LVI SBPO SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
DE 04 ATÉ 07 NOVEMBRO 2024
FORTALEZA | CEARÁ

UMA MATHEURÍSTICA DE MELHORIA PARA O PROBLEMA DA MOCHILA COM CONJUNTOS DE PENALIDADES

Pedro Paulo A. de Paula e Silva, Francisco Ferreira Lima Neto, Pedro dos Santos Zanelato, Edna A. Hoshino
Universidade Federal de Mato Grosso do Sul//Faculdade de Computação
Av. Costa e Silva,s/n. Cidade Universitária. Campo Grande-MS.
{p_paulo,francisco.ferreira,pedro.zanelato,edna.hoshino}@ufms.br

RESUMO

Este trabalho apresenta uma matheurística para o Problema da Mochila com Conjuntos de Penalidades, o qual é uma variante, introduzida recentemente, para o problema clássico da mochila. O objetivo do trabalho é avaliar a eficácia de uma matheurística conhecida, a *Large Neighborhood Search*, aplicada a essa variante. A matheurística combina a destruição parcial de uma solução inicial com a sua reparação otimizada usando programação linear inteira. Os resultados são comparados com os obtidos por um algoritmo exato e pelo estado-da-arte, visando medir a qualidade das soluções geradas. O estudo demonstra que métodos como a *Large Neighborhood Search* podem ser competitivos para resolver variantes mais complexas do Problema da Mochila.

PALAVRAS CHAVE. Matheurísticas, Otimização combinatória, Programação Linear Inteira.

Tópicos: Programação Matemática, Metaheurísticas, Otimização combinatória

ABSTRACT

This work presents a matheuristic for the knapsack problem with forfeit sets, which is a recently introduced variant of the classic knapsack problem. The aim of this work is to evaluate the effectiveness of a known matheuristic, the *Large Neighborhood Search*, applied to this variant. The matheuristic combines the partial destruction of an initial solution with its optimized repair using integer linear programming. The results are compared with those obtained by an exact algorithm and the state-of-the-art, aiming to measure the quality of the generated solutions. The study demonstrates that methods like the *Large Neighborhood Search* can be competitive in solving more complex variants of the knapsack problem.

KEYWORDS. Matheuristics. Combinatorial Optimization. Linear Integer Programming.

Paper topics: Metaheuristics, Mathematical Programming, Combinatorial Optimization

DOI: 10.59254/sbpo-2024-193379
https://proceedings.science/p/193379?lang=pt-br

1. Introdução

O Problema da Mochila (KP, do inglês *Knapsack Problem*), é um clássico problema de otimização combinatória, muito estudado na literatura dado a sua aplicabilidade em diversos cenários, como planejamento de investimentos, alocação de recursos ou logística. Uma das variantes do KP bem conhecida é o Problema de Múltiplas Mochilas (MKP, do inglês *Multiple Knapsack Problem*). Um dos melhores algoritmos para o MKP foi proposto por Pisinger [1999], um algoritmo exato *branch-and-bound* com bom desempenho em instâncias grandes, posteriormente Fukunaga [2011] também propôs um algoritmo *branch-and-bound* para o MKP. Mais recentemente, Laalaoui e M'Hallah [2016] modelaram um problema de escalonamento de máquinas como um MKP. Heurísticas também têm sido propostas para o MKP, veja, por exemplo, Laalaoui [2013].

Outra variante do KP, introduzida recentemente, é o Problema da Mochila com Penalidades (KPF, do inglês *Knapsack Problem with Forfeits*), que considera uma coleçao S de pares de itens e uma penalidade $d_{a,b}$, para cada par $\{a, b\} \in S$, que deve ser paga se ambos os itens a e b forem levados na mochila. O Problema da Mochila com Conjuntos de Penalidades (KPFS, do inglês *Knapsack Problem With Forfeits sets*), introduzido mais recentemente por D'Ambrosio et al. [2023], generaliza o KPF, ao permitir subconjuntos $s \in S$ de qualquer cardinalidade e, nesse caso, paga-se uma penalidade d_s proporcional à quantidade de elementos de s que são levados na mochila e que excedem um limite h_s. Adicionalmente, há um limite k sobre o total de itens que podem exceder h_s. Ou seja, o KPF é um caso especial do KPFS em que $h_s = 1$ e $|s| = 2$, para cada conjunto $s \in S$, e k é igual ao total de itens.

O KPF foi introduzido em Cerulli et al. [2020], no qual um modelo de PLI e duas heurísticas são propostas para resolver o problema. Posteriormente, Capobianco et al. [2022] propuseram uma heurística híbrida para o KPF. Já para o KPFS, D'Ambrosio et al. [2023] apresentaram um modelo PLI, três heurísticas e um conjunto de instâncias de teste. Mais recentemente, Jovanovic e Voß [2024] propuseram uma matheurística para o KPFS.

Nas últimas décadas, o termo matheurística passou a ser usado para se referir a heurísticas que combinam programação linear ou programação linear inteira com metaheurísticas [Boschetti et al., 2023], como é o exemplo da matheurística abordada neste artigo. A principal motivação de se utilizar esses métodos é que segundo Maniezzo et al. [2021], essas estratégias tem se demonstrado muito eficazes em problemas altamente restritos, aonde as soluções viáveis se tornam escassas e uma simples busca local se torna cara e ineficiente.

Portanto, o objetivo deste trabalho consiste em avaliar a qualidade das soluções geradas por uma matheurística bem estudada na literatura, a *Large Neighborhood Search* (LNS), aplicada ao KPFS. Para atingir esse objetivo, testes computacionais são conduzidos em instâncias da literatura e os resultados comparados com o estado-da-arte.

O texto adiante está organizado da seguinte forma. A Seção 2 é dedicada à formulação do KPFS. Posteriormente, a Seção 3 descreve em detalhes a aplicação da LNS para o KPFS. A Seção 4 apresenta os resultados dos testes computacionais e a Seção 5 algumas considerações finais.

2. Definição Formal do Problema

Seja $I = \{1, 2, ..., n\}$ um conjunto de n itens e $S = \{S_1, \ldots, S_m\}$ uma coleção de subconjuntos de itens, ou seja, $S_j \subseteq I$, para cada $j \in J = \{1, \ldots, m\}$. Considere que cada item $i \in I$ tem um peso inteiro não-negativo p_i e um valor inteiro v_i . Adicionalmente, cada conjunto $S_j \in S$ está associado a dois inteiros, d_j e h_j, que representam uma penalidade e um limite de isenção. Dado um subconjunto de itens U, uma penalidade d_j deve ser paga a cada unidade de $|U \cap S_j|$ que exceder h_j , para cada $S_j \in S$.

O **Problema da Mochila com Conjuntos de Penalidades** consiste em, dados I, S e inteiros k e C, determinar um subconjunto U dos itens em I para serem transportados em uma

DOI: 10.59254/sbpo-2024-193379
https://proceedings.science/p/193379?lang=pt-br

mochila de capacidade C tal que a soma dos pesos dos itens em U não ultrapasse a capacidade C, o total de itens que violam os limites de isenção não exceda k e a soma dos valores dos itens transportados, descontando-se as penalidades, seja máxima.

Ou seja, o KPFS procura $U \subseteq I$ tal que:

(i) $\sum_{i \in U} p_i \leq C$,

(ii) $\sum_{j \in J} \max\{0, |U \cap S_j| - h_j\} \leq k$,

(iii) $\sum_{i \in U} v_i - \sum_{j \in J} \max\{0, (|U \cap S_j| - h_j)\} d_j$ seja máxima.

O KPFS é NP-difícil, uma vez que ele generaliza o KP [D'Ambrosio et al., 2023]. Portanto, pode ser muito improvável obter a solução ótima para todas as instâncias testes em um limite de tempo razoável. Nesses casos, a comparação da qualidade pode ser feita usando-se o limitante dual (superior) que é obtido pela relaxação linear do problema. Também será avaliada a melhora no desempenho do algoritmo exato ao incorporar a matheurística como rotinas de callback.

2.1. Modelo de Programação Linear Inteira

Um modelo de programação linear inteira (PLI) para o KPFS, que será usada no desenvolvimento da matheurística, utiliza as variáveis de decisão binárias x_i para cada item $i \in I$ e variáveis inteiras y_j para cada conjunto $j \in J$, de modo que $x_i = 1$ se e, somente, se o item i é selecionado para ser transportado na mochila e y_j é o total de itens em S_j que são transportados na mochila e que excedem h_j.

O modelo de PLI para o KPFS é dado por:

$$
\begin{aligned}
(F1) \quad \max \quad & \sum_{i \in I} v_i x_i - \sum_{j \in J} d_j y_j && && (1)\\
\text{s.a} \quad & \sum_{i \in I} p_i x_i \leq C, && && (2)\\
& \sum_{j \in J} y_j \leq k, && && (3)\\
& \sum_{i \in S_j} x_i - y_j \leq h_j, && \forall j \in J, && (4)\\
& x_i \in \{0,1\}, && \forall i \in I, && (5)\\
& y_j \in \mathbb{Z}, && \forall j \in J. && (6)
\end{aligned}
$$

Nesse modelo, a restrição (2) garante que a soma dos pesos dos elementos associados aos itens transportados na mochila não ultrapasse a sua capacidade e (3) que o total de violações não exceda k. Já as restrições (4) garantem que $y_j = |U \cap S_j| - h_j$. Por fim, as restrições (5) e (6) são as restrições de integralidade. A relaxação linear de um modelo de PLI consiste na relaxação das restrições de integralidade do modelo, permitindo que as variáveis discretas possam aceitar qualquer valor contínuo. O valor da relaxação linear é um limitante superior para a solução ótima de (F1).

3. Métodos Propostos

A matheuristica LNS proposta por Shaw [1998] é uma matheurística de melhoria, que iterativamente realiza três fases: destruição, reparação e aceitação. Na fase de destruição, destrói-se parte de uma solução inicial com base em um critério e, posteriormente, na fase de reparação,

DOI: 10.59254/sbpo-2024-193379
https://proceedings.science/p/193379?lang=pt-br

repara-se a solução de maneira ótima localmente. Na fase de aceitação, a solução inicial é substituída pela solução obtida na fase de reparação, conforme algum critério de aceitação. Por isso, esse método é também conhecido como *Destroy and Repair* e, segundo Pisinger e Ropke [2010], pertence a uma classe de problemas conhecidos como *Very Large Scale Neighborhood search* (VLSN) descrita por Ahuja et al. [2002], sendo essa a classe que compreende técnicas para destruir parte da solução e reconstruir com base em uma busca eficiente no espaço de soluções. A técnica pode ser considerada uma matheurística, pois o método utiliza, em geral, algoritmos exatos baseados em modelos de programação linear inteira na fase de reparação da solução.

Neste trabalho, o modelo descrito na Seção 2.1 é usado para resolver o KPFS de forma exata através de um algoritmo *branch-and-bound* com o LNS sendo resolvido em cada nó da árvore de enumeração e também para a fase de reparação do LNS. Dada uma solução $U \subseteq I$, considere que a solução parcial, após a fase de destruição, seja $\tilde{U}$. Na fase de reparação, considera-se o modelo $(F1)$ com as variáveis $x_i = 1$, para todo item $i \in \tilde{U}$, ou seja, as variáveis referentes aos itens não destruídos da solução são mantidos fixados no modelo. Desde que a parte destruída seja pequena, a fase de reparação não é computacionalmente custosa, pois o espaço de soluções viáveis é reduzido.

Como o LNS é um método de melhoria, uma metaheurística primal gulosa é utilizada para encontrar soluções viáveis. Dada uma solução parcial U, o critério guloso consiste em selecionar um item $i \in I \setminus U$ que satisfaça as seguintes condições:

(i) $i = \arg_{i \in I \setminus U} \max v_i - \sum_{j \in J : i \in S_j} d_j(\max\{0, 1 + |U \cap S_j| - h_j\})$;

(ii) $p_i + \sum_{i' \in U} p_{i'} \leq C$;

(iii) $\sum_{j \in J} \max\{0, |U \cup \{i\} \cap S_j| - h_j\} \leq k$.

A condição (i) refere-se à escolha gulosa do item que ainda não está em U e que maximiza o valor com a sua inclusão na solução descontando-se as penalidades advindas com a sua inclusão, caso exceda os limites de isenção. Já as condições (ii) e (iii) garantem que o item selecionado não viole a capacidade da mochila e o total de violações. Esse algoritmo guloso é resolvido em todo nó da árvore de enumeração do algoritmo *branch-and-bound*.

Na fase de destruição, são considerados dois critérios de remoção de itens $i \in I$ da solução U. O primeiro consiste em remover os que possuem menor razão v_i/p_i, isto é, os que valem menos por unidade de peso. O segundo critério é aleatório, isto é, os itens $i \in I$ são removidos aleatoriamente da solução.

Para o primeiro critério, o LNS é resolvido somente nos nós da árvore de *branch-and-bound* em que uma solução com valor melhor é encontrada, uma vez que o critério é guloso, ele não apresentaria nenhuma melhora. Para o segundo critério, a heurística é resolvida em todos os nós da árvore de enumeração, visto que o critério de remoção é aleatório.

Como a fase de destruição resolve o modelo $(F1)$ usando um algoritmo *branch-and-bound*, utilizamos como critério de parada um limite reduzido de tempo. Dependendo do tamanho da parte destruída e da capacidade residual da mochila, a fase de destruição pode encontrar a solução ótima para reparar a solução ou a melhor solução encontrada dentro do limite de tempo.

O Algoritmo 1 apresenta o pseudocódigo do algoritmo LNS que foi adaptado para o KPFS. Como o LNS é resolvido em muitos nós da árvore de enumeração, consideramos apenas uma iteração do LNS com as fases de destruição e reparação.

A solução de entrada U do LNS consiste na melhor solução encontrada até o nó atual da árvore de enumeração do *branch-and-bound*. No passo 2 do Algoritmo 1, um dos critérios de destruição é usado para destruir um percentual (dado pelo argumento D) da solução U e obter uma

DOI: 10.59254/sbpo-2024-193379
https://proceedings.science/p/193379?lang=pt-br

Algoritmo 1 *Large Neighborhood Search*

Input: Instância I, Solução U, porcentagem para Destruição D
Output: Nova solução U'
1: *fixados* $\leftarrow \emptyset$
2: $U' \leftarrow$ Destruir(U,D)
3: FixaVariáveis(*fixados*, U')
4: $U' \leftarrow$ ResolvePLI(*fixados*)

solução parcial U'. No passo 3, as variáveis x_i são fixadas em 1, para todo $i \in U'$. Por fim, no passo 4, o modelo $(F1)$ é resolvido com as variáveis fixadas para reparar a solução U'.

4. Resultados Computacionais

Os testes foram executados em uma máquina equipada com um Intel(R) Core(TM) i7-4790 (3.60 GHz), 32GB de memória RAM e com Linux. Os algoritmos foram implementados na linguagem C, usando o framework SCIP v3.2.1 para a implementação do *branch-and-bound*(`BB`), com o CPLEX v12.6.1.0, como *solver* da relaxação linear. Na implementação do `BB`, impomos um limite de tempo de 600 segundos, enquanto para a fase de reparação do LNS, consideramos um limite de tempo de 5 segundos.

As instâncias utilizadas para os testes computacionais são oriundas das instâncias introduzidas e usadas por D'Ambrosio et al. [2023]. Os testes foram realizados em 120 instâncias pequenas com $n = 300$ itens, agrupadas em 4 cenários. Os quatro cenários seguem os seguintes critérios:

- **Cenário 1:** O número de conjuntos de penalidade é $l = 5|I|$. O tamanho de cada conjunto de penalidade S_j é escolhido aleatoriamente no intervalo $[2, \frac{|I|}{50}]$ para $j = 1, \ldots, l$. Os limites de isenção para cada conjunto de penalidade S_j, para $j = 1, \ldots, l$ satisfazem $h_j = 1$. $\forall j \in J$.
- **Cenário 2:** O número de conjuntos de penalidade é $l = 3|I|$. O tamanho de cada conjunto de penalidade S_j é escolhido aleatoriamente no intervalo $[2, \frac{|I|}{20}]$ para $j = 1, \ldots, l$. Os limites de isenção para cada conjunto de penalidade S_j, para $j = 1, \ldots, l$ satisfazem $h_j = 1$. $\forall j \in J$.
- **Cenário 3:** O número de conjuntos de penalidade é $l = 5|I|$. O tamanho de cada conjunto de penalidade S_j é escolhido aleatoriamente no intervalo $[2, \frac{|I|}{50}]$ para $j = 1, \ldots, l$. O limite de isenção h_j para o conjunto de penalidade S_j é selecionada aleatoriamente do intervalo $[1, \frac{2}{3}|S_j|]$. $\forall j \in J$.
- **Cenário 4:** O número de conjuntos de penalidade é $l = 3|I|$. O tamanho de cada conjunto de penalidade S_j é escolhido aleatoriamente no intervalo $[2, \frac{|I|}{20}]$ para $j = 1, \ldots, l$. O limite de isenção h_j para o conjunto de penalidade S_j é selecionada aleatoriamente do intervalo $[1, \frac{2}{3}|S_j|]$. $\forall j \in J$.

Cada cenário possui três classes de instâncias, os quais são descritas a seguir:

- Não correlatos ou *Not Correlated* (NC): os pesos p_i e os valores v_i dos itens $i \in I$ são escolhidos aleatoriamente e de forma uniforme no intervalo $[0, 30]$ enquanto as penalidades d_j, de cada conjunto j no intervalo $[1, 20]$.
- Correlatos ou *Correlated* (C): os pesos dos itens e as penalidades são definidos como em NC, mas o valor de cada item i é calculado por $v_i = p_i + 10$.

DOI: 10.59254/sbpo-2024-193379
https://proceedings.science/p/193379?lang=pt-br

- Totalmente correlatos ou *Fully Correlated* (FC): os pesos dos itens são definidos como em NC, o valor dos itens da mesma forma que no cenário C, mas as penalidades dos conjuntos são calculados por $d_j = \left\lceil \frac{\sum_{i \in S_j^+} v_i}{|S_j|} \right\rceil$, no qual S_j^+ é o subconjunto dos $h_j + 1$ itens de maior valor em S_j.

Mais detalhes de quais padrões os parâmetros das instâncias seguem podem ser encontrados de D'Ambrosio et al. [2023].

A Tabela 1 faz uma comparação entre os dois critérios da fase de destruição do LNS descritos na Seção 3. Aqui, não consideramos as diferentes classes de instâncias em cada cenário para realizar a análise. O critério 1 refere-se ao critério guloso e o critério 2 é o aleatório. A coluna **Sol** apresenta a média do valor da melhor solução encontrada em cada cenário, enquanto **Best Sol Time**, o tempo médio para encontrar a melhor solução e **Gap**, o gap de dualidade médio, dado por $100(ub - lb)/ub$, com ub sendo o melhor limitante dual e lb o valor da melhor solução encontrada.

Tabela 1: Comparação entre os dois critérios de remoção

Cenário	**Critério 1**			**Critério 2**		
	Sol	**Best Sol Time**	**Gap**	**Sol**	**Best Sol Time**	**Gap**
1	711.53	361.7	14.74	703.1	191.7	16.94
2	400.7	210.8	16.20	392.0	189.6	25.14
3	985.5	59.2	0.37	984.7	215.4	1.80
4	903.6	21.98	0.0	903.4	134.9	0.58

Podemos perceber que para todos os cenários, o critério 1 obteve um valor da solução ótima, na média, superior ao critério 2. Já, em relação ao tempo médio para encontrar a melhor solução, o primeiro critério nos cenários 1 e 2 obteve um tempo pior, enquanto para o cenário 3 e 4, o critério 1 obteve um tempo consideravelmente inferior que o critério 2. Além disso, o critério 1 obteve gap médio melhor para os 4 cenários. Tendo em vista que o critério 1 obteve os melhores resultados, somente este critério (critério guloso), será considerado para as análises adiante.

As Tabelas 2 e 3 apresentam os valores médios da melhor solução obtida pelo LNS, usando o critério 1, e realiza uma comparação com os resultados da literatura. A coluna **MFSS** apresenta os resultados da *Matheuristic Fixed Set Search* [Jovanovic e Voß, 2024], que é o estado da arte na área e a coluna **MA** os resultados de um algoritmo memético proposto por D'Ambrosio et al. [2023]. A coluna **CPLEX** reúne os resultados obtidos pelo *solver* Cplex, em um tempo limite de 3h, disponibilizado na literatura [D'Ambrosio et al., 2023].

DOI: 10.59254/sbpo-2024-193379
https://proceedings.science/p/193379?lang=pt-br

Tabela 2: Comparação do LNS com o estado-da-arte, MFSS e MA, em instâncias de 300 itens, nos cenários 1 e 2 para o KPFS

Tipo	**Itens**	**Cenário 1**				**Cenário 2**			
		CPLEX	**LNS**	**MA**	**MFSS**	**CPLEX**	**LNS**	**MA**	**MFSS**
NC	300	684.0	*679.6*	681.4	684.0	299.6	299.6	299.1	299.6
C	300	769.5	*734.3*	768.7	772.0	443.9	439.4	443.9	443.9
FC	300	751.3	*720.7*	751.9	753.0	464.4	463	465.4	464.6
Média			711.53	734	736.3		400.6	402.8	402.7
#Melhor			0				0		
#Igual			0				1		
#Pior			3				2		

De acordo com a Tabela 2, podemos ver que, para o cenário 1, o LNS não obteve nenhum resultado significativo em relação à literatura, em nenhum dos três tipos de instância. Entretanto para o cenário 2, para a classe NC, a média das melhores soluções foi igual a do MFSS.

Tabela 3: Comparação do LNS com o estado-da-arte, MFSS e MA, em instâncias de 300 itens, nos cenários 3 e 4 para o KPFS

Tipo	**Itens**	**Cenário 3**				**Cenário 4**			
		CPLEX	**LNS**	**MA**	**MFSS**	**CPLEX**	**LNS**	**MA**	**MFSS**
NC	300	1033.3	*1033.3*	1027.7	1033.3	908.7	*908.7*	903.0	908.7
C	300	968.3	*968.3*	960.2	968.3	906.2	*906.2*	896.4	906.2
FC	300	955.0	*955.0*	952.0	954.8	895.8	*895.4*	889.7	895.8
Média.			985.5	979.9	985.4		903.5	896.3	903.5
#Melhor			1				0		
#Igual			2				2		
#Pior			0				1		

Para o cenário 4 descrito na Tabela 3, a LNS conseguiu apresentar soluções iguais ao estado da arte em duas classes, isto é para as instâncias da classe NC e C, dado que os algoritmos foram executados sob o mesmo tempo limite utilizado por Jovanovic e Voß [2024]. Para o cenário 3, a LNS apresentou para os tipos NC e C, resultados iguais a MFSS, e para o tipo FC um resultado melhor do que a MFSS, isto é, melhor do que o estado da arte.

5. Conclusões

Neste artigo uma generalização do Problema da Mochila, o Problema da Mochila com conjuntos de penalidade, foi resolvida com uma matheurística de melhoria baseada em destruição e reparação, de soluções iniciais geradas ou pela relaxação linear ou por uma metaheurística gulosa. Os testes computacionais realizados em instâncias pequenas da literatura mostraram que o método é competitivo com o estado da arte em alguns cenários de instâncias. Como trabalhos futuros, sugere-se testes adicionais em instâncias maiores e a avaliação de outros critérios para a fase de destruição, bem como o uso de outras matheurísticas.

6. Agradecimentos

O presente trabalho foi realizado com apoio da Universidade Federal de Mato Grosso do Sul – UFMS/MEC – Brasil.

DOI: 10.59254/sbpo-2024-193379
https://proceedings.science/p/193379?lang=pt-br

Referências

Ahuja, R. K., Ergun, Ö., Orlin, J. B., e Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102.

Boschetti, M. A., Letchford, A. N., e Maniezzo, V. (2023). Matheuristics: survey and synthesis. *International Transactions in Operational Research*, 30(6):2840–2866.

Capobianco, G., D'Ambrosio, C., Pavone, L., Raiconi, A., Vitale, G., e Sebastiano, F. (2022). A hybrid metaheuristic for the knapsack problem with forfeits. *Soft Computing*, 26:749–762.

Cerulli, R., D'Ambrosio, C., Raiconi, A., e Vitale, G. (2020). The knapsack problem with forfeits. In *Combinatorial Optimization: 6th International Symposium, ISCO 2020, Montreal, QC, Canada, May 4–6, 2020, Revised Selected Papers 6*, p. 263–272. Springer.

D'Ambrosio, C., Laureana, F., Raiconi, A., e Vitale, G. (2023). The knapsack problem with forfeit sets. *Computers & Operations Research*, 151:106093.

Fukunaga, A. S. (2011). A branch-and-bound algorithm for hard multiple knapsack problems. *Annals of Operations Research*, 184(1):97–119.

Jovanovic, R. e Voß, S. (2024). Matheuristic fixed set search applied to the multidimensional knapsack problem and the knapsack problem with forfeit sets. *OR Spectrum*, p. 1–37.

Laalaoui, Y. (2013). Improved swap heuristic for the multiple knapsack problem. In *Advances in Computational Intelligence: 12th International Work-Conference on Artificial Neural Networks, IWANN 2013, Puerto de la Cruz, Tenerife, Spain, June 12-14, 2013, Proceedings, Part I 12*, p. 547–555. Springer.

Laalaoui, Y. e M'Hallah, R. (2016). A binary multiple knapsack model for single machine scheduling with machine unavailability. *Computers & Operations Research*, 72:71–82.

Maniezzo, V., Stützle, T., e Voß, S. (2021). *Matheuristics: Algorithms and Implementations*. Springer.

Pisinger, D. (1999). An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114(3):528–541.

Pisinger, D. e Ropke, S. (2010). Large neighborhood search. In Gendreau, M. e Potvin, J.-Y., editors, *Handbook of Metaheuristics*, p. 399–419. Springer US, Boston, MA.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, p. 417–431. Springer.

DOI: 10.59254/sbpo-2024-193379
https://proceedings.science/p/193379?lang=pt-br