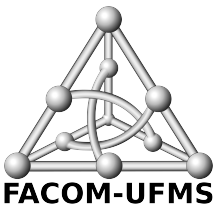
Implementação da meta-heurística de busca em vizinhanças variáveis (VNS) na solução do Problema da Mochila Multidimensional

Klelber Dias Januário

Trabalho de Conclusão de Curso

Orientação
Profa. Dra. Bianca de Almeida Dantas



Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Campo Grande - MS
2023

Aos meus pais Juraci e Élio.
À minha irmã Mirieli e ao meu irmão Edypo.

Eu sou o caminho, e a verdade e a vida. Ninguém vem ao Pai, senão por mim.
(João 14:6)

Agradecimentos

Primeiramente, expresso minha gratidão a Deus por ter aberto as portas e me proporcionado o privilégio de estudar nesta instituição de ensino. Nunca faltou sua guia, e em todas as minhas dificuldades, recorri a Ele, que nunca negou sua ajuda.

À minha mãe Juraci, uma verdadeira guerreira. Tudo que eu precisava, ela estava sempre pronta a ouvir e ajudar. Sempre cuidou de mim e da minha irmã com muito amor e carinho. Com todas as dificuldades, crise, obstáculos, lutas, dores, medos, desafios e julgamentos criou nós. Amor nunca faltou entre nós.

À minha irmã Mirieli, que é um tesouro do céu. Sempre carinhosa e com um coração enorme tem me apoiado em toda as minhas vitórias e derrotas, sempre do meu lado, ajudando em tudo que precisei.

Ao meu pai Élio, um guerreiro e trabalhador. Mesmo com tantas dificuldades, nunca deixou faltar o pão na mesa de casa. Além disso, conselhos e incentivos sempre esteve nele, principalmente, na parte dos estudos.

À minha orientadora Bianca, que desde a disciplina de Algoritmos e Programação II, tem transmitido conhecimento para mim. Além disso, proporcionou uma grande oportunidade de começar um projeto de Iniciação Cientifica, onde, pacientemente, instruiu-me em todo o decorrer do projeto. Por fim, humildemente, aceitou o convite para orientar-me no meu Trabalho de Conclusão de Curso.

A todos os professores da Faculdade de Computação e do Instituto de Matemática que de forma direta ou indireta contribuiu para minha formação. Os meus sinceros agradecimentos a todos vocês.

Sumário

Lista de Tabelas

Lista de Algoritmos

Resumo

A otimização combinatória, um dos principais ramos da computação, aborda desafios complexos, entre os quais se destaca o problema da mochila multidimensional. Apesar de suas diversas aplicações práticas, esse problema é classificado como $\mathcal{NP}$-Difícil, o que implica a ausência, até o presente momento, de um algoritmo polinomial capaz de encontrar uma solução exata para o problema. Diante desse cenário, fomenta a necessidade de desenvolver técnicas que proporcionem soluções eficazes em uma baixa quantidade de tempo, e é nesse contexto que as meta-heurísticas desempenham um papel crucial, evidenciando sua capacidade de atingir ótimos resultados. Este trabalho apresenta a implementação e análise de uma meta-heurística baseada em vizinhança conhecida como Busca de Vizinhança Variável, ou *Variable Neighborhood Search* (VNS) em inglês, aplicada à resolução do problema da mochila multidimensional.

Palavras-chave: *Problema da Mochila Multidimensional*, *Meta-heurísticas*, *Busca em Vizinhança*.

Abstract

Combinatorial optimization, one of the primary branches of computing, addresses complex challenges, among which the multidimensional knapsack problem stands out. Despite its diverse practical applications, this problem is classified as $\mathcal{NP}$-Hard, implying the absence, to date, of a polynomial algorithm capable of finding an exact solution to the problem. Faced with this scenario, there is a need to develop techniques that provide effective solutions in a short amount of time, and it is in this context that metaheuristics play a crucial role, showcasing their ability to achieve optimal results. This work presents the implementation and analysis of a metaheuristic based on a well-known neighborhood search called Variable Neighborhood Search (VNS), applied to solving the multidimensional knapsack problem.

Keywords: *Multidimensional Knapsack Problem*, *Metaheuristics*, *Neighborhood Search*.

CAPÍTULO 1

Introdução

Os problemas de otimização combinatória consistem em encontrar a melhor solução entre um conjunto finito de possibilidades discretas, porém muito grande. A finalidade geral deles é encontrar uma solução que otimize uma determinada função objetivo sujeita a um conjunto de restrições lineares que descrevem o problema. De acordo com (Stutzle, 1998), pode-se definir cada um desses problemas como um par (S, f), onde S representa o conjunto finito de soluções possíveis e $f : S \rightarrow \mathbb{R}$ é uma função objetivo que atribui a cada valor $s \in S$ um valor real $f(s)$. Sendo o objetivo do problema de otimização combinatória encontrar uma solução s^* com o menor valor da função objetivo sujeito às restrições C tal que $f(s^*) \leq f(s^{'}) \; \forall s^{'} \in S$. A solução s^* é dita como globalmente ótima de (S, f). Similarmente, essa definição é estendida para problemas de maximização como abordado em (Foulds, 1981).

O problema da mochila multidimensional encontra-se na categoria Programação Linear Inteira Binária, onde a função objetivo e suas restrições são lineares e os valores para cada incógnita do problema devem ser números inteiros entre $\{0, 1\}$. De acordo com (Wolsey, 1998), esse simples requisito adicionado implica uma maior complexidade computacional para resolver, uma vez que torna desafiador responder à pergunta sobre a existência de uma solução 0-1 viável para o problema.

O problema abordado no trabalho pode ser visto como a generalização do problema da mochila binária. Para entendê-lo, imagine que uma pessoa precisa buscar itens em uma loja e inserir dentro da sua mochila de tal maneira que ao vender esses produtos obtenha o maior valor possível e não exceda o peso máximo que a mochila consegue suportar. Formalmente, tem-se: Dada uma mochila de capacidade W e um conjunto de n itens com peso $p_i \geq 0$ e valor $v_i \geq 0$, ambos inteiros, encontrar um subconjunto tal que (Cáceres e Nishibe, 2005):

$$\begin{aligned} & \max \left\{ \sum_{i=1}^{n} v_i x_i \right\} \\ \text{s.a} \quad & \sum_{i=1}^{n} p_i x_i \leq W, x_i \in \{0,1\} \end{aligned} \tag{1.1}$$

onde $X = \{x_1, x_2, ..., x_n\}$ é o chamado vetor de solução, de maneira que, se o item j está presente na mochila, então $x_j = 1$, caso contrário, $x_j = 0$.

A primeira ideia intuitiva em tentar resolver esse problema seria apresentar todas as soluções viáveis do problema e testar cada uma delas. No entanto, o total de combinações possíveis é 2^n, onde n representa a quantidade de itens disponíveis para levar na solução. Utilizar essa técnica (força bruta) não é uma boa estratégia, uma vez que o tempo de processamento aumenta exponencialmente à medida que n cresce. Por exemplo, considere que o número de itens seja igual a 120 e um computador capaz de realizar 10 quintilhões de operações por segundo. Nesse caso, arredondando os valores do cálculo para múltiplos de 10, levaria mais de 1000 anos para listar todas as alternativas possíveis para o problema. Por esse motivo, surge a necessidade de procurar outras alternativas de resolução do problema e as heurísticas vem surgindo com um grande potencial, pois, de um modo geral, são técnicas capazes de obter boas soluções em um tempo razoável.

Vários problemas de otimização combinatória pertencem à classe $\mathcal{NP}$-difícil (Wolsey, 1998) e, apesar da simplicidade em enunciá-los, muitas das vezes são difíceis de resolver. O problema da mochila binária também está incluído nessa classe, implicando que ainda não é conhecido um algoritmo que resolva esse problema em tempo polinomial. Existem diversas técnicas a fim de encontrar uma solução exata para o problema da mochila 0-1 ou mochila binária. Algumas delas incluem programação dinâmica e *branch & bound*. Para a primeira técnica, a complexidade assintótica do algoritmo é da ordem de $O(nW)$, uma vez que depende da capacidade ou peso (W) da mochila, sendo considerada *pseudopolinomial*. Já a última estratégia, é uma boa técnica para encontrar a solução em pequenos e médios portes de instâncias (Kellerer, Pferschy e Pisinger, 2004). No pior cenário, precisamos calcular toda a árvore, ou seja, todas as combinações.

No momento em que são adicionadas um número m de restrições ao problema da mochila 0-1, tem-se o chamado problema da mochila multidimensional. Da mesma forma que no problema reduzido (mochila binária), métodos como programação dinâmica, *branch & bound* foram aplicados. No entanto, quando as instâncias possuem grande escala, esses métodos apresentam desempenho insatisfatório. E por isso, motivam-se os estudos de novas alternativas para alcançar uma solução, ainda que aproximada. Nesse caso, os algoritmos heurísticos ou em alguns casos os meta-heurísticos geralmente fornecem boas soluções, embora não haja nenhuma garantia sobre o quão bom serão os resultados obtidos.

O objetivo deste trabalho é mostrar os resultados obtidos da implementação da meta-heurísica VNS (*Variable Neighborhood Search*) e da junção de duas meta-heurísticas VNS e *Simulated Annealing* formando uma meta-heurística híbrida. Para tal, será implementada na forma sequencial e testadas com os mesmos conjuntos de instâncias usados em trabalhos de referência (Dantas, 2016).

A estrutura subsequente do texto é a seguinte. No Capítulo 2 são fornecidas a definição do problema da mochila multidimensional, as meta-heurísticas e uma descrição detalhada do ambiente de testes, acompanhada do processo de validação dos resultados alcançados. As meta-heurísticas estudadas são o tema do Capítulo 3, onde são apresentadas as bases teóricas de cada uma delas. As várias implementações e modificações apresentadas até a versão final estão presentes no Capítulo 4. Outrossim, os resultados

obtidos nos conjuntos de testes ficam detalhadas no Capítulo 5. Finalmente, o Capítulo 6 aborda as conclusões do estudo e sugere direções para trabalhos futuros.

CAPÍTULO 2

Contextualização

O problema da mochila destaca-se como um dos problemas mais clássicos da área de otimização combinatória. Embora sua formulação seja simples, esse problema pertence à classe de problemas $\mathcal{NP}$-difíceis. Sua relevância é evidenciada pelo fato de ser um subproblema em muitos outros problemas mais complexos. Por exemplo, em logística, determinar quais itens devem ser adicionados em um caminhão com o objetivo de maximizar o valor ou lucro, entre outros. Por essa importância, motiva a atenção no seu estudo e suas variantes. Existem diversas abordagens do problema na literatura, distinguindo-se, em geral, pelo número de restrições aplicadas à solução, pela possível partição dos itens ou pela ausência de limites, dentre elas:

- Mochila 0-1 (mochila binária): consiste em, dada uma mochila de capacidade W e um conjunto de n itens cujos pesos p_i e valores v_i são fornecidos, determinar quais itens serão adicionados na mochila de maneira que maximize o valor obtido na soma dos itens levados;
- Mochila Fracionária: uma variação da mochila binária, onde os itens não precisam ser levados na sua totalidade. Eles podem ser particionados e colocados frações de cada item na mochila;
- Mochila Inteira: extensão da mochila binária, considerando agora a existência de quantidade infinita do mesmo item para ser adicionada na mochila;
- Mochila Multidimensional: extensão da mochila binária, considerando agora um conjunto de m restrições;
- Mochila Multiobjetivo: extensão da mochila multidimensional que considera mais de um objetivo a ser maximizado. Dessa forma, cada item tem mais de um valor associado, um por objetivo.

O foco principal deste trabalho consiste no problema da mochila multidimensional. Para seu melhor entendimento, é necessário, primeiramente, compreender a versão binária. Em (Chu e Beasley, 1998) foi proposta a formulação como na Equação 2.1:

$$\begin{aligned} &\max \left\{ \sum_{j=1}^{n} v_j x_j \right\} \\ \text{tal que: } &\sum_{j=1}^{n} r_{ij} x_j \leq b_i, i = 1, \ldots, m, \\ &x_j \in \{0, 1\}, j = 1, \ldots, n. \end{aligned} \tag{2.1}$$

onde n é o número de itens disponíveis para adicionar à mochila, m representa a quantidade de restrições, $V = \{v_1, v_2, \ldots, v_n\}$ é o vetor de valores dos itens, $B = \{b_1, b_2, \ldots, b_m\}$ é o vetor que contém a capacidade máxima para cada restrição e $X = \{x_1, x_2, \ldots, x_n\}$ é o vetor de solução, no qual cada elemento pode assumir os valores 1 ou 0, equivalente, respectivamente, à presença ou ausência do item na solução. A matriz $R = \{r_{11}, \ldots, r_{1n}, \ldots, r_{m1}, \ldots, r_{mn}\}$, por sua vez, representa o quanto cada item consome de cada recurso (restrição). É dito que um vetor X é uma solução do problema se satisfazer todas as restrições dadas.

O problema da mochila multidimensional encontra uma gama variedade de aplicações em diversas áreas, incluindo, criptografia, problemas de logística e tomada de decisão, entre outras. A literatura também destaca atividades práticas relacionadas ao nosso problema de estudo, como o corte de materiais, a alocação de processadores e banco de dados em sistemas distribuídos de grande escala, o orçamento de capital e a alocação de recursos (Lemos e Longo, 2021). Devido a seu amplo uso, encontrar algoritmos que encontram a solução ótima ou aproximada em um curto prazo de tempo é fundamental.

Uma técnica viável para alcançar uma solução ótima para o problema consiste em aplicar programação dinâmica utilizando, neste momento, as m restrições fornecidas. Claramente, percebe-se que a complexidade computacional dessa alternativa é mais elevada do que no caso anterior, uma vez que será necessário criar uma matriz com $m + 1$ dimensões, atribuindo uma para cada restrição e outra para os itens possíveis. Essa abordagem diverge das duas dimensões necessárias no contexto do problema da mochila binária. Por isso, outras técnicas precisam ser estudadas e testadas para a resolução desse problema.

2.1 Meta-heurísticas

Formalmente, uma meta-heurística é definida como um processo de geração iterativo que orienta uma heurística subordinada combinando conceitos diferentes de formas inteligentes para explorar o espaço de busca. Normalmente, poucas alterações são necessárias nelas para utilizá-las em diferentes problemas (Blum e Roli, 2003). Infelizmente, essa técnica não permite dar garantia nenhuma sobre o resultado produzido. Porém, um grande número de trabalhos dedicados à aplicação de meta-heurísticas para resolver problemas de otimização combinatória têm mostrado a eficácia de tais métodos, motivando o estudo de suas diferentes variantes na solução de problemas para os quais a obtenção de uma solução exata é difícil, em especial para a classe de problemas $\mathcal{NP}$-difíceis, como o problema da mochila multidimensional, objeto das análises e implementações deste trabalho.

Em sua tese de doutorado, (Dantas, 2016) avalia a implementação das seguintes meta-heurísticas para resolver o problema da mochila multidimensional: Algoritmos Genéticos; Redes Neurais Aumentadas; GRASP; Simulated Annealing; e a junção das Redes Neurais Aumentadas e GRASP. Em sua comparação, foi utilizada a versão sequencial e a paralela, o que permitiu alcançar melhores resultados e conseguir diminuir o tempo de execução. Bons resultados foram obtidos com algumas estratégias adotadas, entretanto, para algumas meta-heurísticas, a porcentagem da diferença entre o valor obtido e o valor ótimo em algumas instâncias foi um pouco alto em relação aos demais (mais de 5%).

2.2 Ambiente de Testes

A meta-heurística estudada e implementada ao longo do desenvolvimento deste trabalho foi executada com as mesmas instâncias de testes usadas por diversos trabalhos de referência para possíveis efeitos comparativos. Todos os programas foram codificados utilizando a linguagem C++. Os testes sequenciais foram executados com as seguintes configurações:

- Processador Intel Core I5-9300H de 2,40 GHz;
- 16 GB de RAM;
- Placa de vídeo NVIDIA GeForce GTX 1650 com 4 GB de memória dedicada e 896 núcleos de processamento;
- Sistema operacional Windows 10;
- Arquitetura de processador 64 bits.

A avaliação dos programas desenvolvidos foi por meio de três conjuntos de testes distintos: o conjunto SAC-94, a biblioteca ORLIB e o conjunto GK. As instâncias do primeiro conjunto (SAC-94) deriva de problemas reais encontrados em diversos artigos, caracterizadas por um número de itens variando entre 10 e 105, e um número de restrições entre 2 e 30. Os testes que a biblioteca ORLIB trazem foram propostos por (Beasley, 1990) e possuem problemas com 5, 10 ou 30 recursos e 100, 250 ou 500 itens. Além disso, para cada combinação de recursos/itens, é introduzido um fator de aperto, representado por α (*tightness factor*). Este fator representa uma proporção entre o valor máximo para cada restrição j e a soma do valor requerido por cada item em relação ao recurso. O relacionamento entre eles pode ser expresso pela Equação 2.2:

$$b_j = \alpha \times \sum_{i=1}^{n} r_{ji}, \text{para } j = 1, 2, ..., m \tag{2.2}$$

A partir dessa equação, é possível afirmar que a instância do problema torna-se mais restrita à medida que α se aproxima de zero, pois o resultado do somatório será reduzido. A biblioteca, em cada uma de suas configurações, abrange 10 instâncias de teste, totalizando 270 problemas distintos. Por fim, o conjunto GK, criado por (Glover

e Kochenberger, 2016), compreende 11 instâncias mais complexas, com números de itens variando entre 100 e 2500, e números de recursos variando entre 15 e 100.

Para mensurar a qualidade das soluções, utilizou-se dois conceitos: *gap* e desvio-padrão (DP). O conceito de *gap* representa a porcentagem da diferença entre a solução obtida e a solução de referência, ou seja, o quão distante a solução obtida está do valor ótimo conhecido para cada instância. Esse cálculo é realizado utilizando a Equação (2.3):

$$gap = 100 \times \left(1 - \frac{valorObtido}{valorReferencia}\right) \tag{2.3}$$

Já o desvio-padrão (DP) calcula a dispersão do *gap* de cada execução em relação à média geral. Dessa forma, se o valor obtido estiver mais próximo de 0, menos dispersos são os dados daquela população. Utiliza-se a Equação 2.4 para obter esse valor, onde são notadas as seguintes variáveis: o vetGap[i] representa o valor do *gap* obtido em cada execução i variando de um até o número máximo de iterações (t); e a variável mg representa a média dos *gaps*. No total foram realizadas 30 execuções para serem gerados dados estatísticos.

$$dp = \sqrt{\frac{\sum_{i=1}^{t} (\text{vetGap}[i] - \text{mg})^2}{t}} \tag{2.4}$$

CAPÍTULO 3

Meta-heurísticas Aplicadas

Neste capítulo serão apresentadas duas meta-heurísticas empregadas em nossa abordagem para buscar soluções eficientes no contexto do problema da mochila multidimensional. Serão detalhadas a definição e o pseudocódigo tanto da *Variable Neighborhood Search* quanto da *Simulated Annealing*.

3.1 Variable Neighborhood Search

A Busca de Vizinhança Variável, conhecida como *Variable Neighborhood Search* (VNS) em inglês, é uma meta-heurística que foi proposta por Mladenović e Hansen em 1997 (Mladenović e Hansen, 1997). Ela representa uma melhoria em relação à VND (*Variable Neighborhood Descent* – Descida em Vizinhança Variável), uma vez que incorpora uma busca local para efetivamente alcançar o ótimo local dentro de uma vizinhança específica. O algoritmo VNS realiza trocas sistemáticas entre vizinhanças para explorar o espaço de busca do problema. Durante a execução do algoritmo, é selecionado supostamente o melhor ótimo local calculado pelas buscas locais, correspondendo ao ótimo global. Os pseudocódigos dessas versões meta-heurísticas (VND e VNS) são descritos nos Algoritmos 3.1 e 3.2, respectivamente.

Algoritmo 3.1: VND()

```
1  s ← Gerar uma solução inicial de forma aleatória;
2  k ← 1;
3  para k ≤ K_max faça
4  |  s' ← Obter a melhor solução na vizinhança N^(k)(s);
5  |  se f(s') > f(s) então
6  |  |  s ← s';
7  |  |  k ← 1;
8  |  senão
9  |  |  k ← k + 1;
10 |  fim
11 fim
12 retorna s;
```

Algoritmo 3.2: VNS()

```
1  s ← Gerar uma solução inicial de forma aleatória;
2  enquanto critério de parada não for satisfeito faça
3  |  k ← 1;
4  |  para k ≤ K_max faça
5  |  |  s' ← Gerar um vizinho qualquer na vizinhança N^(k)(s);
6  |  |  s" ← Busca-Local(s');
7  |  |  se f(s") > f(s) então
8  |  |  |  s ← s";
9  |  |  |  k ← 1;
10 |  |  senão
11 |  |  |  k ← k + 1;
12 |  |  fim
13 |  fim
14 fim
15 retorna s;
```

Os algoritmos VND e VNS fazem uso de diversas vizinhanças (representadas pela letra N - *Neighborhood*), as quais serão definidas no Capítulo 4. As vizinhanças disponíveis variam de 1 até K_{max} e desempenham o papel de encontrar vizinhos distribuídos por todo o espaço de busca, permitindo a análise e teste de várias alternativas de solução. Além disso, nos algoritmos, é incorporada uma função de avaliação $f(s)$ que visa ser maximizada (linha 5 e linha 7, respectivamente, VND e VNS), representando, no estudo deste trabalho, o valor obtido pelos itens incluídos na mochila. Ao final da execução dos programas, deseja-se obter a solução ótima (s), isto é, não deve existir uma solução s^* pertencente ao conjunto de soluções possíveis tal que $f(s^*) > f(s)$.

No VNS, parte-se de uma solução inicial gerada s e, a cada passo, escolhe-se aleatoriamente um vizinho s'. Em seguida, utiliza-se uma estratégia de busca local dando como entrada a solução s' e guardando o resultado obtido em s''. Se o valor da função objetivo do novo resultado encontrado pela busca for maior que o armazenado em s, reinicia-se a busca na primeira vizinhança, pois agora há uma solução superior. Caso contrário, apenas ocorre a troca para a próxima vizinhança.

Algumas observações podem ser notadas no algoritmo apresentado do VNS e elas são descritas abaixo.

- Após cada busca local, o objetivo principal é ter encontrado, supostamente, o ótimo local na vizinhança k. Isso ocorre porque nosso algoritmo de busca, a priori, deve retornar a melhor solução ou aprimorar a solução atualmente armazenada, embora não possamos garantir que essa busca seja perfeita. Essa afirmação pode ser constatada ao analisar a linha 11 do pseudocódigo, onde existe a possibilidade de não retornar mais àquela vizinhança. Por exemplo, considere que $k = 1$ e o algoritmo obtenha um resultado verdadeiro na linha 7 (comparação); assim, será guardado o valor obtido e k será reiniciado para o valor original. Porém, se a partir desse ponto a solução não melhorar ou não retornar o melhor local, o algoritmo apenas alterará a vizinhança até chegar no número máximo.

- Ao final do código, a solução s armazenará o melhor ótimo local obtido entre as vizinhanças ou a melhor solução encontrada. Essa solução é considerada ótima global se, teoricamente, todo espaço de busca foi mapeado e explorado, e o ótimo local foi encontrado em todas as vizinhanças. A verificação das soluções s e s'' dá essa garantia, pois caso tenha sido a melhor solução na comparação, seus dados serão armazenados imediatamente.

- A cada melhoria em relação à solução s, volta-se na primeira estrutura de vizinhança definida. Por certo, a tomada de decisão em reiniciar serve justamente pela dificuldade em mapear todo espaço de busca já que ele, normalmente, é extenso. Através disso, pode ser gerado um vizinho melhor do que foi anteriormente e através da busca melhorar ainda mais o valor final.

Similarmente, ocorre na meta-heurística VND. Porém, nela, há uma sensibilidade à disposição das estruturas de vizinhança que pode impactar no resultado final obtido uma vez que não existe uma busca local envolvida e recaía à função de melhorar e diversificar a solução na definições de vizinhanças. Essa talvez seja a maior dificuldade nela. Mais informações sobre as características do VND podem ser encontradas em (Possagnolo, 2015).

3.2 Simulated Annealing

A Têmpera Simulada, conhecida como *Simulated Annealing* (SA) em inglês, é uma meta-heurística que foi proposta por (Cerny, 1985) e (Kirkpatrick, Gelatt e Vecchi, 1983). Ela utiliza um conceito físico de resfriamento até um ponto que tenha energia mínima. Para evitar mínimos locais, ele utiliza uma busca aleatória que, dependendo do nível de energia, aceita novas soluções que diminuem o valor da função objetivo ao invés de sempre pegar a que maximiza. O Algoritmo 3.3 contém o seu pseudocódigo.

Bertsimas e Tsitsiklis (Bertsimas e Tsitsiklis, 1993) definem os principais elementos básicos do algoritmo SA:

- Um conjunto finito S;

- Uma função de custo real J definida em S;

- Para cada $i \in S$, existe um conjunto $S(i) \subset S - \{i\}$ chamado o conjunto de vizinhos de i;
- Para cada i, uma coleção de coeficientes positivos $q_{ij}, j \in S(i)$ tais que $\sum_{j \in S(i)} q_{ij} = 1$. Assume-se que $j \in S(i)$ se, e somente se, $i \in S(j)$;
- Uma função não crescente $T : I_+ \rightarrow (0, \infty)$, onde I_+ é o conjunto dos números naturais e $T(t)$ é a temperatura no tempo t. A função é chamada de cronograma de resfriamento;
- Um estado inicial $s(0)$ pertencente ao conjunto S.

Algoritmo 3.3: SimulatedAnnealing(t_0, t_f, *tam*, *factor*, s')

```
1  temp ← t_0;
2  MelhorSolução ← s';
3  enquanto temp ≥ t_f faça
4      para it ← 0, tam faça
5          NovaSolução ← s';
6          Escolhe um índice aleatório i ∈ {1,...,n};
7          se item i ∉ Solução então
8              Adiciona item i à NovaSolução;
9              enquanto NovaSolução não é viável faça
10                 Remove item aleatório de NovaSolução;
11             fim
12             Δ ← f(NovaSolução) - f(Solução);
13             num ← valor aleatório entre 0 e 1;
14             se Δ ≥ 0 ou num < exp[Δ/temp] então
15                 Solução ← NovaSolução;
16             fim
17         senão
18             Remove item i de NovaSolução;
19             Adiciona um novo item aleatório à NovaSolução;
20             Δ ← f(NovaSolução) − f(Solução);
21             num ← valor aleatório entre 0 e 1;
22             se Δ ≥ 0 ou num < exp[Δ/temp] então
23                 Solução ← NovaSolução;
24             fim
25         fim
26         se f(Solução) > f(MelhorSolução) então
27             MelhorSolução ← Solução;
28         fim
29     fim
30     temp ← factor × temp;
31 fim
32 retorna MelhorSolução;
```

A temperatura inicial e final, representadas, respectivamente, por (t_0) e (t_f), correspondem aos valores do sólido no início e no fim do programa. O tamanho da cadeia

de Markov (*tam*) define a quantidade de iterações executadas a cada temperatura. Por fim, o fator (*factor*) controla a taxa de resfriamento da temperatura. Considerando que a temperatura é atualizada de forma geométrica, o congelamento (término do programa) torna-se mais demorado à medida que se aproxima de 1.

CAPÍTULO 4

Implementação

No princípio, foi explorada uma abordagem simples para o VNS, envolvendo apenas uma vizinhança e uma busca local. A estrutura de vizinhança consistia em realizar uma rotação circular da solução atual por k posições à direita, sendo k a iteração atual. Por exemplo, o valor do índice i estará na posição $(i + k)$ % n, onde n é a quantidade de itens. A operação resto é necessária, pois a soma $i + k$ pode ultrapassar a quantidade máxima de posições do vetor. Quanto à busca local, um algoritmo foi implementado para criar uma nova solução s^k e tentar adicionar itens até alcançar uma quantidade máxima de tentativas. Embora essa abordagem não tenha gerado resultados satisfatórios por si só, contribuiu com diversas ideias de algoritmos para serem incorporados à nova versão do VNS para abordar o problema em estudo.

Além da versão apresentada no algoritmo 3.2, uma modificação adicional foi implementada para incrementar o tamanho máximo das vizinhanças (K_{max}). Essa adaptação não afeta as cinco estruturas de vizinhança, mas expande a frequência com que o algoritmo as utiliza, visando explorar mais o espaço de busca. Resumidamente, o algoritmo terá mais iterações, porém, mantendo o mesmo número de vizinhanças. Para calcular essa ampliação, é realizado um ajuste subtraindo um do valor da vizinhança atual, seguido pela aplicação do operador resto cinco. O valor resultante varia de zero a quatro, correspondendo, respectivamente, à primeira e à última vizinhança.

4.1 Mapeamento do problema em estudo à meta-heurística VNS

Para representação do problema da mochila multidimensional na meta-heurística VNS foram usadas três estruturas para representar s, s' e s''. Cada estrutura contém as seguintes informações:

- Vetor Itens: consiste em um vetor contendo n posições, representando os itens disponíveis para adicionar à mochila. Cada posição pode ter valor 1 ou 0, indicando, respectivamente, se o item está ou não presente na mochila;

- Vetor RC: consiste em um vetor contendo m posições, representando o quanto cada item levado está ocupando das restrições do problema;
- Valor: corresponde a um inteiro contendo a soma de todos os valores dos itens levados na mochila, isto é, corresponde à função objetivo $f(s)$, onde s é uma solução qualquer.

4.2 Solução Inicial

Inicialmente, foi proposta para a meta-heurística VNS uma geração totalmente aleatória para solução inicial. Nesse processo são gerados valores aleatórios entre 0 e 1 para cada item disponível para ser colocado na mochila. Após isso, verifica-se a viabilidade da solução gerada. Caso seja inviável, itens adicionados são removidos de forma aleatória. Por fim, são realizadas algumas tentativas de adicionar novos itens à solução com o objetivo de melhorá-la.

Entretanto, observou-se que a qualidade da solução inicial exerce uma influência significativa nos resultados finais, assim como diversos outros fatores, por exemplo: a estrutura de vizinhança e a técnica de exploração (Resende e Ribeiro, 2002). Diante dessa constatação, decidiu-se implementar a inicialização construída para o algoritmo GRASP visto que uma geração totalmente aleatória não conseguiu agregar com as estruturas de vizinhança criadas e com a busca local implementada. No algoritmo 4.1 tem-se o pseudocódigo utilizado.

Algoritmo 4.1: ConstróiSolução(*Solução*, β)

```
1 repita
2 |   Calcula as pseudoutilidades dos itens;
3 |   Constrói a RCL baseado no valor de gulo-randômico β;
4 |   Seleciona aleatoriamente um item e da RCL;
5 |   se e pode ser adicionado à Solução então
6 |   |   Solução ← Solução ∪{e};
7 |   fim
8 até que e não possa ser adicionado à Solução;
9 retorna Solução;
```

O procedimento `ConstróiSolução()` adota uma estratégia gulosa para construir a solução inicial. Nessa etapa, um conceito crucial associado é a *lista de candidatos restrita* (*restricted candidate list* - RCL), a qual contém os elementos que mais contribuem para a melhoria da solução em construção, sem comprometê-la. Uma função gulosa de avaliação realiza a seleção desses elementos assegurando que a RCL contenha os elementos que mais contribuem para maximizar o valor final da solução, uma vez que o objeto de estudo deste problema é um problema de maximização.

A escolha dos elementos da lista baseia-se nas pseudoutilidades de cada item i, as quais são calculadas na linha 3 por meio da aplicação da Equação 4.1 (Dantas, 2016):

$$psUt_i = \frac{v_i}{\sum_{j=1}^{m} r_{ji}/rc_j}, i = 1, \ldots, n \tag{4.1}$$

onde v_i representa o valor do item e rc_j indica o quanto do recurso j ainda está disponível. Os itens selecionados possuem a pseudoutilidade calculada dentro do intervalo $[c^{max} - \beta \times (c^{max} - c^{min}), c^{max}]$, em que c^{max} e c^{min} representam, respectivamente, o maior e o menor valor de pseudoutilidade calculados entre os itens. Além disso, nenhum item dessa lista pode comprometer a viabilidade da solução. O parâmetro β controla o grau de aleatoriedade do algoritmo; quando $\beta = 0$, o algoritmo é puramente guloso, enquanto no caso extremo de $\beta = 1$, ele opera de maneira totalmente aleatória.

4.3 Estruturas de Vizinhança

Definir boas estruturas de vizinhança com a finalidade de percorrer uma boa porção do espaço de busca é uma tarefa desafiadora uma vez que ainda não é conhecido nenhum procedimento padrão para a sua construção. Neste contexto, a abordagem é frequentemente guiada pela intuição e requer a realização de inúmeros testes experimentais para decidir as melhores entre elas. A seguir, são mostradas as estruturas usadas para o problema da mochila multidimensional.

- $N_1(s)$ (Troca Simples): dados dois índices i e j aleatórios, com valores diferentes, realizar a troca entre eles e verificar se a mochila continua viável (nenhuma restrição foi violada). Caso não esteja, retire o índice correspondente ao item que se encontra presente no vetor de itens e sorteie novos índices k e l e faça a mesma verificação;

- $N_2(s)$ (Troca Excessiva): dados dois índices i e j variando os índices de $1, ..., n$, para cada par (i, j) com $i \neq j$, verificar qual dupla aumenta o valor da mochila mantendo-a viável. Caso não tenha, retire dois itens aleatórios da mochila;

- $N_3(s)$ (*Path Relinking*): cria uma nova solução e faz um mapeamento de como transformá-la na atual solução do problema guardada em s. A cada passo, é selecionado o melhor movimento $m*$ que maximiza o custo da nova solução. O código se encerra quando não houver mais movimentos dentro do mapeamento;

- $N_4(s)$ (Rotação Circular): aplica uma rotação circular à direita de k posições. Se necessário para manter a mochila viável, retire itens dela de forma aleatória;

- $N_5(s)$ (*Empty Bucket*): dado um tamanho t divisor de n para o balde, escolher de forma randômica uma partição $\{(1, \ldots, t), (t + 1, \ldots, 2t), \ldots, (n - t + 1, \ldots, n)\}$ que contenha mais elementos adicionados a mochila para retirá-los dela.

4.4 Técnica de Busca Local

Após a tentativa de utilizar uma busca local apenas usando a ideia de inserir elementos na solução, percebe-se a necessidade de uma busca mais exploratória uma vez que os resultados obtidos anteriormente não foram muito satisfatórios. Para tal, foi adicionado o *Simulated Annealing* como busca local da meta-heurística em estudo (VNS), tornando-se uma meta-heurística híbrida.

O algoritmo 3.3 foi configurado com os seguintes parâmetros:

- Temperatura inicial: 500;
- Temperatura final: 0,00001;
- Tamanho da cadeia de Markov: número de itens multiplicado por 10;
- Taxa de atualização da temperatura: 0,85.

CAPÍTULO 5

Resultados

A meta-heurística VNS, aprimorada pela busca local realizada pelo *Simulated Annealing*, foi avaliada para dois grupos distintos. No primeiro grupo, a análise foi conduzida considerando a quantidade exata de vizinhanças predefinidas, enquanto no segundo grupo, foi aplicada uma adaptação para permitir um número maior de vizinhanças em comparação com o total originalmente estabelecido. Os algoritmos desenvolvidos foram executados utilizando instâncias de teste e código-fonte disponíveis no repositório online (Januário, 2023), seguindo algumas configurações usadas por Dantas (Dantas, 2016). A seguir, serão listadas todas as variáveis:

- Solução Inicial:
  - Beta (β): 0,1.
- Busca Local:
  - Temperatura inicial: 500;
  - Temperatura final: 0,00001;
  - Tamanho da cadeia de Markov: igual ao número de itens multiplicado por 10;
  - Taxa de atualização da temperatura: 0,85.

A seguir, nas Tabelas 5.1, 5.2 e 5.3, apresentam-se os resultados alcançados pela meta-heurística híbrida com o $K_{max} = 5$.

Tabela 5.1: Resultados da implementação sequencial de *Variable Neighborhood Search* para as instâncias de SAC-94 utilizando $K_{max} = 5$.

Conjunto	Número de Problemas	Ótimos	*Gap* Mínimo	*Gap* Médio	DP	Tempo(s)
HP	2	0	0,6437	3,5209	0,9853	0,0020
PB	6	20	0,0000	5,2644	2,6527	0,0029
PET	6	56	0,0000	0,6450	0,3136	0,0021
SENTO	2	1	0,0000	2,4174	1,0545	0,0080
WEING	8	142	0,0000	0,1869	0,3384	0,0074
WEISH	30	73	0,0000	1,8197	1,1005	0,0035

Tabela 5.2: Resultados da implementação sequencial de *Variable Neighborhood Search* para as instâncias de ORLIB agrupadas por configurações utilizando $K_{max} = 5$.

m	n	α	Ótimos	*Gap* Mínimo	*Gap* Médio	DP	Tempo(s)
5	100	0,25	0	0,1843	2,8183	1,1224	0,0076
		0,50	0	0,0214	1,4863	0,7152	0,0095
		0,75	0	0,0401	0,9065	0,4008	0,0117
	250	0,25	0	0,2784	1,4173	0,5097	0,0285
		0,50	0	0,1246	0,7111	0,2415	0,0353
		0,75	0	0,0481	0,4660	0,1612	0,0429
	500	0,25	0	0,2428	0,8880	0,2536	0,0903
		0,50	0	0,1288	0,4003	0,1237	0,1032
		0,75	0	0,0669	0,2606	0,0815	0,1287
10	100	0,25	0	0,5395	4,6847	1,4823	0,0079
		0,50	0	0,5306	2,2069	0,7936	0,0103
		0,75	0	0,2697	1,3234	0,4828	0,0126
	250	0,25	0	0,5070	2,1325	0,6005	0,0291
		0,50	0	0,2649	1,2304	0,3095	0,0354
		0,75	0	0,1712	0,6370	0,1924	0,0418
	500	0,25	0	0,2520	1,2435	0,3046	0,0863
		0,50	0	0,3283	0,7165	0,1863	0,1069
		0,75	0	0,0792	0,4042	0,1010	0,1308
30	100	0,25	0	1,3624	6,8523	1,8423	0,0131
		0,50	0	0,8544	3,4562	0,9982	0,0167
		0,75	0	0,5773	1,9798	0,5328	0,0188
	250	0,25	0	1,8953	4,2332	0,9022	0,0424
		0,50	0	0,8132	1,9932	0,4241	0,0535
		0,75	0	0,4597	1,1477	0,2577	0,0639
	500	0,25	0	0,9416	2,7753	0,4960	0,1261
		0,50	0	0,4991	1,1435	0,2032	0,1551
		0,75	0	0,3270	0,7803	0,1551	0,1882

Tabela 5.3: Resultados da implementação sequencial de *Variable Neighborhood Search* para as instâncias de GK utilizando $K_{max} = 5$.

m	n	Ótimos	*Gap* Mínimo	*Gap* Médio	DP	Tempo(s)
15	100	0	0,8232	1,8534	0,513	0,0102
25	100	0	2,956	3,5969	0,3095	0,0159
25	150	0	2,5813	3,3039	0,3096	0,0302
50	150	0	1,1271	2,1663	0,4369	0,0355
25	200	0	0,741	1,4552	0,3199	0,0351
50	200	0	2,6069	2,9627	0,1466	0,0623
25	500	0	0,6557	1,987	0,5936	0,1803
50	500	0	0,7657	1,2932	0,2981	0,2179
25	1500	0	1,8163	2,2438	0,2019	1,3938
50	1500	0	2,2446	2,4675	0,1127	1,8045
100	2500	0	0,4893	1,2074	0,2417	7,9929

A seguir, nas Tabelas 5.4, 5.5 e 5.6, apresentam-se os resultados obtidos no último grupo usando a meta-heurística híbrida com o $K_{max} = 100$.

Tabela 5.4: Resultados da implementação sequencial de *Variable Neighborhood Search* para as instâncias de SAC-94 utilizando $K_{max} = 100$.

Conjunto	Número de Problemas	Ótimos	*Gap* Mínimo	*Gap* Médio	DP	Tempo(s)
HP	2	2	0,0000	1,3240	0,6358	0,0285
PB	6	59	0,0000	1,9881	1,5604	0,0395
PET	6	108	0,0000	0,1918	0,1014	0,0300
SENTO	2	2	0,0000	0,7392	0,5569	0,1333
WEING	8	206	0,0000	0,0273	0,0392	0,0887
WEISH	30	274	0,0000	0,6354	0,4850	0,0621

Tabela 5.5: Resultados da implementação sequencial de *Variable Neighborhood Search* para as instâncias de ORLIB agrupadas por configurações utilizando $K_{max} = 100$.

m	n	α	Ótimos	*Gap* Mínimo	*Gap* Médio	DP	Tempo(s)
5	100	0,25	1	0,0000	1,8950	0,8515	0,1357
		0,50	1	0,0000	1,0947	0,5416	0,1708
		0,75	0	0,0951	0,6280	0,2506	0,2240
	250	0,25	0	0,3611	1,1622	0,3771	0,5495
		0,50	0	0,1137	0,5434	0,1973	0,7246
		0,75	0	0,0711	0,3540	0,1220	0,9390
	500	0,25	0	0,1808	0,6772	0,1891	2,0933
		0,50	0	0,0960	0,3308	0,1031	2,4356
		0,75	0	0,0336	0,2074	0,0611	3,1223
10	100	0,25	0	0,3246	3,3958	1,0607	0,1655
		0,50	0	0,2227	1,5312	0,4933	0,2227
		0,75	0	0,0395	0,9359	0,3302	0,2633
	250	0,25	0	0,5325	1,7684	0,4837	0,7300
		0,50	0	0,2570	0,9733	0,2396	1,0121
		0,75	0	0,1254	0,5101	0,1412	1,2193
	500	0,25	0	0,3221	1,0341	0,2550	2,4368
		0,50	0	0,2118	0,6013	0,1433	3,0809
		0,75	0	0,0572	0,3467	0,0813	3,8161
30	100	0,25	0	1,3709	4,3886	1,2081	0,3167
		0,50	0	0,6625	2,3199	0,6382	0,4145
		0,75	0	0,2421	1,3353	0,3572	0,4389
	250	0,25	0	1,6135	3,3885	0,6801	1,2405
		0,50	0	0,7057	1,7104	0,3624	1,5498
		0,75	0	0,3100	0,9959	0,2103	1,7918
	500	0,25	0	1,1027	2,3793	0,3867	3,8624
		0,50	0	0,4500	1,0443	0,1662	4,0778
		0,75	0	0,3958	0,7041	0,1255	5,4206

Tabela 5.6: Resultados da implementação sequencial de *Variable Neighborhood Search* para as instâncias de GK utilizando $K_{max} = 100$.

m	n	Ótimos	*Gap* Mínimo	*Gap* Médio	DP	Tempo(s)
15	100	0	0,6107	1,4401	0,4556	0,2129
25	100	0	1,5664	2,7228	0,4053	0,2719
25	150	0	1,8034	2,6456	0,3319	0,5628
50	150	0	0,8497	1,7554	0,3991	0,5907
25	200	0	0,741	1,1407	0,267	0,7234
50	200	0	1,89	2,3905	0,2396	1,28
25	500	0	0,6766	1,7557	0,493	5,4254
50	500	0	0,6009	1,081	0,3111	5,6192
25	1500	0	1,6355	2,0941	0,2191	55,3334
50	1500	0	1,9478	2,1664	0,13	69,8474
100	2500	0	0,8285	1,1743	0,1794	234,6823

A observação dos resultados obtidos com $K_{max} = 5$ revelou bons resultados. Em praticamente todos os agrupamentos, foi possível observar um *gap* mínimo inferior a um por cento. Além disso, os tempos médios de execução para as instâncias foram geralmente baixos, ficando abaixo de um segundo, com exceção das duas últimas instâncias do GK. O *gap* médio apresentou certa oscilação bem nítida nas instâncias do conjunto da ORLIB. Por fim, destaca-se que, para as instâncias SAC-94, foram alcançados valores ótimos de solução, exceto pela instância HP, que não atingiu o valor ótimo.

Ao adicionar mais iterações, isto é, aumentando K_{max} para o valor 100, foram obtidos resultados superiores em comparação ao experimento anterior. Agora, para todas as instâncias da SAC-94, o valor ótimo foi alcançado e em várias delas dobraram a quantidade de valores ótimos. Essa melhoria resultou em uma redução nos *gaps* médios. Na instância da ORLIB com 5 itens e 100 restrições, também foram encontradas as melhores soluções pelo algoritmo. Este aumento no número de vizinhanças não teve um impacto significativo no tempo, apresentando apenas um leve aumento. Nas instâncias da SAC-94, o tempo médio é inferior a um segundo. O aumento é mais perceptível nas instâncias da ORLIB, na qual o número de restrições é igual a 500, e nas últimas instâncias do GK.

Em ambos os experimentos, quando $\alpha = 0{,}25$, constata-se que, para instâncias da ORLIB, há um valor médio de *gap* mais elevado em comparação com as outras configurações. Provavelmente, um dos fatores contribuintes deve ser a restrição mais rigorosa da capacidade máxima, devido à diminuição do fator alfa. Além disso, o desvio padrão médio foi menor do que em grande parte das instâncias, levando à suposição de que talvez seja necessário modificar a estrutura de vizinhança para obter uma melhor diversificação.

CAPÍTULO 6

Conclusões

Devido a seu grande número de aplicações em diversas áreas, o problema da mochila multidimensional tem sido objeto de estudo há vários anos, buscando novas alternativas para resolver o problema e obter soluções ótimas em curto período de tempo. Como o problema é $\mathcal{NP}$-difícil, diversas técnicas vêm sendo analisadas para obter soluções de boa qualidade, uma vez que a busca pela solução exata em certas instâncias se torna inviável. A técnica que foi usada neste trabalho consiste em uma meta-heurística, pois em geral elas possibilitam alcançar bons resultados não só para o problema em estudo.

Este trabalho propôs a implementação de uma meta-heurística híbrida baseada na *Variable Neighborhood Search* (VNS) e *Simulated Annealing* (SA). Em muitas soluções, foram obtidos resultados de boa qualidade. No algoritmo proposto, com um leve aumento no número de vizinhanças, observou-se um aumento significativo no tempo médio de execução de algumas instâncias e uma diminuição do *gap* médio. Essa situação motiva a busca por novas técnicas para reduzir o tempo de execução, visando ainda encontrar soluções de melhor qualidade para o problema. A técnica de paralelização tem se mostrado bastante relevante, pois tem melhorado o tempo e o valor obtido em comparação aos algoritmos sequenciais.

Sugere-se como trabalhos futuros aprimorar o estudo de vizinhanças, seja por meio de modificações ou pela introdução de novas estruturas, visando mapear uma porção ainda maior do espaço de busca. Outro ponto de interesse seria investigar a viabilidade da paralelização do algoritmo, explorando diferentes abordagens, tanto nos núcleos dos processadores quanto nos núcleos da GPU. No caso da GPU, exigiria um esforço adicional para adaptar a meta-heurística híbrida desenvolvida. Além disso, seria válido considerar novas alternativas para a hibridização das técnicas estudadas, adicionando novas meta-heurísticas ou novas combinações entre elas.

Referências Bibliográficas

Beasley, J.E. (1990). "OR-Library: distributing test problems by electronic mail". Em: *Journal of the Operational Research Society* 41.11, pp. 1069–1072.

Bertsimas, D. e J. Tsitsiklis (fev. de 1993). "Simulated Annealing". Em: *Statistical Science* 8.1, pp. 10–15. DOI: 10.1214/ss/1177011077.

Blum, C. e A. Roli (2003). "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison". Em: *ACM Computing Surveys* 35.3, pp. 268–308. DOI: 10.1145/937503.937505.

Cáceres, E.N. e C. Nishibe (2005). "0-1 Knapsack Problem: BSP/CGM Algorithm and Implementation". Em: *Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2005)*, pp. 331–335.

Cerny, V. (1985). "A thermodynamic approach to the traveling salesman problem: An efficient simulation". Em: *Journal of Optimization Theory and Applications* 45, pp. 41–51.

Chu, P.C. e J.E. Beasley (jun. de 1998). "A Genetic Algorithm for the Multidimensional Knapsack Problem". Em: *Journal of Heuristics* 4.1, pp. 63–86. ISSN: 1381-1231. DOI: 10.1023/A:1009642405419.

Dantas, B.A. (ago. de 2016). "Metaheurísticas para o Problema da Mochila Multidimensional". Tese de dout.

Foulds, L.R. (1981). *Optimization techniques: An Introduction*. 1ª ed. Springer-Verlag.

Glover, F. e G. Kochenberger (abr. de 2016). *Benchmarks for the multiple knapsack problem*. URL: http://hces.bus.olemiss.edu/tools.html.

Januário, K.D. (2023). *Código-fonte e Instâncias de Teste*. https://github.com/klelberdias/research. Repositório do GitHub contendo o código-fonte e as instâncias utilizadas para testar a meta-heurística híbrida (VNS + SA).

Kellerer, H., U. Pferschy e D. Pisinger (jan. de 2004). *Knapsack Problems*. ISBN: 978-3-540-40286-2. DOI: 10.1007/978-3-540-24777-7.

Kirkpatrick, S., C.D. Gelatt e M.P. Vecchi (mai. de 1983). "Optimization by simulated annealing". Em: *Science (New York, N.Y.)* 220.4598, pp. 671–680. ISSN: 0036-8075. DOI: 10.1126/science.220.4598.671.

Lemos, D.V.X. e H.J. Longo (2021). "Uso de GPUs na resolução do Problema da Mochila Multidimensional". Em: *Anais da IX Escola Regional de Informática de Goiás*. Evento Online: SBC, pp. 68–81. DOI: 10.5753/erigo.2021.18434. URL: https://sol.sbc.org.br/index.php/erigo/article/view/18434.

Mladenović, N. e P. Hansen (1997). "Variable neighborhood search". Em: *Computers & Operations Research* 24.11, pp. 1097–1100. ISSN: 0305-0548. DOI: https://doi.org/10.1016/S0305-0548(97)00031-2. URL: https://www.sciencedirect.com/science/article/pii/S0305054897000312.

Possagnolo, L.H.F.M. (2015). "Reconfiguração de sistemas de distribuição operando em vários níveis de demanda através de uma meta-heurística de busca em vizinhança variável". Diss. de mestr. Universidade Estadual Paulista.

Resende, M.G.C. e C.C. Ribeiro (2002). "Greedy randomized adaptive search procedures". English. Em: *Handbook of Metaheuristics*. Ed. por F. Glover e G.A Kochenberger. Kluwer, pp. 219–249.

Stutzle, T. (1998). "Local search algorithms for combinatorial problems: analysis, improvements, and new applications". Diss. de mestr. Faculdade de Computação - Universidade Federal de Mato Grosso do Sul.

Wolsey, L.A. (1998). *Integer programming*. 1ª ed. Wiley Series in Discrete Mathematics e Optimization.