No método de busca tabu, uma solução vizinha é considerada elegível se não for proibida pela lista tabu ou se for melhor que a melhor solução encontrada até aquele ponto na execução. O processo termina quando o critério de parada é alcançado. O modelo geral da busca tabu é

apresentado no Algoritmo 2, apresentado na seção 5.3, e os seus componentes são detalhados nas seções subsequentes (LAI, X.; HAO, J.-K.; GLOVER, 2020).

A busca tabu e outras heurísticas modernas apresentam limitações, especialmente em cenários de grande escala (BRIMBERG; MLADENOVIC; UROŠEVIĆ et al., 2009). Uma das principais limitações é a dependência de uma representação matricial para armazenar as distâncias, o que se torna ineficiente para instâncias esparsas, comuns em problemas de grande escala. Além disso, à medida que o tamanho da instância aumenta, a exploração da vizinhança se torna computacionalmente cara.

2.6 GPU's

Uma GPU (Unidade de Processamento Gráfico) é um circuito eletrônico capaz de efetuar cálculos matemáticos em alta velocidade e realizar tarefas que exigem processamento paralelo, como renderização de gráficos, aprendizado de máquina e edição de vídeo. Sua arquitetura permite que a mesma operação seja executada simultaneamente em vários dados, aumentando a eficiência em tarefas intensivas de computação. Inicialmente criada para controle gráfico, a GPU evoluiu para tarefas de uso geral, como processamento paralelo, a partir do final da década de 1990. A Nvidia, em 1999, foi pioneira ao lançar a GeForce 256, uma GPU de chip único, e em 2007 introduziu o CUDA, permitindo o uso da GPU para diversas aplicações de computação paralela.

As GPUs são amplamente utilizadas em áreas como jogos, visualização profissional, ML, blockchain e simulações científicas, podendo ser autônomas (placas gráficas dedicadas) ou integradas em sistemas móveis como as iGPUs. Elas também podem ser virtualizadas em ambientes de nuvem, proporcionando flexibilidade no uso de recursos computacionais. A principal diferença entre GPUs e CPUs está em suas funções: enquanto a CPU gerencia o sistema e executa tarefas gerais, a GPU é otimizada para tarefas que demandam alto poder de computação, como gráficos e aprendizado de máquina.

Nos anos 80 e 90, os microprocessadores de única unidade central (CPU) impulsionaram grandes aumentos de desempenho, mas a partir de 2003, as limitações de consumo de energia e dissipação de calor levaram ao desenvolvimento de chips multicore. Isso permitiu a execução simultânea de múltiplas sequências de instruções, favorecendo a programação paralela, que, a partir de então, tornou-se crucial para aproveitar o máximo desempenho dos novos

processadores (HWU; KIRK; HAJJ, 2022). A computação paralela é essencial para lidar com a crescente complexidade das aplicações, como simulações moleculares e grandes volumes de dados, e pode oferecer aumentos de desempenho de até 100 vezes ou mais em relação a execuções sequenciais.

A arquitetura das GPUs modernas, como as compatíveis com CUDA, é composta por multiprocessadores de streaming (SMs), cada um contendo múltiplos núcleos de processamento, que compartilham recursos de controle e memória. O sistema CUDA distribui as threads em blocos, atribuindo um bloco a cada SM, o que garante que as threads dentro de um mesmo bloco interajam simultaneamente, permitindo sincronização e acesso a memória compartilhada de baixa latência, o que é fundamental para a execução eficiente de programas paralelos (HWU; KIRK; HAJJ, 2022).

Capítulo 3

Revisão da Literatura

Esta Revisão da Literatura tem como objetivo identificar e analisar os trabalhos relacionados aos problemas propostos, bem como os métodos e algoritmos aplicáveis às suas resoluções. Serão discutidos os principais avanços na área, abordando tanto as soluções clássicas quanto as mais recentes, a fim de fornecer uma visão mais abrangente.

3.1 Trabalhos do Problema MDP

Brimberg et al. (2009) apresentaram uma heurística baseada em busca com vizinhança variável (VNS) para resolver o MDP. Diferentes variantes do método, incluindo VNS enviesado e a combinação de uma heurística construtiva seguida por VNS, foram analisadas. Experimentos computacionais extensivos em grandes grafos aleatórios e instâncias do problema de máxima diversidade mostraram que o VNS superou consistentemente outras heurísticas testadas, obtendo os melhores resultados em termos de qualidade de solução (BRIMBERG; MLADENOVIC; UROŠEVIĆ et al., 2009).

Baseado em uma aplicação na área florestal, Borgwardt et al. (2014) estudaram o MDP, propondo uma rotina de pré-processamento polinomial que preserva uma aproximação da solução ótima. O método identifica vértices de baixo interesse com base em um limiar especial, permitindo uma significativa redução do número de vértices no grafo de entrada. Estudos empíricos demonstraram que o método elimina mais de 90% dos vértices em grafos florestais, viabilizando a resolução ótima do problema nos grafos reduzidos, mesmo para instâncias grandes e sem a desigualdade triangular (BORGWARDT; SCHMIEDL, 2014).

Duarte e Martí (2007) apresentaram um algoritmo baseado em busca tabu para o MDP, incorporando estruturas de memória inovadoras para construção e melhoria das soluções. O método proposto foi comparado com abordagens sem memória e outras heurísticas da literatura, demonstrando superioridade em termos de qualidade das soluções e eficiência computacional. Experimentos extensivos com instâncias médias e grandes confirmaram que o algoritmo supera os melhores métodos existentes, obtendo resultados de alta qualidade em tempos reduzidos (DUARTE; MARTÍ, 2007).

Liu et al. (2022) propõem um algoritmo evolucionário baseado em busca tabu de duas fases (TPTS/EA) para o MDP, equivalente ao problema QUBO com restrição de cardinalidade. O algoritmo integra uma busca tabu com lista de candidatos dinâmicos na primeira fase e refinamento na segunda, além de operadores de recombinação por path-relinking. Testado em 140 instâncias públicas, o TPTS/EA melhora os melhores resultados conhecidos em duas instâncias e empata em 130, demonstrando alta qualidade de solução e eficiência computacional (LIU, X. et al., 2022).

Zhou et al. (2017) propõem um algoritmo memético baseado em oposição (OBMA) para resolver o MDP, integrando o aprendizado baseado em oposição (OBL) no framework de busca memética. O OBMA explora soluções candidatas e suas soluções opostas durante a inicialização e evolução, combinando com um procedimento de otimização local e uma estratégia de atualização de pool baseada em qualidade e distância. Testado em 80 instâncias populares do MDP, o algoritmo empata com as melhores soluções conhecidas na maioria das instâncias e encontra soluções melhores (novos limites inferiores) para 22 delas, evidenciando os benefícios do aprendizado baseado em oposição para resolver o MDP (ZHOU; HAO, J. K., 2017).

O MDP também é conhecido por outros nomes, como o problema do subgrafo mais pesado k (BRIMBERG; MLADENOVIC; UROŠEVIĆ et al., 2009), o problema do subgrafo denso ponderado k (BORGWARDT; SCHMIEDL, 2014) e o problema de máxima soma de diversidade (SPIERS; BUI; LOXTON, 2023). Ele pertence à classe NP-difícil, isso pode ser demonstrado por redução ao problema da clique máxima (FEIGE; SELTSER et al., 1997). Existem muitas aplicações práticas para o MDP, incluindo redistribuição de terras (BORGWARDT; SCHMIEDL, 2014), formação de grupos (KOVALEV; CHALAMON; PETANI, 2023), localização de instalações (PISINGER, 2006) e análise de redes (sociais) (ANDERSEN; CHELLAPILLA, 2009; LETSIOS et al., 2016). Várias dessas aplicações eventualmente exigem lidar com instâncias

massivas com dezenas de milhares de vértices. Heurísticas e algoritmos de aproximação são comumente usados para lidar com esse tipo de instância.

Muitas heurísticas foram propostas para o MDP, incluindo busca tabu (DUARTE; MARTÍ, 2007; KINCAID, 1992), busca tabu em GPU (NOGUEIRA et al., 2024), busca por vizinhança variável (BRIMBERG; MLADENOVIC; UROŠEVIĆ et al., 2009), recozimento simulado (KINCAID, 1992), busca dispersa (GALLEGO et al., 2009), GRASP (DUARTE; MARTÍ, 2007) e algoritmos meméticos (ZHOU; HAO, J.-K.; DUVAL, 2017; LIU, X. et al., 2022). Ao leitor é recomendada a consulta a (MARTÍ; GALLEGO et al., 2013) para uma comparação entre as melhores heurísticas publicadas até o ano de 2012. Comparações mais recentes foram apresentadas em (ZHOU; HAO, J.-K.; DUVAL, 2017; LIU, X. et al., 2022).

Comumente, as heurísticas mais bem-sucedidas para o problema têm a busca local como seu componente mais intensivo em termos computacionais (BRIMBERG; MLADENOVIC; UROŠEVIĆ et al., 2009; ZHOU; HAO, J.-K.; DUVAL, 2017; LIU, X. et al., 2022). Além disso, essas heurísticas dependem fortemente da vizinhança swap-(1,1) em seus procedimentos de busca local. O movimento swap-(1,1) consiste em trocar um vértice na solução por outro fora da solução. Claramente, essa vizinhança tem tamanho $O(m \times (n - m))$, resultando em demandas computacionais significativas, especialmente para instâncias maiores. Para resolver esse problema, heurísticas recentes empregam abordagens de filtragem visando reduzir o tamanho dessa vizinhança. No entanto, apesar da aplicação desses mecanismos de filtragem, essas heurísticas ainda enfrentam dificuldades ao lidar com instâncias massivas.

Para calcular rapidamente o ganho de um movimento swap-(1,1), essas heurísticas também adotam uma estrutura de dados baseada em matrizes para armazenar as distâncias. Isso permite verificar a distância $w(v_1, v_2)$ entre quaisquer dois vértices $v_1, v_2 \in V$ em tempo constante $O(1)$. É importante notar, no entanto, que para instâncias massivas, o armazenamento dessa matriz pode se tornar proibitivamente caro, especialmente se a matriz for esparsa. Como exemplo, considerando que as distâncias são armazenadas utilizando o tipo double (64 bytes), uma instância com 100.000 vértices necessitaria de 76 gigabytes de armazenamento. Na realidade, como a matriz de distâncias é simétrica, um algoritmo inteligente precisaria armazenar apenas metade da matriz, o que ainda é um tamanho considerável.

3.2 Trabalhos do Problema GMaxMeanDP

Estudos como o de Zhao et al. (ZHAO, J.; HIFI; LATREM, 2024) abordam o GMaxMeanDP utilizando uma abordagem híbrida de otimização, que combina otimização por enxame de partículas e busca de vizinhança variável distorcida, visando aprimorar tanto a qualidade das soluções quanto a eficiência computacional. Por outro lado, o trabalho de Lai et al. (LAI, X.; HAO, J.-K.; GLOVER, 2020) propõe dois algoritmos evolutivos híbridos, com busca tabu, para resolver o GMaxMeanDP, o qual, como mencionado, tem diversas aplicações práticas, como a classificação de páginas web, a mineração de comunidades e a construção de redes de confiança. Os algoritmos propostos integram estratégias inovadoras de busca, explorando eficazmente o espaço de soluções. Testados em 160 instâncias de benchmark, os resultados demonstram a eficácia e a utilidade dos algoritmos, que também podem ser aplicados para resolver outros problemas formulados como o GMaxMeanDP (LAI, X.; HAO, J.-K.; GLOVER, 2020).

Prokopyev et al. (2009) introduzem o problema de dispersão equitativa, que visa maximizar a equidade entre elementos em um sistema definido por distâncias inter-elementos, com base em novas medidas orientadas para a equidade no contexto de dispersão. O trabalho desenvolve formulações matemáticas e suas reformulações lineares inteiras mistas, além de discutir questões de complexidade computacional e interpretações gráfico-teóricas, apresentando resultados computacionais preliminares (PROKOPYEV; KONG; MARTINEZ-TORRES, 2009a).

O problema de dispersão Max-Mean, que recentemente recebeu atenção substancial na literatura (BRIMBERG; MLADENOVIC; TODOSIJEVIC et al., 2017; CARRASCO et al., 2015; DELLA CROCE; GARRAFFA; SALASSA, 2016; LAI, X. J.; HAO, J. K., 2016; MARTÍ; SANDOYA, 2013), é um caso particular do GMaxMeanDP, no qual $w_i = 1$ para $\forall i \in \{1, 2, \dots, n\}$. Assim, qualquer algoritmo desenvolvido para o GMaxMeanDP pode ser diretamente aplicado ao problema de dispersão Max-Mean, embora o contrário não seja válido (LAI, X.; HAO, J.-K.; GLOVER, 2020).

Além de sua relevância teórica como um problema NP-difícil (PROKOPYEV; KONG; MARTINEZ-TORRES, 2009a), o GMaxMeanDP possui diversas aplicações práticas, como classificação de páginas na web (KERCHOVE; DOOREN, 2008), mineração de comunidades em redes sociais assinadas (YANG; CHEUNG; LIU, J., 2007) e redes de confiança (CARRASCO et al., 2015), entre outras.

Existem também algoritmos especificamente para o MaxMeanDP, como os apresentados em (BRIMBERG; MLADENOVIC; TODOSIJEVIC et al., 2017), (DELLA CROCE; GARRAFFA; SALASSA, 2016) e (GARRAFFA; DELLACROCE; SALASSA, 2017), que podem ser adaptados para resolver o GMaxMeanDP. Além disso, vários estudos anteriores (BENLIC; HAO, J. K., 2015; GHOSH et al., 2019; ISMKHAN, 2017; MORRA; COCCIA; CERQUITELLI, 2018; SILVA; HRUSCHKA; GAMA, 2017; ZHAO, H.; XU; JIANG, 2015) demonstraram que a computação evolutiva é uma abordagem particularmente eficaz para resolver uma série de problemas difíceis de otimização combinatória. Dada a natureza NP-difícil do GMaxMeanDP, a computação evolutiva se configura como uma abordagem natural a ser investigada para enfrentar esse problema.

3.3 Considerações Finais

Este capítulo apresentou uma revisão abrangente da literatura sobre os problemas de otimização discutidos e os métodos heurísticos propostos para resolvê-los. A busca tabu, como uma meta-heurística amplamente estudada, tem se destacado pela sua capacidade de explorar o espaço de soluções além da ótica de otimalidade local, oferecendo uma abordagem robusta para diversos tipos de problemas, como o problema de diversidade máxima e dispersão máxima média ponderada.

Até onde se tem conhecimento, não há algoritmos baseados em GPU propostos para esses problemas. Além disso, os algoritmos existentes, que utilizam representações baseadas em matrizes, enfrentam dificuldades significativas ao lidar com instâncias de grande escala. Este trabalho aborda essa lacuna ao introduzir dois algoritmos eficientes e escaláveis, provando sua viabilidade e superioridade em instâncias massivas.

Capítulo 4

Problema de Diversidade Máxima

Neste capítulo é abordada a definição do problema de diversidade máxima, abordando suas características, aplicações e a implementação proposta.

4.1 Introdução

O MDP, um problema de otimização combinatória amplamente estudado devido às suas diversas aplicações, tem visto várias abordagens heurísticas propostas. No entanto, nenhuma dessas abordagens abordou os desafios impostos por instâncias massivas. Atualmente, as heurísticas de ponta não são bem adequadas para instâncias massivas devido a dois motivos principais. Primeiro, elas dependem de uma representação baseada em matrizes, que se mostra altamente ineficiente para instâncias esparsas comumente encontradas em cenários massivos. Segundo, à medida que o tamanho do problema aumenta, os operadores de busca local experimentam uma desaceleração. Este capítulo introduz um algoritmo de busca tabu baseado em GPU projetado para lidar com tais instâncias massivas. Para superar as limitações das heurísticas de ponta, a busca tabu em GPU utiliza estruturas de dados mais eficientes para instâncias esparsas e aproveita as capacidades paralelas da GPU para acelerar o processo de busca local.

4.2 Caracterização do problema

Considere um grafo não direcionado $G = (V, E)$, $|V| = n$, e uma função $w : V \times V \rightarrow \mathbb{R}$ que estima a diversidade (ou distância) entre quaisquer dois vértices de G. Assume-se que, se uma aresta (v_1, v_2) não estiver em E, então $w(v_1, v_2) = 0$. Dado um inteiro $m \leq n$, o MDP pede

para encontrar um subconjunto $S \subseteq V$ com $|S| = m$, tal que a soma das distâncias entre os vértices de S seja maximizada. A Figura 4.1. mostra um grafo de entrada, para o MDP, com suas possíveis soluções.

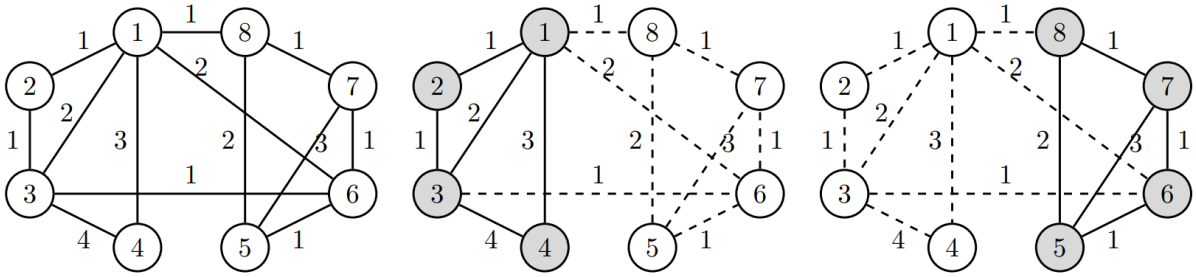


Figura 4.1: Exemplo de um grafo e suas possíveis soluções. Fonte: Autor

Este problema se enquadra em uma ampla categoria de problemas de diversidade ou dispersão. Esses problemas envolvem a escolha de um subconjunto de elementos de forma que a diversidade dos elementos selecionados seja maximizada. A formulação matemática do problema pode ser dada por:

$$\begin{aligned} \max \quad & f(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i = m, \\ & x \in \{0,1\}^n \end{aligned}$$

onde n é o tamanho do grafo, m tamanho do subgrafo, d_{ij} a aresta entre o vértice i e o vértice j e x_i se o vértice i pertence (1) ou não (0) à solução.

O restante deste trabalho está estruturado da seguinte forma: a Seção 4.3 fornece uma explicação detalhada das implementações atuais de busca tabu sequenciais e destaca suas limitações, particularmente quando aplicadas a instâncias em grande escala. A Seção 4.4 introduz a busca tabu paralela em GPU e demonstra como ela pode superar as limitações descritas na Seção 4.3. O desempenho da proposta do trabalho é avaliado no capítulo 6 e, finalmente, o capítulo 7 apresenta as considerações finais para o trabalho.

4.3 Busca Tabu Sequencial

O algoritmo GPU proposto é baseado no procedimento tradicional de busca tabu descrito no Algoritmo 1. A busca tabu é uma meta-heurística de busca local bem conhecida, que utiliza movimentos proibidos baseados em memória para escapar de ótimos locais (GLOVER; LAGUNA; MARTI, 2007).

Entrada: Solução inicial S, quantidade máxima de iterações $MaxIter$, função da distância w, grafo não direcionado $G = (V, E)$, tamanho do subconjunto m
Saída : Melhor solução encontrada S^*

```
1  BuscaTabu(S, MaxIter, w, G, m)
2      Inicializa a lista tabu
3      Inicializa o vetor gain
4      S* = S
5      iter = 1
6      while iter ≤ MaxIter do
7          Determina o melhor swap (v_i, v_j) elegível de acordo com o vetor gain e a lista
             tabu
8          S = S \ {v_i} ∪ {v_j}
9          Atualiza a lista tabu
10         Atualiza o vetor gain
11         iter = iter + 1
12         if Soma (S) > Soma (S*) then
13             S* = S
14     return S*
```

Algorithm 1: Pseudocódigo da busca tabu sequencial.

Começando com uma solução inicial S, o Algoritmo 1 melhora iterativamente a qualidade dessa solução usando os movimentos *swap-(1,1)*. Até que o critério de parada seja atendido, a cada iteração, o melhor movimento *swap-(1,1)* elegível é determinado e aplicado a S. Para evitar que a busca entre em ciclos entre soluções já visitadas, o Algoritmo 1 adota uma memória de curto prazo, conhecida como lista tabu, que proíbe alguns movimentos *swap-(1,1)*. O algoritmo retorna como saída a melhor solução S^* encontrada durante a busca.

Vizinhança. Dada uma solução candidata S, a estrutura de vizinhança *swap-(1,1)*, denotada por $N(S)$, consiste no conjunto de soluções que podem ser obtidas trocando um vértice da solução por outro que não está na solução, ou seja, $N(S) = \{S' : S' = S \setminus \{v_i\} \cup \{v_j\}, v_i \in S, v_j \in V \setminus S\}$. Para determinar rapidamente o ganho de movimento dessa troca (a mudança no valor objetivo ao passar de S para sua solução vizinha $S' \in N(S)$), métodos de última geração

utilizam o vetor de ganho (ZHOU; HAO, J.-K.; DUVAL, 2017; BRIMBERG; MLADENOVIC; UROŠEVIĆ et al., 2009; LIU, X. et al., 2022). Este vetor é inicializado como segue:

$$gain(v_i) = \sum_{v_j \in S} w(v_i, v_j) \tag{4.1}$$

onde gain(v_i, v_j) denota o ganho de movimento ao trocar dois vértices v_i e v_j e w(v_i, v_j) o peso da aresta entre os vértices. Esse ganho pode ser calculado usando a Eq. (4.2):

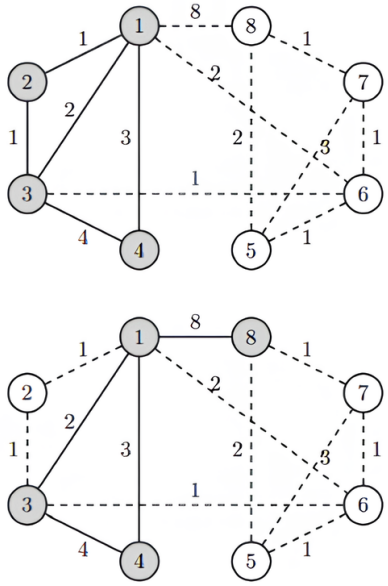
$$gain(v_i, v_j) = gain(v_j) - gain(v_i) - w(v_i, v_j) \tag{4.2}$$

Uma vez que uma troca entre v_i e v_j é realizada, o subconjunto de elementos de *gain* afetados pelo movimento precisa ser atualizado. Isso pode ser feito percorrendo as listas de adjacência de v_i e v_j, usando a Eq. (4.3):

$$gain(y) = \begin{cases} gain(y) + w(v_i, v_j), & \text{if } y = v_i \\ gain(y) - w(v_i, v_j), & \text{if } y = v_j \\ gain(y) - w(v_i, y) + w(v_j, y), & \text{if } y \neq v_i \text{ and } y \neq v_j \end{cases} \tag{4.3}$$

Com o vetor *gain*, o ganho de um movimento de troca pode ser avaliado em $O(1)$ usando a Eq. (4.2). Além disso, a complexidade para atualizar o vetor *gain* é $O(\Delta)$, onde Δ é o grau máximo do grafo G. Como mencionado anteriormente, o tamanho da vizinhança *swap-(1,1)* é $O(m \times (n-m))$. Assim, é necessário $O(m \times (n-m))$ para avaliar toda a vizinhança e selecionar o movimento com o melhor ganho (NOGUEIRA et al., 2024). A Figura 4.2 ilustra um exemplo de uma solução inicial e a solução vizinha obtida após a realização de um movimento swap entre os vértices 2 e 8.

Lista Tabu. No Algoritmo 1, inicialmente todos os movimentos *swap-(1,1)* são permitidos. Após uma troca entre v_i e v_j, ambos v_i e v_j são adicionados à lista tabu. Isso significa que v_i está proibido de entrar na solução pelas próximas T_i iterações, enquanto v_j está proibido de sair da solução pelas próximas T_j iterações. Os valores de T_i e T_j são definidos com base na técnica de gestão dinâmica da lista tabu apresentada em (ZHOU; HAO, J.-K.; DUVAL, 2017). Nesse método, a função f calcula o número de iterações que um vértice permanecerá na lista tabu (tabu tenure), ajustando esse valor dinamicamente conforme a iteração atual do algoritmo. Inicialmente, a função determina uma variável auxiliar $temp$, que corresponde ao resto da



Vértices	v1	v2	v3	v4	v5	v6	v7	v8
Lista Tabu	0	0	0	0	0	0	0	0
Gain	6	2	7	7	0	2	0	8
Soma Solução	11							

Vértices	v1	v2	v3	v4	v5	v6	v7	v8
Lista Tabu	0	15	0	0	0	0	0	10
Gain	13	2	6	7	2	2	1	8
Soma Solução	17							

Figura 4.2: Exemplo de um movimento swap do MDP. Fonte: Autor

divisão da iteração atual por uma constante T. Em seguida, verifica em qual intervalo $temp$ se encontra e ajusta o valor da tabu tenure de acordo com as regras estabelecidas. Por fim, a função retorna o valor calculado para a tabu tenure. O critério de aspiração também é usado, o que significa que, se um movimento leva a uma solução melhor do que qualquer outra já visitada, a lista tabu é ignorada, e a troca é aceita.

Para implementar a lista tabu, utiliza-se um vetor inteiro T_L de tamanho n, cujos elementos são inicializados com o valor 0. Após uma troca (v_i, v_j), define-se $T_L[v_i] = T_i + \text{iter}$ e $T_L[v_j] = T_j + \text{iter}$, onde iter é o contador de iteração atual. Para verificar se um vértice v está na lista tabu, basta verificar se $\text{iter} < T_L[v]$. Assim, a lista tabu pode ser atualizada e verificada em $O(1)$.

Limitações para Instâncias Massivas. O Algoritmo 1 e outras heurísticas de última geração compartilham limitações comuns (BRIMBERG; MLADENOVIC; UROŠEVIĆ et al., 2009; ZHOU; HAO, J.-K.; DUVAL, 2017; LIU, X. et al., 2022). Primeiramente, eles dependem de uma representação baseada em matriz para armazenar as distâncias das instâncias, o que se mostra altamente ineficiente para instâncias esparsas, comumente encontradas em cenários massivos. Em segundo lugar, à medida que o tamanho da instância aumenta, explorar a vizinhança *swap-(1,1)* torna-se muito custoso. Na próxima seção, é demonstrado como a abordagem proposta aborda e supera essas limitações (NOGUEIRA et al., 2024).

4.4 Busca Tabu Paralela Proposta

Esta seção introduz a abordagem de busca tabu em GPU proposta. Para superar as limitações dos métodos de última geração discutidos na seção anterior, esta abordagem em GPU utiliza estruturas de dados mais eficientes para instâncias esparsas e aproveita as capacidades paralelas da GPU para acelerar o processo de busca local.

4.4.1 Estruturas de Dados

Utilizar uma representação matricial para armazenar as distâncias das instâncias não só é ineficiente para instâncias esparsas, mas também apresenta desafios para a implementação em GPU. Isso ocorre principalmente devido à perda de localidade espacial no acesso à memória da GPU, o que diminui a eficiência da GPU. Portanto, nesta abordagem, uma instância de MDP $G = (V, E)$ é armazenada no formato de grafo bem conhecido Compressed Sparse Row (CSR), que é um esquema de codificação compacto e eficiente. O CSR consiste em dois arrays E' e V', com tamanhos $|E|$ e n, respectivamente. O E' contém a concatenação das listas de adjacência do gráfico, e o V' contém os índices indicando onde cada lista de adjacência começa. A distância de cada aresta é armazenada em um array adicional de tamanho $|E|$, denominado E'_w. A Figura 4.3 mostra um grafo G e sua representação CSR.

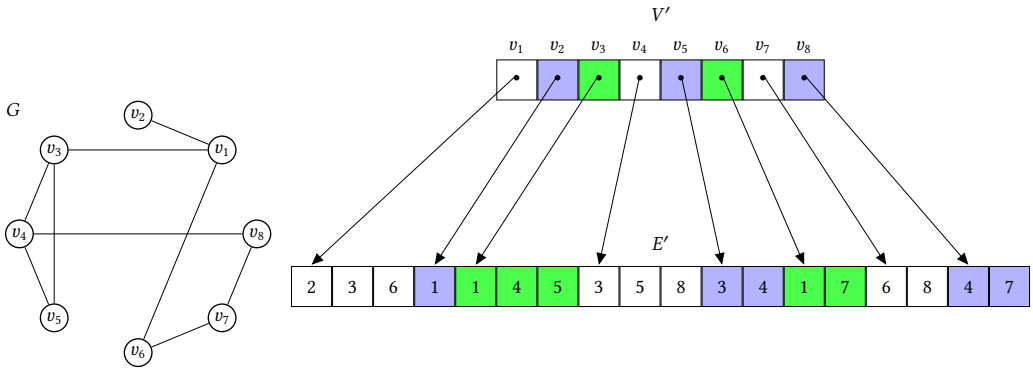


Figura 4.3: Exemplo de grafo G e sua representação CSR correspondente. Fonte: Autor

É importante observar que, embora a representação CSR melhore a eficiência do espaço e a localidade espacial, o acesso à distância $w(v_1, v_2)$ entre dois vértices v_1 e v_2 não é feito em tempo constante $O(1)$, mas sim em $O(\Delta)$. Consequentemente, a exploração da vizinhança swap-(1,1) pode enfrentar desafios significativos, pois a avaliação do ganho de uma troca também não seria mais $O(1)$ (veja a Eq. (4.2)). Para superar esse problema, é proposta a utilização de um novo vetor, denominado W_{swap}, de tamanho $m \times (n - m)$ (NOGUEIRA et al., 2024).

Seja uma solução candidata S representada por uma permutação τ, de modo que os primeiros m elementos de τ representem os vértices na solução, enquanto os $n - m$ elementos restantes representam os vértices fora da solução. Dada essa permutação, $W_{swap}(x)$, $x \in \{1, \ldots, m \times (n - m)\}$, é inicializado da seguinte forma:

$$W_{swap}(x) = w\left(\tau\left(\left\lfloor \frac{x-1}{m} \right\rfloor + 1\right), \tau\left(\text{mod}(x-1, n-m) + m + 1\right)\right), \tag{4.4}$$

onde $\text{mod}(a, b)$ é o módulo de a por b.

Observe que W_{swap} armazena a distância $w(v_i, v_j)$ para cada par (v_i, v_j), onde v_i está na solução S e v_j está fora da solução. Na Eq. (4.4), para um índice dado x, $\tau\left(\frac{x-1}{m} + 1\right)$ retorna um vértice na solução, enquanto $\tau(\text{mod}(x-1, n-m) + m + 1)$ retorna um vértice fora da solução. Usando o vetor W_{swap}, podemos calcular o ganho de movimento de uma troca usando a Eq. (4.5) em vez da Eq. (4.2).

$$gain'(x) = gain\left(\tau\left(\left\lfloor \frac{x-1}{m} \right\rfloor + 1\right)\right) - gain\left(\tau\left(\text{mod}(x-1, n-m) + m + 1\right)\right) - W_{swap}(x), \tag{4.5}$$

Com a Eq. (4.5), é possível avaliar todo a vizinhança swap-(1,1) em $O(m \times (n-m))$ enquanto usamos a representação CSR. O custo de espaço para a representação CSR juntamente com o array W_{swap} é $O(n \times (n-m) + |E| + n)$. Comparado com o custo da representação matricial $O(m \times m)$, esta abordagem é mais eficiente, especialmente em instâncias esparsas massivas, onde $m \ll n$, o que ocorre comumente para instâncias massivas (LETSIOS et al., 2016).

Após uma troca entre v_i e v_j, a permutação τ precisa ser atualizada, o que pode ser feito em $O(1)$ trocando as posições de v_i e v_j. Em seguida, os vetores gain e W_{swap} são simultaneamente atualizados, o que leva $O(\Delta)$ e pode ser realizado percorrendo as listas de adjacência dos vértices v_i e v_j. A atualização de W_{swap} é feita usando a Eq. (4.6).

$$W_{swap}(x) = \begin{cases} w(v_j, y), & \text{if } \tau(\lfloor \frac{x-1}{m} + 1) = v_j, \\ & \tau(\text{mod}(x-1, n-m) + m + 1)) = y, y \neq v_i \\ & \\ w(y, v_i), & \text{if } \tau(\lfloor \frac{x-1}{m} + 1) = y, y \neq v_j, \\ & \tau(\text{mod}(x-1, n-m) + m + 1)) = v_i \end{cases} \tag{4.6}$$

4.4.2 Busca Tabu em GPU

Como mencionado na Seção 4.3, uma das limitações dos métodos atuais baseados em busca local é que, à medida que a instância cresce, eles se tornam menos eficazes devido ao alto esforço computacional exigido. Agora, é descrita uma implementação massivamente paralela em GPU do Algoritmo 1 para superar essa limitação. Para fins de concisão, a descrição dessa implementação em GPU foca nas diferenças em relação ao Algoritmo 1.

A primeira diferença em relação ao Algoritmo 1 é que agora é necessário criar e manter as seguintes estruturas de dados na memória da GPU: (i) instância G, representada no formato CSR, (ii) solução atual S e a melhor solução S', representadas como permutações, (iii) vetor gain e a lista tabu, e (iv) vetor W_{swap}. Dada uma solução inicial S, a implementação em GPU começa criando e inicializando essas estruturas de dados na memória da GPU. Em seguida, entra no loop principal (linhas 6-13 do Algoritmo 1).

No loop principal, primeiro, é preciso encontrar o movimento de troca com o melhor ganho que não esteja na lista tabu (linha 7 do Algoritmo 1). Como discutido na Seção 4.3, essa operação é $O(m \times (n-m))$ e pode se tornar muito custosa à medida que o tamanho da instância cresce. Na implementação em GPU, essa operação é realizada lançando $m \times (n-m)$ threads na GPU, de modo que cada thread avalia em paralelo o ganho de uma troca candidata e, em seguida, verifica se o movimento correspondente é permitido pela lista tabu. Primeiro, cada thread $x \in [1, \ldots, m \times (n-m)]$ calcula seu respectivo ganho de troca (v_i, v_j) usando a Eq. (4.5). Em seguida, a thread verifica se os elementos envolvidos na troca estão na lista tabu, ou seja, se $TL[v_i] < iter$ e $TL[v_j] < iter$. Se um desses elementos for proibido pela lista tabu e o movimento não satisfizer o critério de aspiração, um custo de penalização é adicionado ao ganho do movimento. O custo de penalização é um valor constante negativo muito alto (NOGUEIRA et al., 2024).

Após calcular o ganho de cada troca potencial, o algoritmo emprega uma operação de redução paralela (WILT, 2013) nos ganhos das trocas para identificar o melhor movimento elegível. Nos casos em que todos os movimentos potenciais são restritos pela lista tabu, o movimento com o maior ganho entre os movimentos penalizados da lista tabu é selecionado. A troca (v_i, v_j) selecionada é então aplicada e as estruturas de dados na GPU são atualizadas (linhas 8-13 do Algoritmo 1). Em particular, o vetor W_{swap} é atualizado usando a Eq. (4.6). Todas as estruturas de dados são atualizadas ao mesmo tempo lançando $O(\Delta)$ threads, de

modo que cada thread $x \in [1, \ldots, \Delta]$ é responsável por atualizar um vizinho de v_i e um vizinho de v_j nos vetores W_{swap} e $gain$.

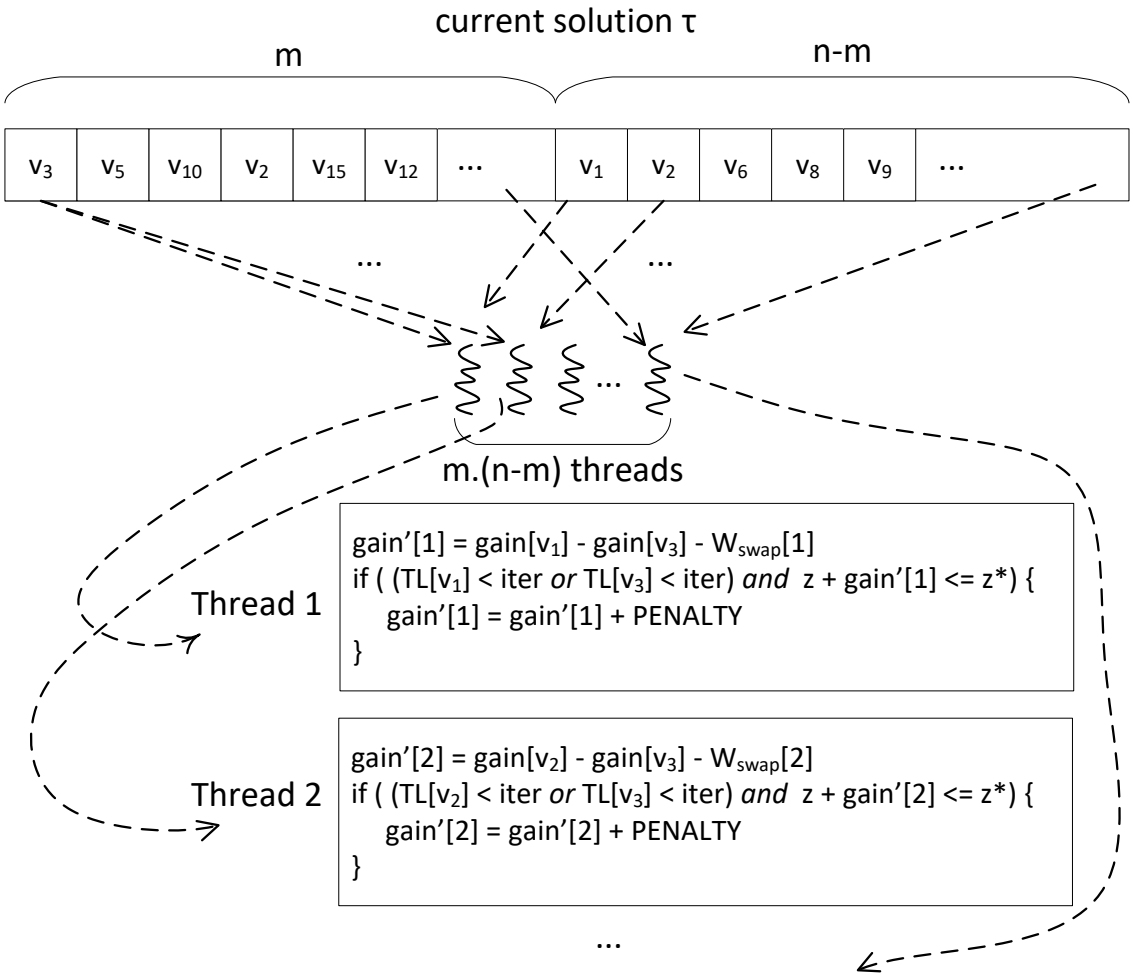


Figura 4.4: Avaliação paralela de movimentos de troca. Fonte: Autor

Em termos de complexidade, a avaliação paralela de todas as trocas potenciais juntamente com a subsequente operação de redução paralela nos valores dos ganhos das trocas é $O(n/p + \log(n) \times n/p)$, onde p é o número de threads disponíveis na GPU. Além disso, leva $O(\Delta/p)$ para atualizar as estruturas de dados propostas da GPU após uma troca ser aplicada (NOGUEIRA et al., 2024). A Figura 4.4 mostra a avaliação paralela de movimentos swaps na GPU.

4.4.3 Filtro 1

Na subseção anterior, foi proposta uma abordagem de busca tabu baseada em GPU que explora exaustivamente a vizinhança swap-(1,1). Como explorar todos os movimentos de troca pos-

síveis pode ser demorado, os métodos de última geração adotam o conceito de estratégia de troca restrita para reduzir o tamanho da vizinhança swap-(1,1). Nesta subseção, é apresentada uma implementação em GPU da estratégia de troca restrita apresentada em (ZHOU; HAO, J.-K.; DUVAL, 2017).

Dada uma solução S, seja minGain $= \min\{\text{gain}(v) : v \in S\}$, maxGain $= \max\{\text{gain}(v) : v \in V \setminus S\}$, e $w_{max} = \max\{w(v_i, v_j) : (v_i, v_j) \in E\}$. Os seguintes conjuntos são definidos: $U_S^c = \{v \in S : \text{gain}(v) \leq \text{minGain} + \frac{w_{max}}{2\rho}\}$ e $U_{V\setminus S}^c = \{v \in V \setminus S : \text{gain}(v) \leq \text{maxGain} - \frac{w_{max}}{2\rho}\}$, onde ρ é um parâmetro definido como 4 em (ZHOU; HAO, J.-K.; DUVAL, 2017). a vizinhança de troca restritoa proposta em (ZHOU; HAO, J.-K.; DUVAL, 2017) filtra as trocas elegíveis para aquelas em que um vértice está em U_S^c e o outro está em $U_{V\setminus S}^c$.

Para incluir essa estratégia de filtragem na implementação em GPU proposta na subseção anterior, os seguintes passos são tomados no início de cada iteração do loop principal da busca tabu. Primeiro, maxGain e minGain são calculados. Isso pode ser feito executando uma operação paralela de redução (max) nos valores de ganho dos vértices em S, e outra operação paralela de redução (min) nos valores de ganho dos vértices em $V \setminus S$. Essas duas operações juntas executam em $O(\log(n) \times n/p)$.

Em seguida, o algoritmo gera dois vetores para representar U_S^c e $U_{V\setminus S}^c$. Usando o valor calculado de minGain, a criação do vetor U_S^c é baseada na operação paralela de compactação (também conhecida como operação de seleção ou filtragem) (WILT, 2013). Essa operação gera o vetor U_S^c filtrando de S os vértices que não têm valores de ganho menores ou iguais a minGain+ $\frac{w_{max}}{2\rho}$. Da mesma forma, uma operação paralela de compactação é aplicada para gerar o vetor $U_{V\setminus S}^c$, considerando os vértices em $V \setminus S$ que não têm valores de ganho maiores que maxGain− $\frac{w_{max}}{2\rho}$.

A complexidade temporal das duas operações de compactação necessárias para criar U_S^c e $U_{V\setminus S}^c$ é $O(\log(n) \times n/p)$.

Após gerar U_S^c e $U_{V\setminus S}^c$, todas as trocas elegíveis na vizinhança restrita são avaliadas, e a melhor é selecionada. Com a vizinhança restrita, essa etapa leva $O(C/p + \log(C) \times C/p)$, onde $C = |U_S^c| \times |U_{V\setminus S}^c|$. Após a troca selecionada ser aplicada, atualizamos as estruturas de dados da GPU, o que tem a mesma complexidade da implementação em GPU apresentada na subseção anterior.

4.4.4 Filtro 2

Agora é proposta uma implementação em GPU de um novo filtro de vizinhança, como uma alternativa ao filtro apresentado na Subsecção 4.4.3. Seja τ_1 uma permutação de tamanho m onde os vértices no conjunto S são ordenados em ordem crescente com base em seus valores de ganho, e τ_2 uma permutação de tamanho $n-m$ onde os vértices no conjunto $V \setminus S$ são ordenados em ordem decrescente com base em seus valores de ganho. Nesta nova estratégia de filtragem, é proposto restringir as trocas elegíveis para os casos em que o primeiro vértice pertence aos primeiros $\lceil m \times \omega \rceil$ elementos de τ_1, e o segundo vértice pertence aos primeiros $\lceil (n-m) \times \omega \rceil$ elementos de τ_2. Aqui, $\omega \in [0, 1]$ é um parâmetro definido como 0.2 nos experimentos.

A implementação dessa estratégia de filtragem é semelhante à apresentada na Subsecção 4.4.3. A principal diferença é que agora, em vez de criar os conjuntos U_S^c e $U_{V \setminus S}^c$, são criadas as permutações τ_1 e τ_2. Na implementação, essas permutações são criadas usando um algoritmo paralelo de ordenação radix, o qual pode ser realizado em $O(b(k/p + \log(p)))$, onde b e k são, respectivamente, o número de bits nos valores a serem ordenados e o tamanho da permutação.

Capítulo 5

Problema de Dispersão Máxima Média Ponderada

Neste capítulo é abordada a definição do Dispersão Máxima Média Ponderada, abordando suas características, aplicações e a implementação proposta.

5.1 Introdução

O GMaxMeanDP possui uma ampla gama de aplicações práticas, incluindo a classificação de páginas web, mineração de comunidades e redes de confiança. No entanto, as abordagens existentes até o momento não enfrentam adequadamente os desafios impostos por instâncias de grande escala. Embora algoritmos baseados em estratégias de busca tenham sido desenvolvidos para explorar o espaço de soluções, esses métodos não são eficazes para lidar com instâncias massivas. Isso se deve pois à medida que o tamanho do problema aumenta, os operadores de busca sofrem uma desaceleração significativa. Este capítulo apresenta um algoritmo de busca tabu otimizado para execução em GPU, especialmente projetado para enfrentar as dificuldades associadas a instâncias massivas.

5.2 Caracterização do problema

Dado um grafo completo ponderado $G = (V, E, D, W)$, onde V é o conjunto de n vértices, E é o conjunto de arestas, D é o conjunto de pesos das arestas d_{ij} (com $i \neq j$), que podem ser positivos, negativos ou zero, e W representa os pesos dos vértices w_i (com $i = 1, 2, \ldots, n$),

o problema GMaxMeanDP consiste em selecionar um subconjunto $M \subseteq V$ de modo que a dispersão média ponderada do subgrafo induzido por M seja maximizada (LAI, X.; HAO, J.-K.; GLOVER, 2020). A Figura 5.1 mostra um exemplo de grafo G do GMaxMeanDP e possíveis soluções.

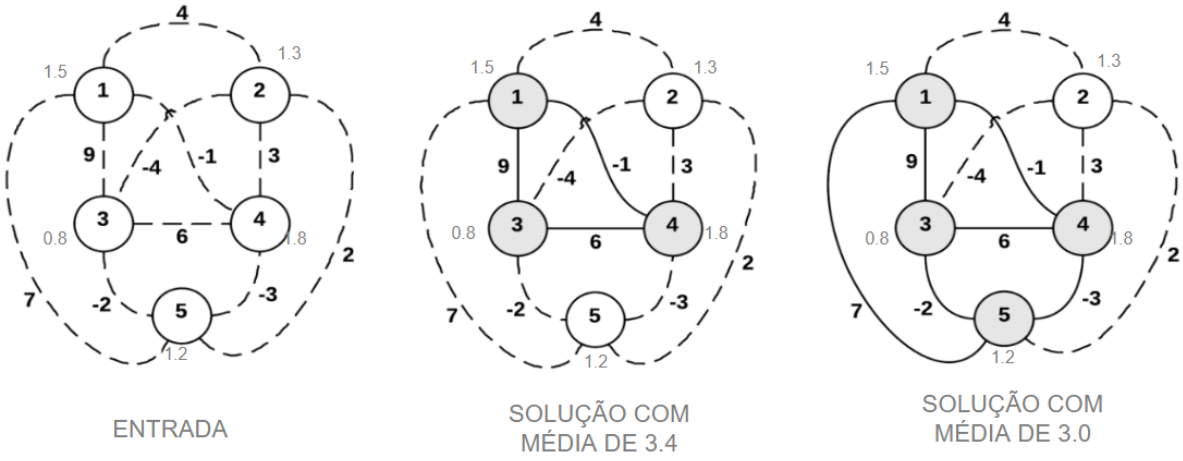


Figura 5.1: Exemplo de um grafo e suas possíveis soluções. Fonte: (CARRASCO et al., 2015)

De maneira formal, o GMaxMeanDP é um problema de otimização combinatória fracionária 0-1 sem restrições, em que as variáveis binárias x_i indicam se o elemento i é selecionado ($x_i = 1$) ou não ($x_i = 0$) (PROKOPYEV; KONG; MARTINEZ-TORRES, 2009b). A formulação matemática do problema pode ser dada por:

$$\text{Maximize } f(s) = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j}{\sum_{i=1}^{n} w_i x_i} \tag{5.1}$$

$$x_i \in \{0, 1\},\ i = 1, 2, \ldots, n$$

onde n é o tamanho do grafo, d_{ij} a aresta entre o vértice i e o vértice j, x_i se o vértice i pertence (1) ou não (0) à solução e w_i o peso do vértice i.

O GMaxMeanDP apresenta semelhanças com o problema MDP. A principal diferença entre eles está na flexibilidade do tamanho do subgrafo e na forma de avaliação das soluções. No MDP, o objetivo é encontrar um subgrafo de tamanho fixo que maximize a soma dos valores associados às arestas entre os vértices do subgrafo. Já no GMaxMeanDP, o tamanho do subgrafo não é fixo, podendo variar dinamicamente durante o processo de busca pela solução ótima. Além disso, no GMaxMeanDP, os elementos possuem pesos atribuídos, tornando-se um problema ponderado. Assim, a solução ótima no GMaxMeanDP é definida pela maximiza-

ção da média ponderada dos valores, enquanto no MDP busca-se a maximização da soma dos valores.

O restante deste trabalho está estruturado da seguinte forma: a Seção 5.3 fornece uma explicação detalhada das implementações atuais de busca tabu sequenciais e destaca suas limitações, particularmente quando aplicadas a instâncias em grande escala. A Seção 5.4 introduz a busca tabu paralela em GPU e demonstra como ela pode superar as limitações descritas na Seção 5.3. O desempenho da proposta do trabalho é avaliado no capítulo 6 e, finalmente, o capítulo 7 apresenta as considerações finais para o trabalho.

5.3 Busca Tabu Sequencial

O algoritmo em GPU proposto aqui é baseado no procedimento tradicional de busca tabu, conforme descrito no Algoritmo 1, que já foi abordado no Capítulo 4. A principal diferença entre eles está na construção das vizinhanças. No caso do MDP, o tamanho do subgrafo é fixo, o que permite apenas a realização de swaps (1-1). No entanto, para o problema GMaxMeanDP, o tamanho do subgrafo pode variar de 2 a n, sendo n o tamanho do grafo. Assim, na abordagem de busca tabu proposta, são aceitos movimentos de inserção ou remoção de 1 ou 2 vértices, além de swaps. O pseudocódigo do algoritmo sequencial é mostrado no Algoritmo 2.

Input: Solução inicial s_0, estrutura de vizinhança $N(s)$, função de avaliação $Media(s)$, quantidade máxima de iterações $Iter_{max}$
Output: Melhor solução encontrada s_b
`BuscaTabu(`$s_0, N(s), f, Iter_{max}$`)`
    $s \leftarrow s_0$ `// Inicializa a solução atual`
    $s_b \leftarrow s$ `// Inicializa a melhor solução`
    $iter \leftarrow 0$ `// Inicializa o contador de iterações`
    repeat
    Escolhe aleatoriamente a melhor solução vizinha elegível $s' \in N(s)$, onde s' é considerada elegível se não estiver proibida pela lista tabu ou se for melhor que s_b
    $s \leftarrow s'$
    Atuaiza a lista tabu
    if $Media(s) > Media(s_b)$ **then**
        $s_b \leftarrow s$ `// Atualiza a melhor solução caso seja encontrada uma melhoria`
    endif
    $iter \leftarrow iter + 1$
until $iter = iter_{max}$
return s_b `// Retorna a melhor solução encontrada`

Algorithm 2: Pseudocódigo da busca tabu sequencial.

Estruturas de Vizinhança. Nesta parte, são investigadas as seguintes três estruturas de vizinhança:

1. **Vizinhança 1-flip.**

   A vizinhança 1-flip (denotada por N_1) é definida pela alteração do valor de uma única variável x_i para o seu complemento $1 - x_i$. O tamanho dessa vizinhança é n, onde n é o número de elementos.

2. **Vizinhança 2-flip.**

   A vizinhança 2-flip (denotada por N_2) altera simultaneamente os valores de duas variáveis x_i e x_j para os seus valores complementares, gerando uma solução vizinha. O tamanho da vizinhança N_2 é dado por $n(n-1)/2$, considerando todas as combinações possíveis de pares de variáveis.

3. **Vizinhança Unificada Reduzida.**

   A vizinhança unificada reduzida (denotada por N_3) combina a vizinhança N_1 com um subconjunto de alta qualidade N_2^* da vizinhança N_2, ou seja, $N_3 = N_1 \cup N_2^*$. Para uma solução s, $N_2^*(s)$ é definido como:

   $$N_2^*(s) = \{s \oplus \text{Flip}\langle i, j\rangle \ : \ i \neq j, \{\Delta_i, \Delta_j\} > \Delta_{\max} - 0.05(\Delta_{\max} - \Delta_{\min})\},$$

   onde $\Delta_{\max} = \max_{l \leq n} \Delta_l$; $\Delta_{\min} = \min_{l \leq n} \Delta_l$; Δ_l representa o valor do movimento ao alterar uma única variável x_l para seu valor complementar, $\text{Flip}\langle i, j\rangle$ representa um movimento de 2-flip que altera simultaneamente os valores das variáveis x_i e x_j para seus valores complementares (LAI, X.; HAO, J.-K.; GLOVER, 2020).

   Assim, o tamanho de N_3 é dado por $n + |N_2^*|$ e é atualizado dinamicamente durante o processo de busca (LAI, X.; HAO, J.-K.; GLOVER, 2020). No algoritmo PBEA, a vizinhança N_3 foi escolhida como estrutura de vizinhança .

O vetor $\text{gain}(v_i)$ na Eq. (4.1) continua sendo calculado e atualizado da mesma maneira que no capítulo 4. Já os valores dos n primeiros ganhos de $\text{gain}(v_i, v_j)$ são calculados usando a fórmula de 1-flip da seguinte forma:

$$\text{gain}(v_i, v_j) = \begin{cases} \frac{-f(s)w_i + gain_i}{S_M + w_i}, & \text{para } x_i = 0; \\ \frac{f(s)w_i - gain_i}{S_M - w_i}, & \text{para } x_i = 1; \end{cases} \tag{5.2}$$

onde $f(s)$ é o valor da solução atual s e S_M é a soma dos pesos dos vértices dos elementos selecionados em s, ou seja, $S_M = \sum_{i \in M} w_i$.

Para os demais casos, ou seja, quando o movimento envolve variáveis x_i e x_j, a fórmula de 2-flip é usada. O valor do movimento $\text{gain}(v_i, v_j)$ é obtido por:

$$\text{gain}(v_i, v_j) = \begin{cases} \frac{-f(s)(w_i+w_j)+gain_i+gain_j+d_{ij}}{S_M+w_i+w_j}, & \text{para } x_i = 0, x_j = 0; \\ \frac{f(s)(w_i+w_j)-gain_i-gain_j+d_{ij}}{S_M-w_i-w_j}, & \text{para } x_i = 1, x_j = 1; \\ \frac{f(s)(w_i-w_j)+gain_j-gain_i+2d_{ij}}{S_M-w_i+w_j}, & \text{para } x_i = 1, x_j = 0; \\ \frac{f(s)(w_j-w_i)+gain_i-gain_j+2d_{ij}}{S_M-w_j+w_i}, & \text{para } x_i = 0, x_j = 1; \end{cases} \tag{5.3}$$

onde d_{ij} é a distância entre os elementos i e j (LAI, X.; HAO, J.-K.; GLOVER, 2020).

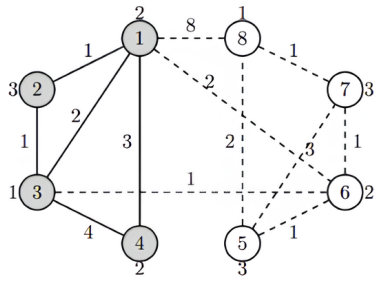
Com o vetor *gain*, o ganho de um movimento de troca pode ser avaliado em $O(1)$ usando a Eq. (5.2). ou a Eq. (5.3). Como foi mencionado, o tamanho da vizinhança é dado por $O(n+|N_2^*|)$. Assim, é necessário $O(n+|N_2^*|)$ para avaliar toda a vizinhança e selecionar o movimento com o melhor ganho. A Figura 5.2 apresenta um exemplo de transição entre soluções. Inicialmente, é exibida uma solução inicial, seguida de uma solução vizinha obtida após a remoção do vértice 2. Posteriormente, é mostrada uma nova solução resultante da inserção dos vértices 5 e 8.

Lista Tabu. A estrutura da lista tabu utilizada segue o mesmo formato descrito no Algoritmo 1. No entanto, no contexto do Algoritmo 2, dependendo do tipo de movimento realizado, é possível que apenas um vértice seja incluído na lista tabu, diferentemente do comportamento observado no Algoritmo 1, onde dois vértices eram sempre adicionados pois o único movimento realizado era o swap.

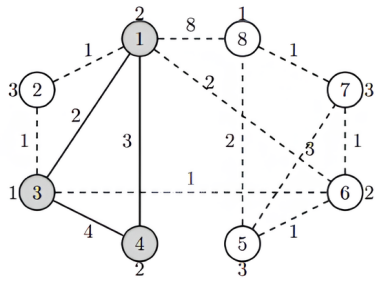
Limitações para Instâncias Massivas. O Algoritmo 2, assim como o Algoritmo 1, possui limitações comuns. Ambos dependem de uma representação baseada em matriz para armazenar as distâncias das instâncias e à medida que o tamanho da instância aumenta, explorar a vizinhança torna-se muito custoso.

5.4 Busca Tabu Paralela Proposta

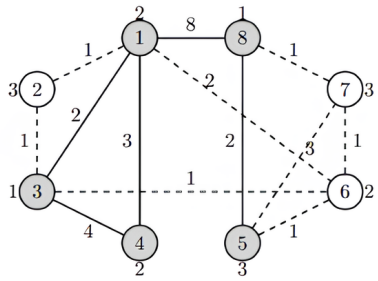
Esta seção introduz a abordagem de busca tabu em GPU proposta. Para superar as limitações dos métodos discutidos na seção anterior, esta abordagem em GPU utiliza estruturas de dados



Vértices	v1	v2	v3	v4	v5	v6	v7	v8
Lista Tabu	0	0	0	0	0	0	0	0
Gain	6	2	7	7	0	2	0	8
Média Solução	1.4							



Vértices	v1	v2	v3	v4	v5	v6	v7	v8
Lista Tabu	0	15	0	0	0	0	0	0
Gain	5	2	6	7	0	3	0	8
Média Solução	1.8							



Vértices	v1	v2	v3	v4	v5	v6	v7	v8
Lista Tabu	0	15	0	0	10	0	0	10
Gain	13	2	6	7	2	4	4	10
Média Solução	2.1							

Figura 5.2: Exemplos de movimentos do GMaxMeanDP. Fonte: Autor

menores para instâncias esparsas e aproveita as capacidades paralelas da GPU para acelerar o processo de busca local.

5.4.1 Estruturas de Dados

Por se tratar de um problema onde o tamanho das soluções é dinâmico, não foi utilizada a W_{swap}. Logo, nesta abordagem, o grafo de uma instância do GMaxMean $G = (V, E)$ é armazenado no formato conhecido como Matriz Triangular Superior (UTM) (BIRKENMEIER et al., 2000), que é um esquema de codificação compacto e eficiente. A Figura 5.3 ilustra um exemplo.

A matriz superior triangular é uma variação de matriz quadrada onde todos os elementos abaixo da diagonal principal são zero. Ela é representada de forma compacta, armazenando apenas os valores não nulos, ou seja, os elementos na diagonal e acima dela. Isso resulta em um armazenamento mais eficiente, reduzindo significativamente a quantidade de memória necessária, especialmente em matrizes grandes.

Se uma matriz normal de tamanho $n \times n$ possui n^2 elementos, a matriz superior triangular, ao armazenar apenas os valores não nulos, possui um tamanho de $\frac{n(n+1)}{2}$. Para grandes valo-

res de n, a redução no uso de memória torna a matriz superior triangular muito vantajosa, especialmente quando se lida com matrizes esparsas.

É importante observar que, a matriz triangular continua com o acesso O(1), porém para acessar cada valor de aresta nela é necessário realizar o cálculo da posição da aresta (i, j):

$$\text{posição} = i \cdot n - i \cdot \frac{i-1}{2} + j - 1 \tag{5.4}$$

onde $i < j$ e $d_{ij} = U_{tm}(posicao)$.

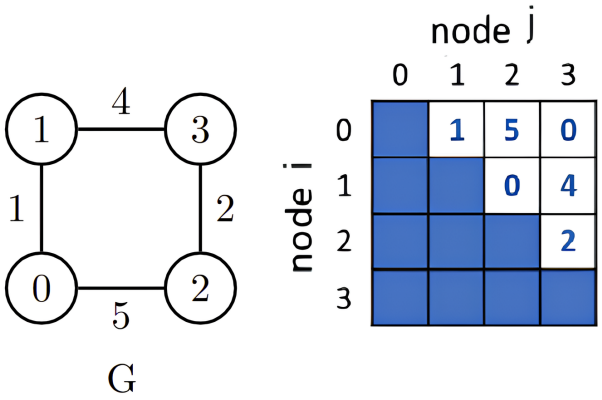


Figura 5.3: Exemplo de grafo G e sua representação UTM correspondente. Fonte: (WONG; MORADI, 2019)

5.4.2 Busca Tabu em GPU

Como mencionado na Seção 4.3, uma das limitações dos métodos atuais baseados em busca local é que, à medida que a instância cresce, eles se tornam menos eficazes. Agora, é descrita uma implementação paralela do Algoritmo 2 para superar essa limitação. Para fins de concisão, a descrição dessa implementação em GPU foca nas diferenças em relação ao Algoritmo 2 (LAI, X.; HAO, J.-K.; GLOVER, 2020).

A primeira diferença em relação ao Algoritmo 2 é que agora é necessário criar e manter as seguintes estruturas de dados na memória da GPU: (i) instância G, representada no formato UTM, (ii) solução atual S e a melhor solução S', representadas como permutações, (iii) vetor gain e a lista tabu. Dada uma solução inicial S, a implementação em GPU começa criando e inicializando essas estruturas de dados na memória da GPU. Em seguida, entra no loop principal (linhas 6-14 do Algoritmo 2).

No loop principal, é preciso encontrar o movimento com o melhor ganho que não esteja na lista tabu (linha 7 do Algoritmo 2). Essa operação é $O(n + n \times \frac{(n-1)}{2})$ e pode se tornar muito custosa à medida que o tamanho da instância cresce. Na implementação em GPU, essa operação é realizada lançando $n + n \times \frac{(n-1)}{2}$ threads na GPU, de modo que cada thread avalia em paralelo o ganho de um movimento e, em seguida, verifica se o movimento correspondente é permitido pela lista tabu. A Figura 5.4 ilustra o processo de cálculo de remoção ou inserção de um vértice, ou seja, da vizinhança N_1 de tamanho n. Já a Figura 5.5 ilustra o processo que altera simultaneamente os valores de duas variáveis, ou seja, da vizinhança N_2 de tamanho $n \times \frac{(n-1)}{2}$. Primeiro, cada thread $x \in [1, \ldots, n+n\times\frac{(n-1)}{2}]$ calcula seu respectivo ganho de movimento usando (5.2). ou (5.3). Em seguida, a thread verifica se o(s) elemento(s) envolvido(s) no movimento estão na lista tabu, ou seja, se $TL[v_i] < iter$ e $TL[v_j] < iter$. Se um desses elementos for proibido pela lista tabu e o movimento não satisfizer o critério de aspiração, um custo de penalização é adicionado ao ganho do movimento. O custo de penalização é um valor constante negativo muito alto.

Após calcular o ganho de cada potencial movimento, o algoritmo emprega uma operação de redução paralela (WILT, 2013) nos ganhos dos movimentos para identificar o melhor movimento elegível. Nos casos em que todos os movimentos potenciais são restritos pela lista tabu, o movimento com o maior ganho entre os movimentos penalizados da lista tabu é selecionado. O movimento selecionado é então aplicado e as estruturas de dados na GPU são atualizadas.

5.4.3 Filtro 1

Na subseção anterior, foi apresentada uma abordagem de busca tabu baseada em GPU que explora exaustivamente a vizinhança N_2. Contudo, explorar todos os movimentos de troca possíveis pode ser computacionalmente caro. Por isso, métodos de última geração utilizam o conceito de estratégia de troca restrita para reduzir o tamanho da vizinhança explorada. Nesta subseção, é proposta uma implementação em GPU da estratégia de troca restrita descrita em (LAI, X.; HAO, J.-K.; GLOVER, 2020).

Dada uma solução S, seja Δ o conjunto de valores obtidos na vizinhança de 1-flip N_1. Cada valor em Δ é calculado alterando o valor de uma única variável $x_i \in V$ para seu valor complementar $1 - x_i$. Esses valores representam o impacto na solução atual S quando o elemento correspondente é adicionado ou removido da solução. A partir de Δ, definimos:

$$\text{minGain} = \min\{\Delta\}, \quad \text{maxGain} = \max\{\Delta\}.$$

Com base nesses valores, é definido o conjunto U_S^c, que contém os elementos de V que apresentam maior potencial de melhoria para a solução. O conjunto U_S^c é dado por:

$$U_S^c = \{v \in V \,:\, \Delta_v > \text{maxGain} - 0.05 \cdot (\text{maxGain} - \text{minGain})\},$$

onde Δ_v representa o valor associado ao vértice v na vizinhança N_1 (LAI, X.; HAO, J.-K.; GLOVER, 2020).

Para incorporar essa estratégia de filtragem na implementação em GPU proposta anteriormente, são realizados os seguintes passos no início de cada iteração do loop principal da busca tabu:

1. Cálculo de minGain e maxGain: Esses valores são obtidos utilizando operações paralelas de redução (*max* e *min*) sobre os valores de Δ. Ambas as operações possuem complexidade $O(\log(n) \times n/p)$.

2. Criação do vetor U_S^c: Usando os valore calculados de maxGain e minGain, o vetor U_S^c é gerado por meio de uma operação paralela de compactação (NOGUEIRA et al., 2024). Essa operação filtra os vértices de V, mantendo apenas aqueles que satisfazem a condição:

$$\Delta_v > \text{maxGain} - 0.05 \cdot (\text{maxGain} - \text{minGain}),$$

onde Δ_v representa o impacto do vértice v na solução atual.

Após gerar U_S^c, todos os movimentos elegíveis na vizinhança restrita são avaliados, e o melhor movimento é selecionado. Com a vizinhança restrita, a complexidade dessa etapa é dada por:

$$O\left(\frac{C}{p} + \log(C) \cdot \frac{C}{p}\right),$$

onde C é o número de combinações possíveis de pares de vértices em U_S^c, definido como:

$$C = \frac{|U_S^c| \cdot (|U_S^c| - 1)}{2}.$$

Após aplicar o movimento selecionado, as estruturas de dados na GPU são atualizadas, seguindo a mesma complexidade apresentada na implementação da subseção anterior.

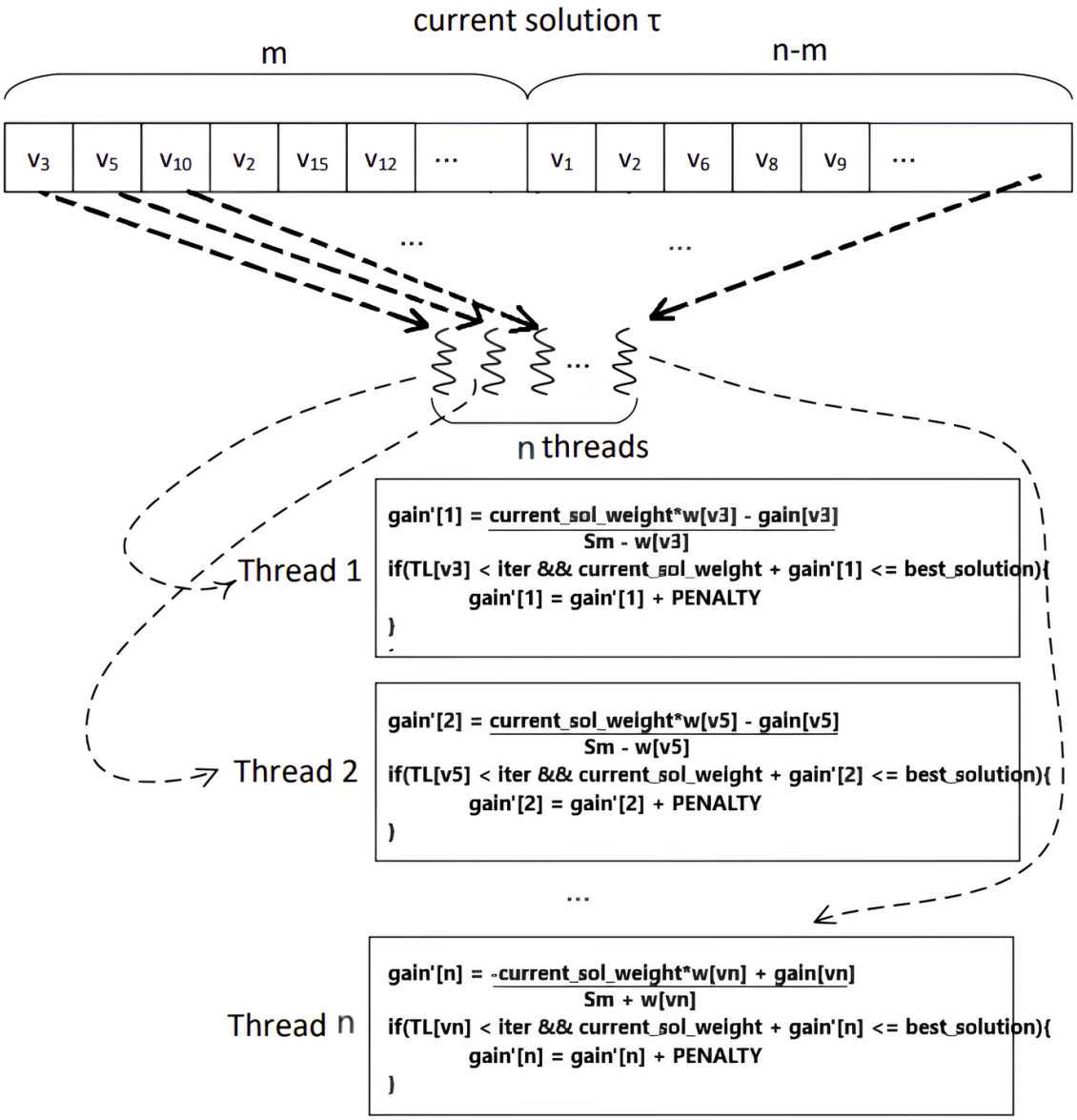


Figura 5.4: Avaliação paralela de inserção ou remoção. Fonte: Autor

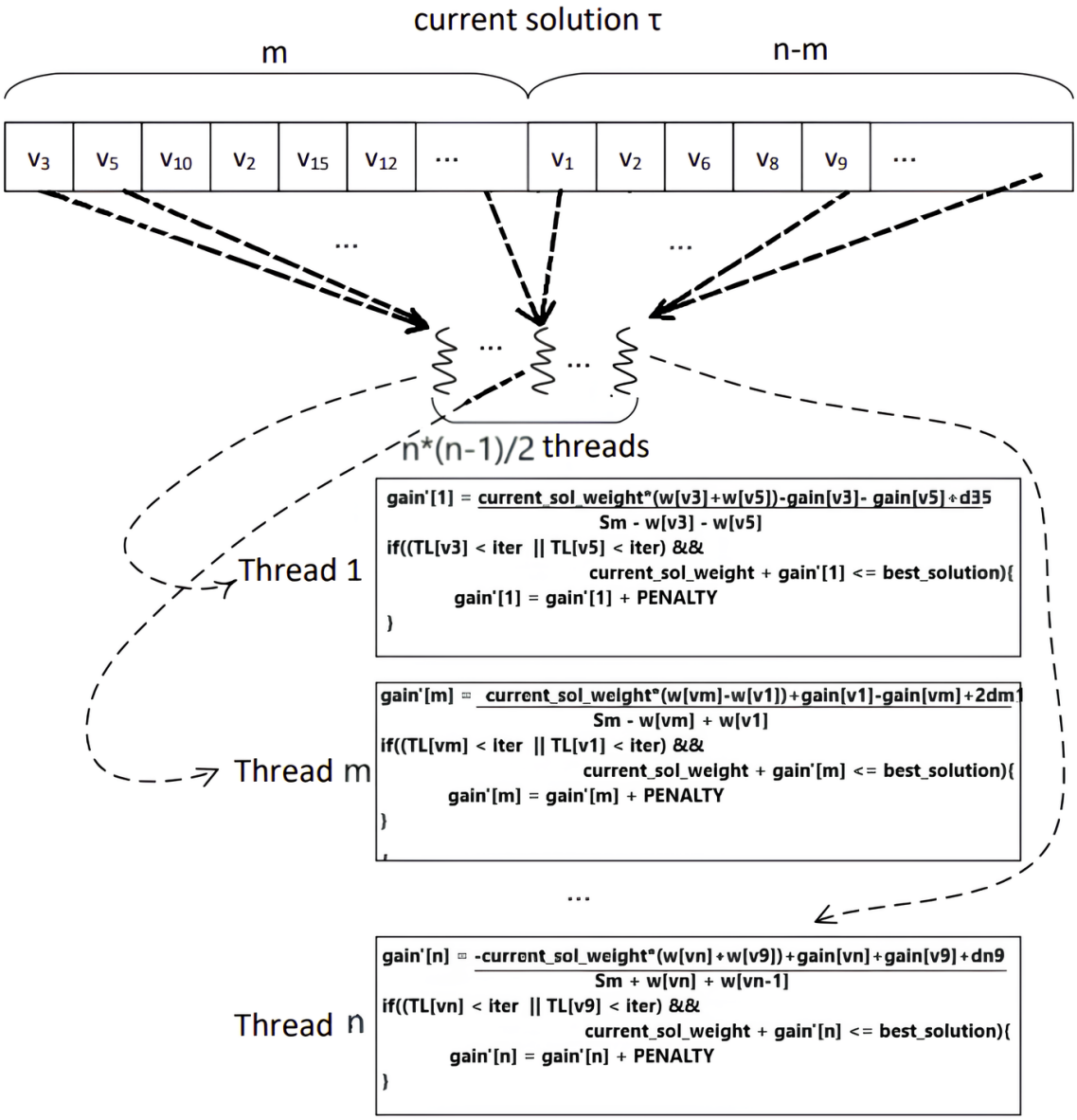


Figura 5.5: Avaliação paralela de movimentos entre mais de um vértice. Fonte: Autor

Capítulo 6

Resultados e Discussão

Este capítulo apresenta os experimentos realizados para avaliar o desempenho das abordagens propostas. A plataforma experimental utilizada consiste em um processador Intel i9-10900KF com 3.7 GHz, 32 GB de memória (utilizando apenas um núcleo da CPU) e uma GPU NVIDIA RTX 3080 com 10 GB de memória. Os algoritmos baseados em GPU foram implementados em CUDA C++ e compilados com o `nvcc` 11.5. Operadores paralelos, como redução, compactação e ordenação por radix, foram implementados utilizando a biblioteca CUB.

Para testar as implementações, os algoritmos foram testados em conjuntos de instâncias amplamente conhecidos, descritos a seguir:

- **Conjunto 1 (MDG)**: Este benchmark consiste em três conjuntos de dados da MDPLIB2: MDG-a (21–40), MDG-b (21–40) e MDG-c (1–20). No total, são 60 matrizes com números reais selecionados aleatoriamente entre 0 e 10 a partir de uma distribuição uniforme. MDG-a e MDG-b possuem $n = 2000$ e $m = 200$, enquanto MDG-c possui $n = 3000$ e $m = \{500, 600\}$.

- **Conjunto 2 (b2500)**: Contém 10 instâncias com $n = 2500$ e $m = 1000$, em que cada distância é um número inteiro gerado aleatoriamente no intervalo $[-100, 100]$. Este benchmark faz parte da ORLIB3.

- **Conjunto 3 (p3000 e p5000)**: O conjunto de dados p3000 contém 5 instâncias com $n = 3000$ e $m = 1500$, enquanto o p5000 possui 5 instâncias com $n = 5000$ e $m = 2500$. Em ambos os casos, as distâncias foram geradas a partir de uma distribuição uniforme $[0, 100]$. Este conjunto de dados foi obtido junto aos seus autores.

- **Conjunto 4 (g 100, h 100, h 300)**: Este benchmark consiste em 16 instâncias esparsas com $n = 1000$ ($h100$), 16 instâncias esparsas com $n = 3000$ ($h300$) e 11 instâncias densas com $n = 1000$ ($g100$). Para este benchmark, consideramos $m = 100$.

- **Conjunto 5 (MDP)**: Este benchmark consiste em dois conjuntos de dado: MDPI (1–10), MDPII (1–10). No total, são 20 matrizes com números reais selecionados aleatoriamente entre -10 e 10 a partir de uma distribuição uniforme. MDP possui $n = 5000$.

- **Conjunto 6 (I_3000_5000)**: Este benchmark consiste em oito conjuntos de dados: I_3000 (1–10), II_3000 (1–10), III_3000 (1–10), IV_3000 (1–10), I_5000 (1–10), II_5000 (1–10), III_5000 (1–10) e IV_5000 (1–10). No total, são 80 matrizes com números reais selecionados aleatoriamente, onde 40 instâncias possuem $n = 3000$ as outras 40 possuem $n = 5000$.

- **Conjunto 7 (SuiteSparse Matrix Collection)**: Este é um grande conjunto de matrizes esparsas provenientes de aplicações reais. Foram selecionados 18 grafos ponderados não direcionados com pelo menos 5000 vértices deste conjunto de dados. Além disso, para o MDP, assume-se que $m = 100$.

Devido à natureza não determinística das abordagens testadas, cada método foi executado 10 vezes em cada instância, utilizando diferentes seeds. Adotou-se um limite de tempo de 60 segundos para cada execução. Quanto aos parâmetros dos algoritmos, foram utilizadas as configurações sugeridas para o OBMA em (ZHOU; HAO, J. K., 2017). e para o PBEA em (LAI, X.; HAO, J.-K.; GLOVER, 2020)

6.1 Resultados para o MDP

Para o MDP, avaliaram-se o desempenho das seguintes implementações, derivadas dos algoritmos descritos na Seção 4.4:

- **GPU-TS**: Representa a implementação baseada em GPU do algoritmo descrito na Subseção 4.4.2, ou seja, a implementação sem mecanismos de filtragem.

- **GPU-TS-f1** e **GPU-TS-f2**: Essas implementações incorporam os mecanismos de filtragem apresentados, respectivamente, nas Subseções 4.4.3 e 4.4.4.

- **GPU-TS-f3**: Combina elementos do **GPU-TS** e do **GPU-TS-f2**. Mais especificamente, dois terços das iterações do tabu são executados sem nenhum mecanismo de filtragem (**GPU-TS**), enquanto no terço restante das iterações o algoritmo emprega o filtro introduzido na Subseção 4.4.4 (**GPU-TS-f2**).

Essas implementações foram comparadas com uma meta-heurística de última geração para o problema, o algoritmo memético OBMA (ZHOU; HAO, J. K., 2017). O código-fonte do OBMA foi obtido com seus autores. O OBMA combina conceitos de algoritmos evolutivos com busca tabu, sendo a busca tabu a parte mais intensiva em termos computacionais. A busca tabu do OBMA segue a abordagem de filtragem apresentada na Subseção 4.4.3. Assim, o **GPU-TS-f1** pode ser considerado uma paralelização em GPU da busca tabu do OBMA.

Para garantir uma comparação justa com o OBMA, que consiste em uma fase evolutiva e uma fase de busca tabu, a fase evolutiva do OBMA foi incluída nas quatro propostas baseadas em GPU. Consequentemente, todos os resultados apresentados nesta seção consideram essa adição. Outra vantagem significativa de integrar nossas propostas paralelas com a parte evolutiva do OBMA é a demonstração de que o método para resolver o MDP pode ser incorporado a outros algoritmos de maneira eficiente. Vale mencionar que, embora exista um algoritmo mais recente para o problema (LIU, X. et al., 2022), que também emprega busca tabu como o OBMA, não foi possível obter seu código-fonte para realizar uma comparação justa.

Para testar os algoritmos foram utilizados os conjuntos de intâncias 1, 2, 3, 4, e 7.

6.1.1 Avaliação de Aceleração e Memória

Iniciou-se analisando a aceleração proporcionada pelas capacidades paralelas da GPU. Neste experimento, foi comparado o número de iterações de busca tabu realizadas pelas implementações em GPU com aquelas realizadas pelo OBMA. Como todos os algoritmos utilizaram o mesmo limite de tempo, o número de iterações é uma boa métrica para estimar a velocidade de exploração da vizinhança.

A Figura 6.1 ilustra a aceleração obtida pelas abordagens em GPU em relação aos algoritmos sequenciais como uma função de n. A aceleração foi calculada, para cada instância, como a razão entre o número médio de iterações do algoritmo em GPU e o número médio de iterações do OBMA original.

Conforme mostrado na Figura 6.1, a aceleração aumenta com n. O experimento revela que para $n > 5000$, a aceleração sempre supera 1, indicando que a GPU oferece vantagens signifi-

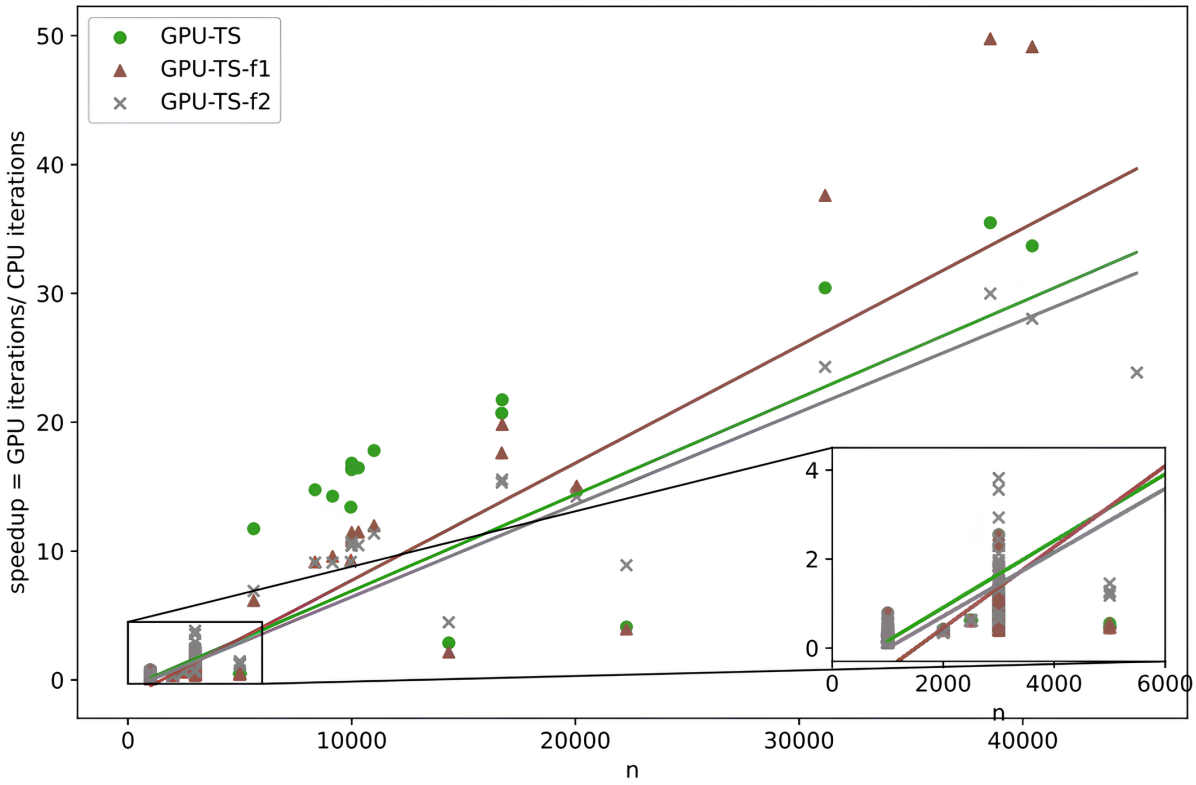


Figura 6.1: Speedup das implementações propostas para o MDP. Fonte: Autor

cativas. Por outro lado, para $n < 3000$, a aceleração tende a ser próxima de 1, sugerindo que o overhead de comunicação entre GPU e CPU se torna um gargalo, tornando o uso da GPU desaconselhável. No intervalo $3000 \leq n \leq 5000$, observam-se resultados mistos. GPU-TS e GPU-TS-f1 alcançam aceleração maior que 1 em apenas 24% dos casos. GPU-TS-f2 consistentemente alcança aceleração maior que 1 para $n \geq 5000$, mas apenas em 40% dos casos para $3000 \leq n < 5000$.

Entre as abordagens em GPU, para o MDP, GPU-TS-f1 destaca-se como o método capaz de atingir as maiores acelerações, chegando a uma aceleração de até 50x (lembrando que GPU-TS-f1 pode ser considerado uma paralelização em GPU da busca tabu do OBMA). No entanto, a versão GPU-TS foi capaz de executar mais iterações do que GPU-TS-f1 e GPU-TS-f2 em algumas instâncias. Isso sugere que a aceleração proporcionada pela redução *swap-(1,1)* nem sempre compensa o overhead de filtragem.

Por fim, é importante mencionar que as implementações em GPU não utilizam representações matriciais n^2 para armazenar instâncias (ver Subseção 4.4.1). Portanto, além de executar a busca tabu de forma mais rápida em instâncias grandes, também é necessário menos memória do que heurísticas sequenciais atuais. Isso é claramente demonstrado na Figura 6.2, que

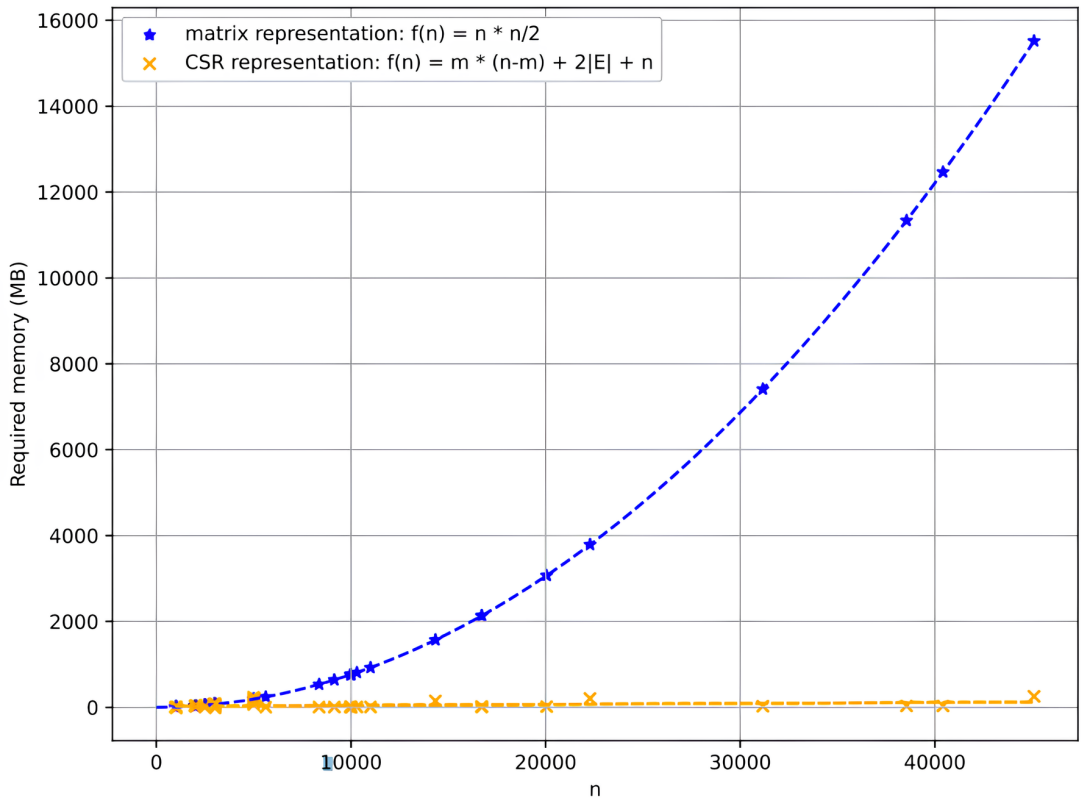


Figura 6.2: Análise da memória necessária. Fonte: Autor

compara, para cada instância, a memória necessária para a representação matricial versus a representação em CSR.

6.1.2 Análise da Qualidade das Soluções

Esta subseção analisa o impacto da aceleração via GPU na qualidade das soluções. A tabela 6.1 apresenta uma comparação resumida entre os algoritmos para cada conjunto de instâncias. A primeira coluna indica o conjunto de instâncias, a segunda mostra os valores de n para cada conjunto enquanto as demais apresentam os valores médios dos *gaps* obtidos pelos algoritmos. O valor do *gap* é calculado como $\text{gap} = \frac{\text{Weight}(S^*) - \text{Weight}(S')}{\text{Weight}(S^*)}$, onde S^* e S' são, respectivamente, a melhor solução encontrada entre os métodos e a melhor solução encontrada pelo método em análise.

A análise de aceleração apresentada na Subseção 6.3 indicou que, para $n \leq 5000$, o uso da GPU apresenta ganhos de aceleração baixos. Esta observação é corroborada pela Tabela 6.1, onde, exceto para os conjuntos de instâncias 4 (*g_100, h_100, h_300*) e 5 (*SuiteSparse Matrix*), o OBMA demonstra desempenho competitivo em comparação com as implementações em GPU. Os resultados também revelam que o GPU-TS é geralmente inferior às abordagens em

GPU que utilizam filtros para limitar o tamanho da vizinhança. Isso ocorre porque, além de acelerar a exploração da vizinhança, as abordagens com filtragem intensificam a busca em trocas promissoras, descartando movimentos que não devem trazer ganhos. Ademais, a Tabela 6.1 indica que o GPU-TS-f3 é o método mais robusto, consistentemente alcançando os melhores *gaps* entre todos os métodos.

Tabela 6.1: Comparação dos Gaps dos algoritmos para o MDP

Instance set	n	OBMA	GPU-TS	GPU-TS-f1	GPU-TS-f2	GPU-TS-f3
SuiteSparse	[5620, 45101]	0.5829	0.211	0.3405	0.1851	**0.1741**
p30005000	[3000, 5000]	< **0.0001**	< **0.0001**	< **0.0001**	< **0.0001**	< **0.0001**
h_100/h_300/g_100	[1000, 3000]	0.005	0.0016	0.0009	0.0012	**0.0006**
b2500	2500	< **0.0001**	< **0.0001**	< **0.0001**	< **0.0001**	< **0.0001**
MDG	[2000, 3000]	**0.0001**	0.0003	0.0002	0.0003	**0.0001**

As tabelas 6.3 e 6.4 apresentam resultados detalhados para cada instância testada envolvendo o problema MDP. Nas tabelas 6.3 e 6.4, as colunas rotuladas como *Instance*, n e d representam o nome da instância, o valor de n e a densidade, respectivamente. Além disso, as colunas *Avg.* e *Gap* apresentam o valor médio da solução obtido em todas as execuções do método e o *gap* correspondente. O final da tabela inclui um resumo que mostra, para cada método, o número de vezes que ele atingiu a melhor solução média entre os métodos e o *gap* médio total em todas as instâncias.

Os resultados nas tabelas 6.3 e 6.4 indicam que o GPU-TS-f2 é superior ao GPU-TS-f1, alcançando um *gap* médio total menor (0,1866) e determinando a melhor solução média mais vezes (58). Além disso, esta tabela mostra que o OBMA foi o método que mais frequentemente determinou a melhor solução média (90), com o GPU-TS-f3 em segundo lugar (59). Isso pode ser explicado pelo fato de que a maioria das instâncias possui $n \leq 3000$. Como indicado pelos resultados na Subseção 6.1, o uso de algoritmos em GPU nesses casos não é aconselhável. No entanto, o OBMA obteve o pior *gap* médio total entre os métodos (0,5879). Em contraste, o GPU-TS-f3 alcançou o melhor valor para esta métrica, 0,1747, três vezes melhor que o *gap* obtido pelo OBMA.

Uma comparação direta entre o OBMA e a versão em GPU de melhor desempenho (GPU-TS-f3) em instâncias com $n > 3000$ revela que o GPU-TS-f3 supera ou iguala o *gap* do OBMA em 18 das 23 instâncias e atinge uma solução média melhor ou igual em 21 instâncias. Em comparação, o OBMA consegue este feito em apenas 10 e 4 instâncias, respectivamente. Isso

sugere que: (i) embora o OBMA se destaque em cenários menores, seu desempenho se deteriora à medida que o tamanho das instâncias aumenta, e (ii) optar pelos métodos propostos em GPU, especialmente a versão GPU-TS-f3, prova ser uma alternativa altamente eficaz para instâncias maiores.

6.2 Resultados para o GMaxMeanDP

Para o GMaxMeandDP, avaliou-se o desempenho da seguinte implementação, derivada dos algoritmos descritos na Seção 5.4:

- **GPU**: Representa a implementação baseada em GPU do algoritmo descrito na Subseção 5.4.3, ou seja, a implementação com filtragem e vizinhança N_3.

Essa implementação foi comparada com uma meta-heurística para o problema, o algoritmo memético PBEA (LAI, X.; HAO, J.-K.; GLOVER, 2020). O PBEA é um algoritmo evolutivo baseado em perturbação, ele combina os conceitos de algoritmos evolutivos com busca tabu, sendo a busca tabu a parte mais intensiva em termos computacionais. A busca tabu do PBEA segue a abordagem de filtragem apresentada na Subseção 5.4.3. Assim, o **GPU** pode ser considerado uma paralelização em GPU da busca tabu do PBEA.

Para testar os algoritmos foram utilizados os conjuntos de intâncias 5, 6 e 7.

6.2.1 Avaliação de Aceleração e Memória

Para o GMMDP, também iniciou-se analisando a aceleração proporcionada pelas capacidades paralelas da GPU. Neste experimento, foi comparado o número de iterações de busca tabu realizadas pelas implementações em GPU com aquelas realizadas pelo PBEA.

A Figura 6.3 ilustra a aceleração obtida pela abordagem em GPU em relação ao algoritmo sequencial como uma função de n. A aceleração foi calculada, para cada instância, como a razão entre o número médio de iterações do algoritmo em GPU e o número médio de iterações do PBEA original.

Conforme mostrado na Figura 6.3, a aceleração aumenta com n. O experimento revela que também para $n > 5000$, a aceleração sempre supera 1, indicando que a GPU oferece vantagens significativas. Por outro lado, para $n < 3000$, a aceleração também tende a ser próxima de 1, sugerindo que o overhead de comunicação entre GPU e CPU se torna um gargalo, tornando o uso da GPU desaconselhável.

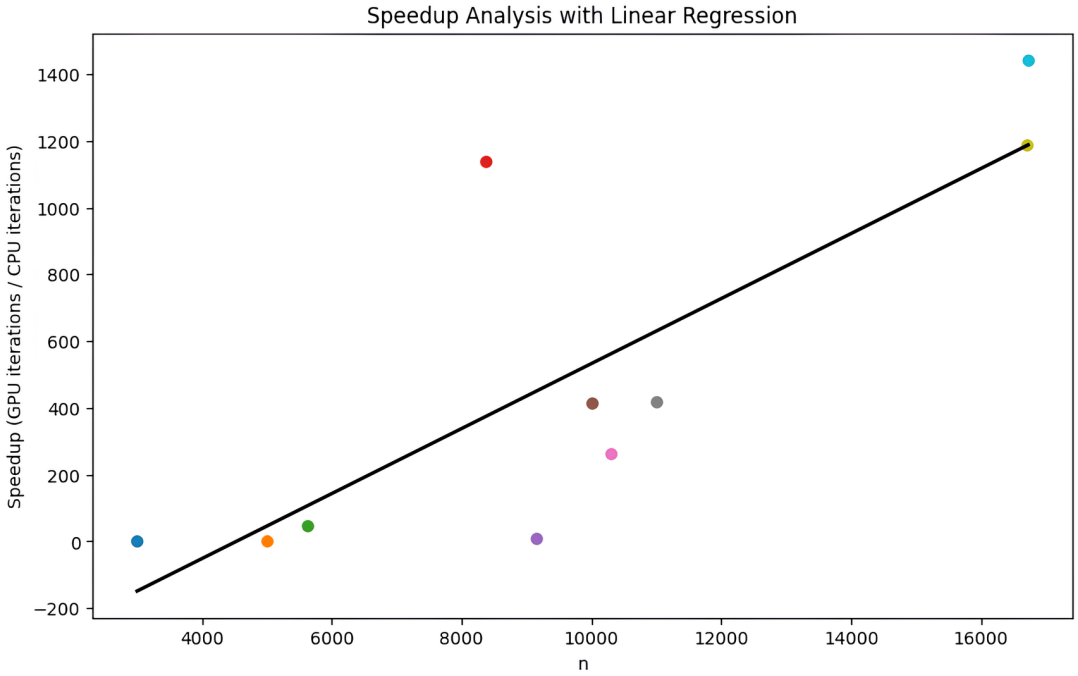


Figura 6.3: Speedup da implementação proposta para o GMMDP. Fonte: Autor

Para o GMMDP, a implementação em GPU destaca-se como o método capaz de atingir uma acelerações muito altas,(lembrando que GPU pode ser considerado uma paralelização em GPU da busca tabu do PBEA).

6.2.2 Análise da Qualidade das Soluções

Esta subseção analisa o impacto da aceleração via GPU na qualidade das soluções. A tabela 6.2 apresenta uma comparação entre os algoritmos para cada conjunto de instâncias.

A análise de aceleração apresentada na Subseção 6.2.1 também indicou que, para $n \leq 5000$, o uso da GPU apresenta ganhos de aceleração baixos. A Tabela 6.2 indica que o algoritmo proposto para o GMaxMeanDP em GPU é o método mais robusto, consistentemente alcançando os melhores *gaps* entre os método sequencial.

Tabela 6.2: Comparação dos Gaps dos algoritmos para o GMaxMeanDP

Instance set	n	CPU-PBEA	GPU-GMMDP
SuiteSparse	[5620, 45101]	2.4856	0.0017
MDP	5000	0.0053	0.0037
I_3000_5000	[3000, 5000]	0.0074	0.0043

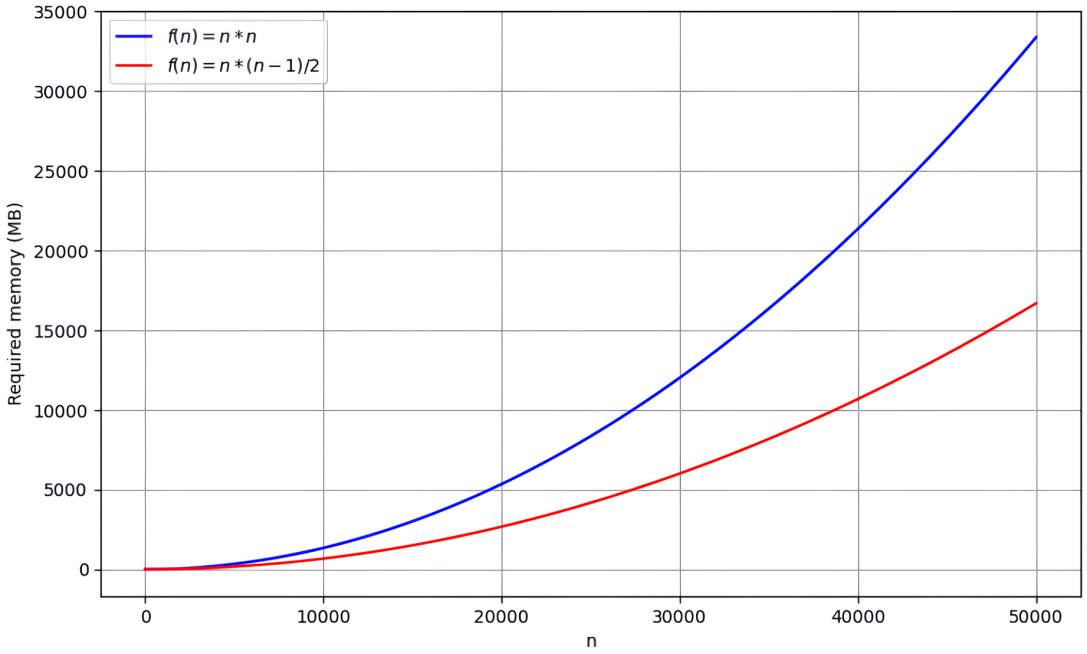


Figura 6.4: Análise da memória necessária. Fonte: Autor

As tabelas 6.5 e 6.6 apresentam resultados detalhados para cada instância testada envolvendo o GMaxMeanDP. Nas tabelas 6.5 e 6.6, as colunas rotuladas como *Instance* e N representam o nome da instância, e o valor de N, respectivamente. Além disso, as colunas *Avg.*, *Best* e *Gap* apresentam o valor médio da solução obtido em todas as execuções do método, a melhor solução encontrada e o *gap* correspondente.

Os resultados nas tabelas 6.5 e 6.6 indicam que o algoritmo em GPU é superior ao algoritmo em CPU, alcançando um *gap* médio total de (0,0100) e determinando a melhor solução média mais vezes (63). Além disso, esta tabela mostra que o PBEA em CPU determinou a melhor solução média em (40) vezes. Isso pode ser explicado pelo fato de que a maioria das instâncias possui $n \leq 3000$. Como indicado pelos resultados na Subseção 6.1, o uso de algoritmos em GPU nesses casos não é aconselhável. No entanto, o PBEA em CPU obteve o pior *gap* médio total entre os métodos (2.4984).

6.3 Discussão dos Resultados

Os resultados experimentais apresentados nas Subseções 6.1 e 6.2 demonstraram que as propostas em GPU, especialmente a versão GPU-TS-f3 para o MDP e a versão em GPU para o GMMDP, são muito eficazes em instâncias maiores. Os resultados também revelaram as limitações dos métodos *state-of-the-art* em instâncias grandes.

Para os dois problemas apresentandos nos capitulos 4 e 5, as duas principais limitações dos algoritmos sequenciais foram: (i) sua representação baseada em matrizes e (ii) aumento das combinações da busca local conforme o problema aumenta de tamanho. O primeiro problema foi abordado empregando estruturas de dados mais eficientes, chamadas CSR (Subseção 4.4.1) e a UTM (Subseção 5.4.1). Essas estruturas não apenas melhoram a localidade de espaço, mas também reduz os requisitos de memória. Contudo, diferente da estrutura matricial, a CSR não possui tempo de acesso $O(1)$. Para mitigar isso, foi introduzida uma nova estratégia para explorar a vizinhança *swap-(1,1)*, que aproveita a CSR enquanto mantém a complexidade temporal da representação matricial.

O segundo problema (i.e., busca local de alto custo) foi resolvido utilizando a GPU. Para o MDP, foi proposto quatro métodos para explorar a vizinhança *swap-(1,1)* com GPU: GPU-TS, GPU-TS-f1, GPU-TS-f2 e GPU-TS-f3. A versão GPU-TS-f1 é essencialmente uma versão paralelizada em GPU da busca tabu do OBMA. Embora esta versão tenha melhorado os resultados do OBMA em instâncias maiores, seus resultados foram inferiores à versão GPU-TS-f3. O GPU-TS-f3 adota uma nova estratégia para explorar a vizinhança *swap-(1,1)*, começando a busca sem filtrar movimentos e, ao final, utiliza o novo filtro proposto na Seção 4.4.4 para intensificar a busca em regiões promissoras. Uma comparação direta entre o OBMA sequencial e o GPU-TS-f3 nas instâncias maiores ($n > 3000$) mostrou que a melhor versão em GPU melhora ou encontra a mesma qualidade média de solução em 21 das 24 instâncias, enquanto esse número é 4 para o OBMA. Já para o GMMDP, foi proposto um método para explorar a vizinhança com GPU. Essa versão em GPU é essencialmente uma versão paralelizada em GPU da busca tabu do PBEA. O algoritmo GPU adota uma estratégia para explorar a vizinhança utilizando o filtro proposto na Seção 5.4.3 para intensificar a busca em regiões promissoras. Uma comparação direta entre o PBEA sequencial e o em GPU nas instâncias maiores ($n > 3000$) mostrou que a melhor versão em GPU melhora ou encontra a mesma qualidade média de solução em 57 das 70 instâncias, enquanto esse número é 13 para o PBEA.

Tabela 6.3: Comparação dos Resultados para o MDP

Instance	n	d	OBMA		GPU-TS		GPU-TS-f1		GPU-TS-f2		GPU-TS-f3	
			Avg.	Gap	Avg.	Gap	Avg.	Gap	Avg.	Gap	Avg.	Gap
Fashion_MNIST_nor...	10000	0.0016	32	0.0095	34	**0**	34	0.0062	34	0.0085	34	0.0048
JapaneseVowelsSmall...	9961	0.0013	92	0.0319	104	0.0136	107	**0**	106	0.0232	107	0.0131
astro-ph	16706	0.0009	987	0.1294	1131	**0**	1145	**0**	11	**0**	113	**0**
cond-mat-2003	31163	0.0002	908	0.0443	956	**0**	905	0.0743	962	< 0.0001	948	0.0082
cond-mat-2005	40421	0.0002	1033	0.0684	11	0.0284	1055	0.0679	1102	0.0359	111	**0**
cond-mat	16726	0.0003	594	**0**	626	0.0152	604	0.0441	621	0.0394	619	0.0331
har_10NN	10299	0.0014	28	0.0278	31	0.0246	32	**0**	31	0.0278	31	0.0278
hep-th	8361	0.0005	796	0.005	82	0.006	795	0.0299	818	**0**	817	0.005
human_gene1	22283	0.0496	549	**0**	549	**0**	549	**0**	549	**0**	549	**0**
human_gene2	14340	0.0878	54	**0**	54	**0**	54	**0**	54	**0**	54	**0**
indianpines_10NN	9144	0.0015	48	0.0201	53	0.0298	53	**0**	52	0.0201	53	0.0276
k49_norm_10NN	38547	0.0004	29	0.011	34	0.011	34	0.011	34	**0**	34	**0**
kmnist_norm_10NN	10000	0.0016	26	0.0079	26	0.0071	26	0.0035	26	**0**	26	0.0061
mnist_test_norm_10NN	10000	0.0015	32	0.017	34	0.0129	34	**0**	34	0.0129	34	0.0092
mouse_gene	45101	0.0142	436	**0**	591	**0**	7	**0**	808	**0**	655	**0**
optdigits_10NN	5620	0.0025	36	0.0829	42	0.0111	41	**0**	4	0.0119	41	0.0001
usps_norm_5NN	11000	0.0007	16	0.0953	17	0.0082	16	0.1035	17	**0**	17	0.0011
worms20_10NN	20055	0.0006	134	0.0324	137	0.0431	14	**0**	14	0.0052	138	0.0381
b2500-1	2500	0.0995	1153063	**0**	1153068	**0**	1153064	**0**	115304	**0**	1152911	**0**
b2500-2	2500	0.0991	1129254	**0**	1129133	**0**	1129141	**0**	1129079	**0**	1129074	**0**
b2500-3	2500	0.0989	1115537	**0**	1115538	**0**	111546	**0**	1115462	**0**	111523	**0**
b2500-4	2500	0.0991	1147739	**0**	1147765	**0**	1147672	**0**	114769	**0**	1147542	**0**
b2500-5	2500	0.0991	1144756	**0**	1144708	**0**	114474	**0**	114474	**0**	1144756	**0**
b2500-6	2500	0.0989	1133572	**0**	1133572	**0**	1133564	**0**	1133558	**0**	1133545	**0**
b2500-7	2500	0.0993	1149046	**0**	1149009	**0**	1149001	**0**	1149023	**0**	1149006	**0**
b2500-8	2500	0.099	1142762	**0**	1142754	**0**	1142739	**0**	1142757	**0**	1142747	**0**
b2500-9	2500	0.0992	1138865	**0**	1138799	**0**	1138845	**0**	1138831	**0**	1138833	**0**
b2500-10	2500	0.099	1153936	**0**	1153916	**0**	1153866	**0**	11539	**0**	1153878	**0**
MDG-a_21_n2000_m200	2000	1	114271	**0**	114271	**0**	11427	**0**	114271	**0**	11427	**0**
MDG-a_22_n2000_m200	2000	1	114327	**0**	114327	**0**	114327	**0**	114327	**0**	114327	**0**
MDG-a_23_n2000_m200	2000	1	114195	**0**	114193	**0**	114194	**0**	114192	**0**	114193	**0**
MDG-a_24_n2000_m200	2000	1	114093	**0**	114089	**0**	114092	**0**	11409	**0**	11409	**0**
MDG-a_25_n2000_m200	2000	1	114196	**0**	114195	**0**	114195	**0**	11418	**0**	114189	**0**
MDG-a_26_n2000_m200	2000	1	114265	**0**	114264	**0**	114265	**0**	114264	**0**	114265	**0**
MDG-a_27_n2000_m200	2000	1	114361	**0**	114361	**0**	114361	**0**	114361	**0**	114361	**0**
MDG-a_28_n2000_m200	2000	1	114327	**0**	114327	**0**	114327	**0**	114327	**0**	114327	**0**
MDG-a_29_n2000_m200	2000	1	114199	**0**	114199	**0**	114199	**0**	114199	**0**	114198	**0**
MDG-a_30_n2000_m200	2000	1	114229	**0**	114228	**0**	114228	**0**	114229	**0**	114228	**0**
MDG-a_31_n2000_m200	2000	1	114211	**0**	114212	**0**	114211	**0**	114211	**0**	114214	**0**
MDG-a_32_n2000_m200	2000	1	114208	**0**	114208	**0**	114211	**0**	114199	**0**	114211	**0**
MDG-a_33_n2000_m200	2000	1	114233	**0**	114233	**0**	114233	**0**	114233	**0**	114233	**0**
MDG-a_34_n2000_m200	2000	1	114216	**0**	114216	**0**	114216	**0**	114216	**0**	114216	**0**
MDG-a_35_n2000_m200	2000	1	11424	**0**	114239	**0**	114239	**0**	11424	**0**	11424	**0**
MDG-a_36_n2000_m200	2000	1	114335	**0**	114335	**0**	114335	**0**	114335	**0**	114335	**0**
MDG-a_37_n2000_m200	2000	1	114255	**0**	114255	**0**	114255	**0**	114255	**0**	114255	**0**
MDG-a_38_n2000_m200	2000	1	114408	**0**	114408	**0**	114408	**0**	114408	**0**	114408	**0**
MDG-a_39_n2000_m200	2000	1	114201	**0**	114201	**0**	114201	**0**	114201	**0**	114201	**0**
MDG-a_40_n2000_m200	2000	1	114349	**0**	114349	**0**	114349	**0**	114349	**0**	114349	**0**
MDG-b_21_n2000_m200	2000	1	11299895	**0**	11299895	**0**	11299895	**0**	11299895	**0**	11299895	**0**
MDG-b_22_n2000_m200	2000	1	11292398	**0**	11292081	**0**	11292256	**0**	11291732	**0**	11292282	**0**
MDG-b_23_n2000_m200	2000	1	1129994	**0**	1129994	**0**	1129994	**0**	1129994	**0**	1129994	**0**
MDG-b_24_n2000_m200	2000	1	11291095	**0**	11290988	**0**	11291065	**0**	11290923	**0**	11291037	**0**
MDG-b_25_n2000_m200	2000	1	11298025	**0**	11297967	**0**	11297989	**0**	11298016	**0**	11297933	**0**
MDG-b_26_n2000_m200	2000	1	1129843	**0**	1129843	**0**	1129843	**0**	1129843	**0**	1129797	**0**
MDG-b_27_n2000_m200	2000	1	11305677	**0**	11305677	**0**	11305677	**0**	11305677	**0**	11305677	**0**
MDG-b_28_n2000_m200	2000	1	11282891	**0**	11282744	**0**	11282868	**0**	11282788	**0**	1128285	**0**
MDG-b_29_n2000_m200	2000	1	11297339	**0**	11297319	**0**	11297324	**0**	11297294	**0**	11297319	**0**
MDG-b_30_n2000_m200	2000	1	11298065	**0**	11297989	**0**	11297978	**0**	11297943	**0**	11297984	**0**
MDG-b_31_n2000_m200	2000	1	11288901	**0**	11288901	**0**	11288901	**0**	11288901	**0**	112889	**0**
MDG-b_32_n2000_m200	2000	1	11283489	**0**	11283481	**0**	11283502	**0**	11283502	**0**	11283502	**0**
MDG-b_33_n2000_m200	2000	1	11297913	**0**	11297727	**0**	11297855	**0**	11297989	**0**	11297781	**0**
MDG-b_34_n2000_m200	2000	1	11290483	**0**	11289658	**0**	11289979	**0**	11290483	**0**	11290216	**0**
MDG-b_35_n2000_m200	2000	1	11307424	**0**	11307424	**0**	11307424	**0**	11307424	**0**	11307424	**0**
MDG-b_36_n2000_m200	2000	1	11302892	**0**	11302866	**0**	11302886	**0**	11302492	**0**	11302892	**0**
MDG-b_37_n2000_m200	2000	1	11295483	**0**	11295726	**0**	11295647	**0**	11295481	**0**	11295605	**0**
MDG-b_38_n2000_m200	2000	1	11296536	**0**	11296394	**0**	11296509	**0**	11296407	**0**	11296381	**0**
MDG-b_39_n2000_m200	2000	1	11295054	**0**	11295045	**0**	1129503	**0**	11295045	**0**	11295009	**0**
MDG-b_40_n2000_m200	2000	1	11309163	**0**	11309163	**0**	11309143	**0**	1130896	**0**	11309163	**0**
MDG-c_1_n3000_m300	3000	1	24924093	**0**	24923113	**0**	24922892	**0**	2492295	< 0.0001	24924083	**0**
MDG-c_2_n3000_m300	3000	1	24908761	**0**	24908672	**0**	24909312	< 0.0001	24906478	< 0.0001	24908878	< 0.0001
MDG-c_3_n3000_m300	3000	1	24897268	**0**	24899252	**0**	24897143	**0**	24898301	**0**	24897341	**0**

Tabela 6.4: Comparação dos Resultados para o MDP

Instance	n	d	OBMA		GPU-TS		GPU-TS-f1		GPU-TS-f2		GPU-TS-f3	
			Avg.	Gap	Avg.	Gap	Avg.	Gap	Avg.	Gap	Avg.	Gap
MDG-c_4_n3000_m300	3000	1	24905614	**0**	24903886	0.0002	24904698	**0**	24903461	0.0002	24904975	**0**
MDG-c_5_n3000_m300	3000	1	24893806	**0**	24893663	**0**	24893782	< 0.0001	24892941	**0**	24893896	< 0.0001
MDG-c_6_n3000_m400	3000	1	43441351	< 0.0001	43439549	< 0.0001	43439895	< 0.0001	43440547	**0**	4344017	**0**
MDG-c_7_n3000_m400	3000	1	43476931	**0**	4347556	**0**	43473342	< 0.0001	43472635	< 0.0001	43474584	< 0.0001
MDG-c_8_n3000_m400	3000	1	43461974	**0**	4346175	< 0.0001	43462468	**0**	43461283	**0**	43463643	**0**
MDG-c_9_n3000_m400	3000	1	43447983	**0**	43447682	**0**	43447929	**0**	43447522	**0**	43446852	**0**
MDG-c_10_n3000_m400	3000	1	43465283	**0**	43465275	**0**	43464696	**0**	43465144	**0**	43465111	**0**
MDG-c_11_n3000_m500	3000	1	67020052	**0**	67018935	**0**	67018372	< 0.0001	67018466	**0**	6701883	**0**
MDG-c_12_n3000_m500	3000	1	67012048	**0**	67007759	< 0.0001	6700055	< 0.0001	67000366	**0**	6700402	**0**
MDG-c_13_n3000_m500	3000	1	67019281	**0**	67018252	**0**	67016717	**0**	67016573	**0**	67016006	< 0.0001
MDG-c_14_n3000_m500	3000	1	67027748	**0**	67024486	**0**	67026719	**0**	67022727	< 0.0001	67027004	**0**
MDG-c_15_n3000_m500	3000	1	67054104	**0**	6705345	**0**	67052668	< 0.0001	67052421	**0**	67053858	**0**
MDG-c_16_n3000_m600	3000	1	95637422	**0**	9563697	**0**	95637018	**0**	95635319	**0**	95637403	**0**
MDG-c_17_n3000_m600	3000	1	95567935	**0**	95563604	< 0.0001	95562331	< 0.0001	95564778	**0**	95566396	< 0.0001
MDG-c_18_n3000_m600	3000	1	95525197	**0**	95523428	**0**	95523437	< 0.0001	95523904	< 0.0001	95524766	**0**
MDG-c_19_n3000_m600	3000	1	95596794	< 0.0001	95593262	< 0.0001	95592864	**0**	95591038	< 0.0001	95595593	< 0.0001
MDG-c_20_n3000_m600	3000	1	95581594	**0**	95579194	**0**	95579359	< 0.0001	95580108	**0**	95580499	**0**
g_100_1000	1000	0.2002	983744	**0**	983744	**0**	983744	**0**	983744	**0**	983744	**0**
g_100_1100	1000	0.2202	106033	**0**	106033	**0**	106033	**0**	106033	**0**	106033	**0**
g_100_1200	1000	0.2402	1121136	**0**	1121136	**0**	1121136	**0**	1121136	**0**	1121136	**0**
g_100_1300	1000	0.2603	1191238	**0**	1191238	**0**	1191238	**0**	1191238	**0**	1191238	**0**
g_100_1400	1000	0.2803	1257993	**0**	1257993	**0**	1257993	**0**	1257993	**0**	1257993	**0**
g_100_1500	1000	0.3003	1323428	**0**	1323428	**0**	1323428	**0**	1323428	**0**	1323428	**0**
g_100_1600	1000	0.3203	138742	**0**	138742	**0**	138742	**0**	138742	**0**	138742	**0**
g_100_1700	1000	0.3403	1452116	**0**	1452116	**0**	1452116	**0**	1452116	**0**	1452116	**0**
g_100_1800	1000	0.3604	1505436	**0**	1505436	**0**	1505436	**0**	1505436	**0**	1505436	**0**
g_100_1900	1000	0.3804	1569826	**0**	1569826	**0**	1569826	**0**	1569826	**0**	1569826	**0**
g_100_2000	1000	0.4004	1629915	**0**	1629915	**0**	1629915	**0**	1629915	**0**	1629915	**0**
h_100_05	1000	0.01	36462	**0**	36462	**0**	36462	**0**	36457	**0**	36463	**0**
h_100_06	1000	0.012	40104	**0**	40104	**0**	40104	**0**	40104	**0**	40104	**0**
h_100_07	1000	0.014	43707	**0**	43707	**0**	43707	**0**	43707	**0**	43707	**0**
h_100_08	1000	0.016	46245	**0**	46238	**0**	46245	**0**	46241	**0**	46242	**0**
h_100_09	1000	0.018	48618	**0**	48618	**0**	48618	**0**	48618	**0**	48618	**0**
h_100_10	1000	0.02	51283	**0**	51283	**0**	51283	**0**	51283	**0**	51283	**0**
h_100_11	1000	0.022	5412	**0**	5412	**0**	5412	**0**	5412	**0**	5412	**0**
h_100_12	1000	0.024	57144	**0**	57144	**0**	57144	**0**	57144	**0**	57144	**0**
h_100_13	1000	0.026	59435	**0**	59435	**0**	59435	**0**	59435	**0**	59435	**0**
h_100_14	1000	0.028	62452	**0**	62452	**0**	62452	**0**	62452	**0**	62452	**0**
h_100_15	1000	0.03	64566	**0**	64566	**0**	64566	**0**	64566	**0**	64566	**0**
h_100_16	1000	0.032	66633	**0**	66633	**0**	66633	**0**	66633	**0**	66633	**0**
h_100_17	1000	0.034	6843	**0**	6843	**0**	6843	**0**	6843	**0**	6843	**0**
h_100_18	1000	0.036	70562	**0**	70561	**0**	70559	**0**	70561	**0**	70562	**0**
h_100_19	1000	0.038	73254	**0**	73251	**0**	73252	**0**	73253	**0**	73254	**0**
h_100_20	1000	0.04	75594	**0**	75594	**0**	75594	**0**	75594	**0**	75594	**0**
h_300_15	3000	0.0033	1396062	< 0.0001	1399153	**0**	1399715	**0**	1399285	**0**	1399131	< 0.0001
h_300_18	3000	0.004	1474857	0.0022	1476893	**0**	147755	**0**	1476893	**0**	147733	**0**
h_300_21	3000	0.0047	157408	**0**	1575515	**0**	157571	**0**	1575123	**0**	1575341	**0**
h_300_24	3000	0.0053	164367	0.0003	1645657	**0**	1646428	**0**	1644986	**0**	1646434	**0**
h_300_27	3000	0.006	1715246	**0**	1717893	**0**	1717294	**0**	1716458	**0**	1717281	**0**
h_300_30	3000	0.0067	1792224	0.0008	1793538	**0**	1794121	**0**	1794246	**0**	1793702	0.0005
h_300_33	3000	0.0073	1885578	**0**	1886194	**0**	1886085	**0**	1886194	**0**	1886085	**0**
h_300_36	3000	0.008	1897373	0.0011	1898578	0.0004	1898628	0.0004	1897843	0.0011	1898949	**0**
h_300_39	3000	0.0087	1973518	0.0005	1972029	**0**	1971486	0.0005	1975793	**0**	1972916	**0**
h_300_42	3000	0.0093	2027509	**0**	2029294	**0**	2028951	**0**	2029609	**0**	2029355	**0**
h_300_45	3000	0.01	2095656	**0**	209603	< 0.0001	2095351	< 0.0001	2097465	< 0.0001	2096821	**0**
h_300_48	3000	0.0107	2150834	**0**	2154956	**0**	2155157	**0**	215582	**0**	215604	**0**
h_300_51	3000	0.0113	2222662	**0**	222172	0.0012	2224548	**0**	2222772	**0**	222503	**0**
h_300_54	3000	0.012	2272972	**0**	2272274	**0**	2272344	**0**	2272744	**0**	227248	**0**
h_300_57	3000	0.0127	2331948	**0**	2332508	**0**	2332534	**0**	233249	**0**	2332761	**0**
h_300_60	3000	0.0133	2375113	**0**	2375059	**0**	23755	**0**	2375663	**0**	237583	**0**
p3000_1	3000	0.1002	6502277	**0**	6502274	**0**	6502265	**0**	6502265	**0**	6502148	< 0.0001
p3000_2	3000	0.2997	18272557	**0**	1827255	**0**	18272524	**0**	18272536	**0**	18272504	**0**
p3000_3	3000	0.4999	29867078	**0**	29867097	**0**	29867111	**0**	29867076	**0**	29867079	**0**
p3000_4	3000	0.8001	4691499	**0**	46914974	**0**	46914987	**0**	4691499	**0**	46914917	< 0.0001
p3000_5	3000	1	58095437	**0**	5809539	**0**	58095408	**0**	58095439	**0**	58095298	**0**
p5000_1	5000	0.0998	17508645	< 0.0001	17508567	< 0.0001	17508431	**0**	17508803	< 0.0001	17508265	< 0.0001
p5000_2	5000	0.2998	50101332	< 0.0001	50101659	**0**	50101915	< 0.0001	50101851	< 0.0001	50101627	< 0.0001
p5000_3	5000	0.5	82036527	< 0.0001	82036746	**0**	82036534	< 0.0001	82036392	< 0.0001	82036918	< 0.0001
p5000_4	5000	0.7999	12941207	< 0.0001	129412658	< 0.0001	12941244	**0**	129411934	< 0.0001	129412097	< 0.0001
p5000_5	5000	1	160597581	< 0.0001	160597461	< 0.0001	160597554	< 0.0001	160597718	**0**	160597232	< 0.0001
#Best mean (Avg. Gap)			**90** (0.5879)		53 (0.2129)		56 (0.3416)		58 (0.1866)		59 (**0.1747**)	

Tabela 6.5: Comparação dos Resultados para o GMMDP

Instance	N	CPU			GPU		
		Avg	**Best**	**Gap**	**Avg**	**Best**	**Gap**
Fashion_MNIST_norm_10NN.mtx	10000	0.248771	0.249919	0.009107	**0.251549**	0.252216	0.000000
astro-ph.mtx	16706	2.229207	2.257277	0.854062	**15.184047**	15.467407	0.000000
cond-mat.mtx	16726	1.611642	1.629307	0.741500	**6.046782**	6.302933	0.000000
har_10NN.mtx	10299	0.231308	0.231537	0.001428	**0.231506**	0.231868	0.000000
hep-th.mtx	8361	2.594509	2.772977	0.800460	**13.896829**	13.896829	0.000000
indianpines_10NN.mtx	9144	0.788410	0.788410	0.000000	0.788410	0.788410	0.000000
kmnist_norm_10NN.mtx	10000	0.207855	0.208784	0.050641	**0.219779**	0.219921	0.000000
optdigits_10NN.mtx	5620	0.295514	0.300092	0.000000	**0.296694**	0.299557	0.001783
usps_norm_5NN.mtx	11000	0.109927	0.111519	0.027377	**0.113783**	0.114658	0.000000
worms20_10NN.mtx	20055	1.156589	1.162766	0.001073	**1.161634**	1.164015	0.000000
MDPI1_5000.txt	5000	251.722668	251.909659	0.000000	**251.736739**	251.904952	0.000019
MDPI2_5000.txt	5000	250.169191	250.323047	0.000126	**250.271551**	250.354546	0.000000
MDPI3_5000.txt	5000	252.402031	252.601401	0.001355	**252.625266**	252.944175	0.000000
MDPI4_5000.txt	5000	254.933852	255.373387	0.000000	**254.979971**	255.311913	0.000241
MDPI5_5000.txt	5000	252.661091	252.871031	0.000149	**252.747538**	252.908830	0.000000
MDPI6_5000.txt	5000	255.973707	256.169373	0.000000	**255.975529**	256.092670	0.000299
MDPI7_5000.txt	5000	254.557263	254.884334	0.000000	**254.624188**	254.869103	0.000060
MDPI8_5000.txt	5000	**253.929619**	254.291458	0.000000	253.896880	254.105919	0.000730
MDPI9_5000.txt	5000	251.468915	251.735760	0.000153	**251.671694**	251.774347	0.000000
MDPI10_5000.txt	5000	255.726418	255.852833	0.000000	**255.727485**	255.788342	0.000252
MDPII1_5000.txt	5000	337.531608	337.710249	0.000367	**337.606120**	337.834283	0.000000
MDPII2_5000.txt	5000	335.665258	335.953202	0.000381	**335.885310**	336.081231	0.000000
MDPII3_5000.txt	5000	335.816156	335.989345	0.001221	**336.175503**	336.400036	0.000000
MDPII4_5000.txt	5000	335.942613	336.147809	0.000960	**336.078257**	336.470810	0.000000
MDPII5_5000.txt	5000	337.437989	337.807299	0.000000	**337.505803**	337.700116	0.000317
MDPII6_5000.txt	5000	334.788207	335.578401	0.000000	**335.061814**	335.428631	0.000446
MDPII7_5000.txt	5000	**335.543163**	336.022379	0.000000	335.520558	335.779778	0.000722
MDPII8_5000.txt	5000	339.670369	340.212181	0.000000	**339.701937**	340.002843	0.000615
MDPII9_5000.txt	5000	334.737162	334.993086	0.000606	**335.022698**	335.196143	0.000000
MDPII10_5000.txt	5000	335.920066	336.256928	0.000000	**335.966675**	336.243000	0.000041
I_5000_1.txt	5000	**101.889848**	101.984641	0.000029	101.886195	101.987575	0.000000
I_5000_2.txt	5000	98.541087	98.628047	0.000000	**98.557482**	98.610616	0.000177
I_5000_3.txt	5000	98.186993	98.250208	0.000000	**98.207864**	98.250208	0.000000
I_5000_4.txt	5000	99.335944	99.369885	0.000000	**99.336097**	99.369885	0.000000
I_5000_5.txt	5000	**102.026092**	102.038177	0.000000	102.022155	102.038177	0.000000
I_5000_6.txt	5000	**100.481097**	100.519335	0.000000	100.443569	100.509535	0.000097
I_5000_7.txt	5000	102.106976	102.158395	0.000133	**102.130864**	102.171978	0.000000
I_5000_8.txt	5000	99.789219	99.856302	0.000000	**99.849717**	99.856302	0.000000
I_5000_9.txt	5000	**99.060066**	99.115116	0.000084	99.052756	99.123456	0.000000
I_5000_10.txt	5000	**98.514495**	98.529044	0.000132	98.505651	98.542048	0.000000
II_5000_1.txt	5000	120.570647	120.683124	0.000000	**120.570988**	120.683124	0.000000
II_5000_2.txt	5000	115.337788	115.429723	0.000000	**115.347840**	115.429723	0.000000
II_5000_3.txt	5000	**117.278555**	117.533543	0.000000	117.203785	117.388792	0.001232
II_5000_4.txt	5000	**117.244169**	117.268860	0.000000	117.244119	117.268149	0.000006
II_5000_5.txt	5000	120.824115	120.904732	0.000000	**120.858005**	120.904732	0.000000
II_5000_6.txt	5000	119.270447	119.370888	0.000000	**119.290087**	119.340032	0.000258
II_5000_7.txt	5000	121.298894	121.426844	0.000000	**121.309094**	121.426844	0.000000
II_5000_8.txt	5000	**117.243748**	117.279096	0.000257	117.236455	117.309301	0.000000
II_5000_9.txt	5000	**116.809140**	116.861181	0.000000	116.807937	116.842093	0.000163
II_5000_10.txt	5000	117.112675	117.128161	0.000000	**117.117136**	117.128161	0.000000
III_5000_1.txt	5000	35.841968	35.868232	0.000404	**35.854094**	35.882716	0.000000
III_5000_2.txt	5000	35.606623	35.653123	0.000000	**35.614803**	35.631536	0.000605
III_5000_3.txt	5000	35.749780	35.802025	0.000000	**35.757069**	35.793564	0.000236
III_5000_4.txt	5000	35.913777	35.964861	0.000000	**35.915390**	35.952205	0.000352
III_5000_5.txt	5000	36.158072	36.191031	0.000000	**36.161797**	36.183455	0.000209
III_5000_6.txt	5000	**36.195951**	36.215220	0.000699	36.190697	36.240535	0.000000
III_5000_7.txt	5000	35.994708	36.036137	0.001166	**35.995316**	36.078219	0.000000
III_5000_8.txt	5000	36.090443	36.128157	0.000713	**36.110195**	36.153933	0.000000
III_5000_9.txt	5000	35.717251	35.739852	0.000112	**35.724014**	35.743871	0.000000
III_5000_10.txt	5000	**35.948288**	35.976525	0.000000	35.945579	35.967363	0.000255
IV_5000_1.txt	5000	357.703671	358.037249	0.000164	**357.847517**	358.095900	0.000000
IV_5000_2.txt	5000	358.192500	358.600933	0.000136	**358.369766**	358.649561	0.000000
IV_5000_3.txt	5000	353.971411	354.480057	0.000000	**354.113585**	354.309234	0.000482
IV_5000_4.txt	5000	353.458035	353.786816	0.001147	**353.797316**	354.193113	0.000000
IV_5000_5.txt	5000	363.106914	363.535565	0.000000	**363.321859**	363.453787	0.000225
IV_5000_6.txt	5000	355.830089	356.253521	0.000455	**356.110059**	356.415771	0.000000
IV_5000_7.txt	5000	359.460724	359.661934	0.000978	**359.739784**	360.013996	0.000000
IV_5000_8.txt	5000	361.697997	362.026555	0.000060	**361.748107**	362.048110	0.000000
IV_5000_9.txt	5000	359.051018	359.382632	0.000000	**359.085222**	359.375848	0.000019
IV_5000_10.txt	5000	355.752907	356.120749	0.000818	**355.982227**	356.412371	0.000000

Tabela 6.6: Comparação dos Resultados para o GMMDP

Instance	N	CPU			GPU		
		Avg	**Best**	**Gap**	**Avg**	**Best**	**Gap**
I_3000_1.txt	3000	**76.176331**	76.187020	0.000000	76.166421	76.187020	0.000000
I_3000_2.txt	3000	**77.455805**	77.456661	0.000000	77.450185	77.456661	0.000000
I_3000_3.txt	3000	**74.873863**	74.875507	0.000000	74.871494	74.875507	0.000000
I_3000_4.txt	3000	**77.117067**	77.129478	0.000000	77.113162	77.125362	0.000053
I_3000_5.txt	3000	**75.919801**	75.950514	0.000000	75.895246	75.950514	0.000000
I_3000_6.txt	3000	**78.418876**	78.418876	0.000000	78.418723	78.418876	0.000000
I_3000_7.txt	3000	77.453989	77.454874	0.000000	**77.454284**	77.454874	0.000000
I_3000_8.txt	3000	74.126382	74.180346	0.000000	**74.151199**	74.180346	0.000000
I_3000_9.txt	3000	**75.744976**	75.766182	0.000000	75.726911	75.766182	0.000000
I_3000_10.txt	3000	**77.484601**	77.497833	0.000000	77.472014	77.497833	0.000000
II_3000_1.txt	3000	89.699289	89.699289	0.000000	89.699289	89.699289	0.000000
II_3000_2.txt	3000	90.632740	90.632740	0.000000	90.632740	90.632740	0.000000
II_3000_3.txt	3000	89.212552	89.212552	0.000000	89.212552	89.212552	0.000000
II_3000_4.txt	3000	**91.485673**	91.507781	0.000000	91.467476	91.507781	0.000000
II_3000_5.txt	3000	**88.643869**	88.643869	0.000000	88.641897	88.643869	0.000000
II_3000_6.txt	3000	**92.159940**	92.159940	0.000000	92.154048	92.159940	0.000000
II_3000_7.txt	3000	**91.619903**	91.633552	0.000000	91.564826	91.630850	0.000029
II_3000_8.txt	3000	**87.949340**	87.949340	0.000000	87.947808	87.949340	0.000000
II_3000_9.txt	3000	**89.296846**	89.316453	0.000000	89.223651	89.309886	0.000074
II_3000_10.txt	3000	91.794531	91.794531	0.000000	91.794531	91.794531	0.000000
III_3000_1.txt	3000	**27.517257**	27.520511	0.000000	27.516299	27.520511	0.000000
III_3000_2.txt	3000	28.369090	28.373383	0.000000	**28.370332**	28.373383	0.000000
III_3000_3.txt	3000	27.810449	27.810449	0.000000	27.810449	27.810449	0.000000
III_3000_4.txt	3000	27.891417	27.903593	0.000000	**27.892053**	27.903593	0.000000
III_3000_5.txt	3000	**27.533049**	27.545323	0.000000	27.530620	27.545323	0.000000
III_3000_6.txt	3000	**27.633520**	27.633756	0.000000	27.633409	27.633756	0.000000
III_3000_7.txt	3000	**27.659199**	27.667870	0.000000	27.653925	27.667595	0.000010
III_3000_8.txt	3000	**27.728646**	27.733535	0.000000	27.726627	27.733535	0.000000
III_3000_9.txt	3000	**27.242078**	27.242377	0.000000	27.237028	27.242377	0.000000
III_3000_10.txt	3000	28.732278	28.732278	0.000000	28.732278	28.732278	0.000000
IV_3000_1.txt	3000	**273.835334**	273.909416	0.000000	273.825019	273.909416	0.000000
IV_3000_2.txt	3000	**278.728857**	278.818807	0.000000	278.654968	278.818807	0.000000
IV_3000_3.txt	3000	**275.870389**	275.962877	0.000000	275.798256	275.962877	0.000000
IV_3000_4.txt	3000	280.996033	281.126915	0.000018	**281.000800**	281.131868	0.000000
IV_3000_5.txt	3000	274.495819	274.560375	0.000000	**274.496558**	274.560375	0.000000
IV_3000_6.txt	3000	**276.943979**	276.972256	0.000000	276.890122	276.972256	0.000000
IV_3000_7.txt	3000	**276.394916**	276.525935	0.000016	276.310783	276.530359	0.000000
IV_3000_8.txt	3000	277.118803	277.151163	0.000000	**277.128601**	277.151163	0.000000
IV_3000_9.txt	3000	**271.809956**	271.906519	0.000000	271.798638	271.905355	0.000004
IV_3000_10.txt	3000	**280.850009**	280.861467	0.000000	280.790196	280.861467	0.000000
#Best mean (Avg. Gap)			**40** (2.4984)		63 (0.0100)		

Capítulo 7

Conclusão e Trabalhos Futuros

7.1 Conclusão

Este trabalho propôs dois algoritmos baseados em GPU para resolver instâncias massivas do problema de diversidade máxima e do problema de dispersão máxima média ponderada. Para explorar o poder da arquitetura de GPU, novas estruturas de dados e estratégias de busca local foram desenvolvidas. Essas abordagens foram testadas em uma variedade de instâncias bem conhecidas da literatura, sendo comparadas com métodos sequenciais de ponta, a fim de avaliar a performance e a eficácia das soluções propostas.

Os resultados experimentais demonstraram que, ao empregar as soluções em GPU, foi possível obter uma aceleração muito superior em relação aos métodos tradicionais. Além disso, em instâncias grandes, as propostas não apenas reduziram o tempo de execução de maneira expressiva, mas também alcançaram soluções de melhor qualidade. Essa aceleração foi possibilitada pela paralelização das tarefas computacionais e pelo uso de estratégias que exploram a arquitetura de memória das GPUs, como a utilização das estruturas de dados CSR e UTM, que ajudam a reduzir o uso de memória e a melhorar a localidade de acesso aos dados.

A análise dos resultados apresentados nas Subseções 6.1 e 6.2 confirmou a eficácia das abordagens propostas em GPU, especialmente a versão GPU-TS-f3 para o MDP e a versão em GPU para o GMaxMeanDP. Ambas as abordagens se mostraram muito eficazes em lidar com instâncias de grande porte, onde a complexidade do problema e o aumento da busca local tornavam os métodos tradicionais ineficazes. A adaptação das estruturas de dados foi uma das principais responsáveis por esse sucesso, uma vez que permitiu otimizar o uso da memória e melhorar a velocidade de acesso aos dados durante a execução. O uso de filtros, que intensi-

ficam a busca em regiões promissoras, também foi uma estratégia eficaz que permitiu obter melhores resultados nas instâncias de maior tamanho, superando as limitações dos métodos sequenciais.

No entanto, é importante observar que as abordagens baseadas em GPU, embora extremamente eficazes em instâncias grandes, não são sempre a melhor solução para instâncias menores. Isso ocorre porque a transferência de dados entre a memória da CPU e da GPU, embora otimizada, ainda representa uma sobrecarga para instâncias de pequeno porte, onde o tempo de execução pode ser menor do que o tempo necessário para estabelecer a comunicação entre os dispositivos. Em casos onde as instâncias são pequenas, os métodos sequenciais podem ser mais eficientes, pois não há o overhead associado à utilização da GPU. Essa é uma das limitações dos algoritmos baseados em GPU, que devem ser usados principalmente em cenários em que as instâncias são grandes o suficiente para justificar a utilização de recursos paralelos de forma eficiente.

Outro ponto importante a ser destacado é a complexidade envolvida na implementação de algoritmos baseados em GPU. Embora as GPUs ofereçam um grande potencial de aceleração, elas exigem conhecimento especializado em programação paralela e no uso de bibliotecas específicas como o CUDA. Além disso, é necessário hardware específico para rodar esses algoritmos, o que limita o acesso a essas soluções em alguns ambientes. A implementação de tais algoritmos também precisa de cuidados com a gestão de memória e a sincronização entre as diferentes threads, o que adiciona uma camada adicional de complexidade em relação aos métodos sequenciais tradicionais.

Portanto, os resultados experimentais confirmam a eficácia das abordagens baseadas em GPU para otimização de grandes instâncias do MDP e GMaxMeanDP, proporcionando não apenas aceleração, mas também melhores soluções. Embora existam limitações no uso de GPU para instâncias menores, a escalabilidade das soluções é uma vantagem clara para problemas de maior porte. No entanto, a implementação de algoritmos em GPU apresenta desafios significativos em termos de complexidade e requisitos de hardware, o que deve ser considerado ao optar por essa abordagem. A utilização de GPUs é, portanto, uma solução altamente eficaz, mas que precisa de um balanceamento cuidadoso entre as características do problema, a disponibilidade de recursos e a complexidade da implementação.

7.2 Trabalhos Futuros

Nos trabalhos futuros, pode-se explorar a utilidade de integrar as abordagens propostas em outras meta-heurísticas híbridas. Além disso, é estudado a adaptação dessas abordagens para outros problemas de diversidade e de dispersão. Por fim, é investigada uma estratégia de redução baseada em GPU, com o objetivo de reduzir o tamanho de instâncias massivas.

7.3 Publicações Obtidas

O artigo intitulado "GPU Tabu Search: A Study on Using GPU to Resolve Massive Instances of the Maximum Diversity Problem" apresenta uma abordagem inovadora baseada em busca tabu implementada em GPU para resolver instâncias massivas do Problema da Diversidade Máxima. Publicado no Journal of Parallel and Distributed Computing, este trabalho aborda as limitações das heurísticas tradicionais ao lidar com problemas de grande escala. A solução proposta por Nogueira et al. (2024) emprega estruturas de dados otimizadas para instâncias esparsas e utiliza o paralelismo da GPU para acelerar significativamente o processo de busca local. O algoritmo foi testado em instâncias de diferentes escalas, incluindo casos massivos com até 45.000 vértices, obtendo um aumento de até 50 vezes na velocidade em comparação com métodos do estado da arte, comprovando sua eficiência em cenários de grande escala (NOGUEIRA et al., 2024).

Além disso, um novo artigo que explorará o GMaxMeanDP está em processo de desenvolvimento. Este problema, que se distingue do MDP por sua flexibilidade no tamanho do subconjunto de vértices e pela ponderação dos elementos selecionados, está sendo abordado com uma estratégia similar, aproveitando o poder computacional das GPUs. Acreditamos que os avanços metodológicos apresentados nesse novo estudo terão um impacto significativo na resolução de problemas combinatórios em cenários reais de larga escala.

Referências bibliográficas

AARDAL, K. et al. (Ed.). **Handbooks in OR & MS, Vol. 12**. [S.l.]: Elsevier B.V., 2005. cap. 1. On the History of Combinatorial Optimization (Till 1960).

ANDERSEN, R.; CHELLAPILLA, K. Finding dense subgraphs with size bounds. In: INTERNATIONAL Workshop on Algorithms and Models for the Web-Graph. [S.l.]: Springer, 2009. P. 25–37.

BENLIC, U.; HAO, J. K. Memetic search for the quadratic assignment problem. **Expert Systems with Applications**, v. 42, p. 584–595, 2015.

BIRKENMEIER, G. F.; HEATHERLY, H. E.; KIM, J. Y.; PARK, J. K. Triangular Matrix Representations. **Journal of Algebra**, v. 230, n. 2, p. 558–595, 2000. DOI: 10.1006/jabr.2000.8328.

BLUM, C.; ROLI, A. **Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison**. [S.l.]: Springer, 2003.

BORGWARDT, S.; SCHMIEDL, F. Threshold-based preprocessing for approximating the weighted dense k-subgraph problem. **European Journal of Operational Research**, v. 234, n. 3, p. 631–640, 2014.

BRIMBERG, J.; MLADENOVIC, N.; TODOSIJEVIC, R.; UROŠEVIC, D. Less is more: Solving the max-mean diversity problem with variable neighborhood search. **Information Sciences**, v. 382, p. 179–200, 2017.

BRIMBERG, J.; MLADENOVIC, N.; UROŠEVIĆ, D.; NGAI, E. Variable neighborhood search for the heaviest k-subgraph. **Computers & Operations Research**, v. 36, n. 11, p. 2885–2891, 2009.

CARRASCO, R. et al. Tabu search for the max-mean dispersion problem. **Knowledge Based System**, v. 85, p. 256–264, 2015.

COOK, S. A. The Complexity of Theorem-Proving Procedures. In: PROCEEDINGS of the Third Annual ACM Symposium on Theory of Computing. [S.l.]: ACM, 1971. P. 151–158. DOI: 10.1145/800157.805047.

CORMEN, T. H. et al. **Algoritmos: Teoria e Prática**. Tradução: Vandenberg D. de Souza. 2ª edição. [S.l.]: Elsevier, 2002.

DELLA CROCE, F.; GARRAFFA, M.; SALASSA, F. A hybrid three-phase approach for the max-mean dispersion problem. **Computers & Operations Research**, v. 71, p. 16–22, 2016.

DUARTE, A.; MARTÍ, R. Tabu search and grasp for the maximum diversity problem. **European Journal of Operational Research**, v. 178, n. 1, p. 71–84, 2007.

DUMITRESCU, I.; STUETZLE, T. Combinations of local search and exact algorithms. In: APPLICATIONS of Evolutionary Computation. [S.l.]: Springer, 2003. v. 2611, p. 211–223.

FEIGE, U.; SELTSER, M. et al. On the densest k-subgraph problem, 1997.

FULBER-GARCIA, V.; AIBIN, M. Heurística vs. Meta-Heurística vs. Algoritmos Probabilísticos, 2023. Disponível em: <https://www.baeldung.com/cs/heuristics-vs-meta-heuristics-vs-probabilistic-algorithms>.

GALLEGO, M.; DUARTE, A.; LAGUNA, M.; MARTÍ, R. Hybrid heuristics for the maximum diversity problem. **Computational Optimization and Applications**, v. 44, p. 411–426, 2009.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. [S.l.]: W. H. Freeman e Company, 1979.

GARRAFFA, M.; DELLACROCE, F.; SALASSA, F. An exact semidefinite programming approach for the max-mean dispersion problem. **Journal of Combinatorial Optimization**, v. 34, p. 1–23, 2017.

GHOSH, M. et al. Recursive memetic algorithm for gene selection in microarray data. **Expert Systems with Applications**, v. 116, p. 172–185, 2019.

GLOVER, F.; LAGUNA, M. **Tabu search**. [S.l.]: Kluwer Academic Publishers, 1997.

GLOVER, F.; LAGUNA, M.; MARTI, R. Principles of Tabu Search. In: APPROXIMATION Algorithms and Metaheuristics. [S.l.: s.n.], 2007. v. 23. P. 1–12.

HAAS, L.; NAMBIAR, A. **How do you choose between heuristic and exact algorithms?** [S.l.: s.n.], 2024. Disponível em: https://www.linkedin.com/advice/0/how-do-you-choose-between-heuristic-exact-algorithms-jydge, acessado em 27 nov. 2024.

HWU, W.-M.; KIRK, D.; HAJJ, I. **Programming Massively Parallel Processors: A Hands-on Approach**. Fourth. [S.l.]: Morgan Kaufmann, 2022. Disponível em: <https://www.sciencedirect.com/science/book/9780323912310>.

ISMKHAN, H. Effective three-phase evolutionary algorithm to handle the large-scale colorful traveling salesman problem. **Expert Systems with Applications**, v. 67, p. 148–162, 2017.

KARP, R. Reducibility Among Combinatorial Problems. In: COMPLEXITY of Computer Computations. [S.l.: s.n.], 1972. v. 40, p. 85–103. ISBN 978-3-540-68274-5. DOI: 10.1007/978-3-540-68279-0_8.

KERCHOVE, C.; DOOREN, P. V. The page trust algorithm: how to rank web pages when negative links are allowed?, p. 346–352, 2008.

KINCAID, R. Good solutions to discrete noxious location problems via metaheuristics. **Annals of Operations Research**, v. 40, n. 1, p. 265–281, 1992.

KOVALEV, S.; CHALAMON, I.; PETANI, F. J. Maximizing single attribute diversity in group selection. **Annals of Operations Research**, v. 320, n. 1, p. 535–540, 2023.

LAI, X. J.; HAO, J. K. A tabu search based memetic search algorithm for the max-mean dispersion problem. **Computers & Operations Research**, v. 72, p. 118–127, 2016.

LAI, X.; HAO, J.-K.; GLOVER, F. A study of two evolutionary/tabu search approaches for the generalized max-mean dispersion problem. **Expert Systems with Applications**, v. 139, p. 112856, 2020. ISSN 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2019.112856. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417419305585>.

LETSIOS, M. et al. Finding heaviest k-subgraphs and events in social media. In: 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW). [S.l.]: IEEE, 2016. P. 113–120.

LIU, X. et al. A two-phase tabu search based evolutionary algorithm for the maximum diversity problem. **Discrete Optimization**, v. 44, p. 100613, 2022.

MARTÍ, R.; GALLEGO, M.; DUARTE, A.; PARDO, E. Heuristics and metaheuristics for the maximum diversity problem. **Journal of Heuristics**, v. 19, p. 591–615, 2013.

MARTÍ, R.; SANDOYA, F. GRASP and path relinking for the equitable dispersion problem. **Computers& Operations Research**, v. 40, n. 12, p. 3091–3099, 2013.

MARTÍ, R.; MARTÍNEZ-GAVARA, A.; PÉREZ-PELÓ, S.; SÁNCHEZ-ORO, J. A review on discrete diversity and dispersion maximization from an OR perspective. **European Journal of Operational Research**, v. 299, n. 3, p. 795–813, 2022. DOI: 10.1016/j.ejor.2021.07.044.

MIYAZAWA, F. K. **Otimização Combinatória**. [S.l.]: Universidade Estadual de Campinas, 2024. https://www.ic.unicamp.br/~fkm/problems/combopt.html, acessado em 25 nov. 2024.

MORRA, L.; COCCIA, N.; CERQUITELLI, T. Optimization of computer aided detection systems: An evolutionary approach. **Expert Systems with Applications**, v. 100, p. 145–156, 2018.

NOGUEIRA, B.; ROSENDO, W.; TAVARES, E.; ANDRADE, E. GPU tabu search: a study on using GPU to solve massive instances of the maximum diversity problem. **Journal of Parallel and Distributed Computing**, p. 105012, 2024. ISSN 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2024.105012. Disponível em: <https://www.sciencedirect.com/science/article/pii/S074373152400176X>.

PAPADIMITRIOU, C.; STEIGLITZ, K. **Combinatorial Optimization: Algorithms and Complexity**. [S.l.: s.n.], 1982. v. 32. ISBN 0-13-152462-3. DOI: 10.1109/TASSP.1984.1164450.

PISINGER, D. Upper bounds and exact algorithms for p-dispersion problems. **Computers & Operations Research**, v. 33, n. 5, p. 1380–1398, 2006.

PROKOPYEV, O. A.; KONG, N.; MARTINEZ-TORRES, D. L. The equitable dispersion problem. **European Journal of Operational Research**, v. 197, n. 1, p. 59–67, 2009.

PROKOPYEV, O. A.; KONG, N.; MARTINEZ-TORRES, D. L. The equitable dispersion problem. **European Journal of Operational Research**, v. 197, n. 1, p. 59–67, 2009. ISSN 0377-2217.

DOI: https://doi.org/10.1016/j.ejor.2008.06.005. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0377221708004712>.

PUCHINGER, J.; RAIDL, G. R. Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification. In: MIRA, J.; ÁLVAREZ, J. R. (Ed.). **Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. P. 41–53. ISBN 978-3-540-31673-2.

SCHRIJVER, A. **Combinatorial Optimization, Polyhedra and Efficiency**. [S.l.]: Springer, 2005.

SHAH, T. Understanding NP Complexity, 2023. https://medium.com/@tanmays309/understanding-np-complexity-8a7cd9957710.

SILVA, J. D. A.; HRUSCHKA, E. R.; GAMA, J. An evolutionary algorithm for clustering data streams with a variable number of clusters. **Expert Systems with Applications**, v. 67, p. 228–238, 2017.

SOBRAPO. **O que é Pesquisa Operacional?** [S.l.: s.n.], 2024. Disponível em: <https://www.sobrapo.org.br/o-que-e-pesquisa-operacional>.

SPIERS, S.; BUI, H.; LOXTON, R. An exact cutting plane method for the euclidean max-sum diversity problem. **European Journal of Operational Research**, 2023.

UNISOMA. **O que é Pesquisa Operacional e Como Colocá-la em Prática na Sua Empresa?** [S.l.: s.n.], 2024. Disponível em: <https://www.unisoma.com.br/o-que-e-pesquisa-operacional-e-como-coloca-la-em-pratica-na-sua-empresa>.

VOITTO. **Pesquisa Operacional: Conceitos e Aplicações**. [S.l.: s.n.], 2020. Disponível em: <https://voitto.com.br/blog/artigo/pesquisa-operacional>.

WILT, N. **The CUDA Handbook: A Comprehensive Guide to GPU Programming**. [S.l.]: Pearson Education, 2013.

WONG, K.; MORADI, M. SegNAS3D: Network Architecture Search with Derivative-Free Global Optimization for 3D Image Segmentation, set. 2019. DOI: 10.48550/arXiv.1909.05962.

YANG, B.; CHEUNG, W.; LIU, J. Community mining from signed social networks. **IEEE Transactions on Knowledge& Data Engineering**, v. 19, n. 10, p. 1333–1348, 2007.

ZHAO, H.; XU, W.; JIANG, R. The memetic algorithm for the optimization of urban transit network. **Expert Systems with Applications**, v. 42, p. 3760–3773, 2015.

ZHAO, J.; HIFI, M.; LATREM, K. Hybrid Particle Swarm Optimization and Skewed Variable Neighborhood Search Techniques for the Generalized Max-Mean Dispersion Problem, out. 2024. DOI: 10.21203/rs.3.rs-5246898/v1.

ZHOU, Y.; HAO, J. K. An iterated local search algorithm for the minimum differential dispersion problem. **Knowledge-Based Systems**, v. 125, p. 26–38, 2017.

ZHOU, Y.; HAO, J.-K.; DUVAL, B. Opposition-based memetic search for the maximum diversity problem. **IEEE Transactions on Evolutionary Computation**, v. 21, n. 5, p. 731–745, 2017.

Apêndice A

Exemplo

A.1 Exemplo