

UMA HEURÍSTICA GRASP COM PATH-RELINKING PARA O PROBLEMA DA MOCHILA COM PENALIDADES

Bruno Belo Comachio

Universidade Federal de Mato Grosso
Av. Fernando Corrêa da Costa 2367, Boa Esperança, Cuiabá - MT, 78088-800
comachiobelobruno@gmail.com

Eduardo Theodoro Bogue

Universidade Federal de Mato Grosso
Av. Fernando Corrêa da Costa 2367, Boa Esperança, Cuiabá - MT, 78088-800
eduardo.theodoro@ic.ufmt.br

Thiago Ferreira de Noronha

Universidade Federal de Minas Gerais
Rua Reitor Píres Albuquerque, Pampulha, Belo Horizonte - MG, 31270-901
tfn@dcc.ufmg.br

RESUMO

Neste trabalho, exploramos uma variante do tradicional Problema da Mochila, denominada Problema da Mochila com Penalidades. Nessa versão, é adicionado um conjunto composto por pares de itens distintos, conhecidos como pares de penalidades. Cada par consiste em itens cuja seleção simultânea na solução resulta em um acréscimo de penalidade ao custo total da solução. Apresentamos uma abordagem heurística baseada em GRASP com Path-relinking para resolver este problema. Os resultados obtidos revelam que a heurística supera outras abordagens da literatura, alcançando um *gap* médio inferior a 1% nas instâncias avaliadas.

PALAVRAS CHAVE.

Otimização Combinatória, L&T – Logística e Transportes, Problema da Mochila com Penalidades, GRASP, Heurística, Path-relinking.

ABSTRACT

In this work, we explore a variant of the traditional Knapsack Problem, known as the Knapsack Problem with Forfeits. In this version, a set composed of pairs of distinct items, known as forfeit pairs, is added. Each pair consists of items whose simultaneous selection in the solution results in an increase in penalty to the total solution cost. We present a heuristic approach based on GRASP with Path-relinking to solve this problem. The results obtained reveal that our heuristic outperforms other approaches in the literature, achieving an average gap of less than 1% on the evaluated instances.

KEYWORDS.

Combinatorial Optimization, L&T Logistics and Transport, Knapsack Problem with Forfeits, GRASP, Heuristics, Path-relinking.

1. Introdução

Problemas de otimização combinatória são frequentemente encontrados em diversas situações cotidianas, como alocação de recursos, agendamento, logística de transporte e distribuição, entre outras áreas. Exemplos clássicos incluem problemas de roteamento e empacotamento, que têm sido extensivamente estudados devido à sua importância e complexidade na área. Com o avanço das aplicações onde estes problemas se encontram, novas variantes podem surgir, seja por uma adição ou remoção de uma restrição ao problema. Neste trabalho, estudamos uma variação do clássico Problema da Mochila proposta em [Cerulli et al., 2020], denominada de Problema da Mochila com Penalidades (em inglês, *Knapsack Problem with Forfeits* - KPF).

O Problema da Mochila com Penalidades é definido sobre um conjunto de itens I , em que cada item $i \in I$ possui um peso $w_i \in \mathbb{N}$ e um lucro $p_i \in \mathbb{N}$. Ademais, adiciona-se um conjunto F de pares distintos de itens, denominados pares de penalidades. Cada par de penalidade $f \in F$ é formado por itens que, quando selecionados em conjunto para a solução, resultam no pagamento de uma penalidade $d_f \in \mathbb{N}$ ao custo da solução. O problema consiste em encontrar um subconjunto $I^* \subseteq I$, cuja soma dos pesos dos itens não ultrapasse uma capacidade $B \in \mathbb{N}$, e que a soma dos lucros dos itens, descontando as penalidades correspondentes, seja maximizada.

No contexto do KPF, a consideração de pares de penalidades é motivada pela presença de problemas com escolhas mutuamente exclusivas. Nessas situações, existe uma série de pares de itens em que, no máximo, um item de cada par pode ser incluído na solução [Yamada et al., 2002]. No entanto, no KPF, essa restrição é flexibilizada, permitindo que itens anteriormente mutuamente exclusivos sejam selecionados simultaneamente, desde que uma penalidade seja aplicada.

Podemos visualizar o conjunto de itens e seus respectivos pares de penalidades como um grafo $G = (V, A)$, em que cada item $i \in I$ é representado como um vértice $v \in V$ contendo uma tupla (w_i, p_i) , e cada aresta $(i, j) \in A$ representa um par de penalidades entre dois itens (i, j) , sendo o peso dessa aresta equivalente ao custo da penalidade d_f , em que $f = \{i, j\} \in F$, associada a inserção de ambos os itens na solução. A Figura 1 ilustra o exemplo de um grafo em uma instância do KPF.

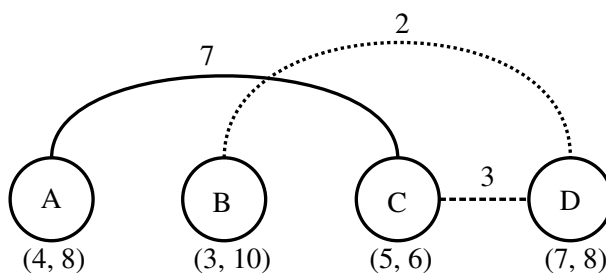


Figura 1: Exemplo de um grafo em uma instância do KPF.

O KPF encontra uma de suas aplicações em cenários de investimentos em ações, em que os itens representam possíveis investimentos a serem realizados. Nesse contexto, o peso de cada item corresponde ao custo do investimento, enquanto o lucro associado a cada item representa o retorno esperado em um determinado período. As penalidades entre pares de itens refletem a interferência negativa que ocorre quando certos investimentos são feitos simultaneamente, como investir no mesmo setor ou em empresas concorrentes, que podem impactar negativamente os lucros uns dos outros. Assim, a penalidade resultante da seleção de ambos os itens indica a diminuição dos lucros esperados devido à competição interna ou à saturação do mercado. Portanto, ao decidir quais investimentos realizar, é crucial maximizar o retorno total, considerando não apenas os custos, mas

também minimizando as penalidades decorrentes das interações negativas entre os investimentos selecionados.

Como o Problema da Mochila é NP-Difícil [Karp, 1972] e representa um caso específico do Problema da Mochila com Penalidades quando o conjunto de pares de penalidades $F = \emptyset$, temos então que o KPF também é NP-Difícil. Dado que não há métodos conhecidos para criar algoritmos exatos de tempo polinomial para problemas NP-Difíceis, este trabalho visa desenvolver algoritmos heurísticos para o problema. Em particular, apresentamos uma heurística *Greedy Randomized Adaptive Search Procedures* (GRASP) aplicada em conjunto com a técnica de *Path-relinking* para o KPF.

Este artigo está estruturado da seguinte maneira: na Seção 2 são abordados trabalhos relacionados. Na Seção 3 é apresentada uma formulação de programação linear inteira descrita em [Cerulli et al., 2020] para o KPF. Em seguida, na Seção 4 é detalhada a heurística GRASP proposta, enquanto a Seção 5 aborda a técnica de *Path-relinking* aplicada em conjunto com a heurística GRASP desenvolvida. Por fim, os experimentos computacionais são discutidos na Seção 6, seguidos das conclusões e sugestões para futuras pesquisas na última seção.

2. Trabalhos Relacionados

Em [Cerulli et al., 2020], os autores introduzem o Problema da Mochila com Penalidades, destacam suas potenciais aplicações e apresentam uma formulação de programação linear inteira, além de duas heurísticas propostas para abordar o problema, denominadas de *GreedyForfeits* e *CarouselForfeits*, sendo a primeira baseada em um método guloso e a segunda baseada no paradigma *Carousel Greedy* proposto em [Cerrone et al., 2017]. Para as instâncias pequenas avaliadas, os autores mostram que a heurística apresentou um *gap* médio de 4.7% entre a solução obtida pela heurística *CarouselForfeits* e a melhor solução obtida pela formulação de programação linear inteira resolvida pelo algoritmo de *branch-and-bound* do CPLEX, enquanto para as instâncias grandes o *gap* médio obtido foi de 4.5%.

Em [Moura et al., 2021] é apresentado o algoritmo *ILSForfeits*, sendo este baseado na heurística *Iterated Local Search* (ILS) [Lourenço et al., 2003]. A heurística ILS é um método de trajetória desenvolvida em torno de uma busca local, que utiliza-se de mecanismos para escapar da estagnação em ótimos locais e, assim, explorar novas áreas da região factível. A heurística *ILSForfeits* apresentou um *gap* e um tempo médio de 1.86% e 54.66 segundos, respectivamente, para o conjunto de instâncias pequenas, enquanto obteve um *gap* médio de 1.54% e 130.29 segundos quando executado nas instâncias grandes.

Já em [Capobianco et al., 2022], os autores propõem uma heurística híbrida denominada *GA-CG Forfeits* para o problema, de modo que a heurística utiliza-se do paradigma *Carousel Greedy* em conjunto com algoritmos genéticos. Na heurística proposta, cada solução é derivada de um processo evolucionário típico dentro dos algoritmos genéticos, sendo esta então refinada utilizando o paradigma *Carousel Greedy*. Este algoritmo híbrido conseguiu alcançar um *gap* médio de 1.62% e 1.71%, com um tempo de execução médio de 335.06 e 1192.39 segundos para as instâncias pequenas e grandes, respectivamente.

3. Formulação de Programação Linear Inteira

Nesta seção é apresentada uma formulação de programação linear inteira descrita em [Cerulli et al., 2020] para o Problema da Mochila com Penalidades. Sejam as variáveis binárias $x_i \in \{0, 1\}$, de modo que $x_i = 1$ se o item $i \in I$ for selecionado para a solução final e $x_i = 0$ caso contrário. Além disso, as variáveis v_f são definidas, onde $v_f = 1$ se os itens do par de penalidades $f = \{i, j\} \in F$ forem selecionados para a solução final, e $v_f = 0$ caso contrário. O KPF pode ser formulado como um problema de programação linear inteira, conforme definido em (1) - (5).

$$\text{Maximize } \sum_{i \in I} p_i x_i - \sum_{f \in F} d_f v_f \quad (1)$$

$$\text{sujeito a } \sum_{i \in I} w_i x_i \leq B \quad (2)$$

$$x_i + x_j - v_f \leq 1, \quad \forall f = \{i, j\} \in F \quad (3)$$

$$0 \leq v_f \leq 1 \quad \forall f \in F \quad (4)$$

$$x_i \in \{0, 1\} \quad \forall i \in I \quad (5)$$

A função objetivo (1) busca maximizar a soma dos lucros dos itens selecionados, considerando os custos associados aos pares de penalidades. A restrição de capacidade é definida pela desigualdade (2), em que B representa a capacidade da mochila, enquanto as restrições (3) garantem que se ambos os itens de um par de penalidades $f = \{i, j\} \in F$ forem selecionados, então $v_f = 1$, resultando na aplicação da penalidade d_f ao custo da solução. Por fim, as restrições (4) e (5) são restrições de integralidade.

4. Heurística GRASP-KPF

Considerando que a complexidade no pior caso dos algoritmos baseados na formulação PLI da Seção 3 cresce exponencialmente com o número de itens, propomos nesta seção uma heurística GRASP para o problema, denominada GRASP-KPF. A heurística multi-start GRASP [Feo e Resende, 1995] é um método iterativo composto por duas fases principais: a fase de construção e a fase de busca local. Durante a fase de construção, uma solução é desenvolvida de forma incremental, adicionando elementos de maneira gulosa, mas introduzindo aleatoriedade para explorar diferentes regiões do espaço de solução. Posteriormente, a solução corrente é refinada por meio de uma busca local, na qual um ótimo local encontrado a partir da vizinhança da solução corrente construída é retornado. Este processo iterativo de construção e busca local é repetido por várias iterações, sendo a melhor solução encontrada ao longo de todas as iterações selecionada como o resultado final.

Durante a fase de construção do GRASP-KPF, o processo iterativo envolve a inclusão gradual de itens na solução, até que a capacidade máxima da mochila seja alcançada ou não seja mais viável incluir itens adicionais. Em cada iteração dessa fase, apenas uma fração dos itens de maior densidade, ou seja, aqueles com uma melhor relação benefício-custo, são selecionados para compor a Lista Restrita de Candidatos (LCR). A LCR visa priorizar a seleção dos itens com uma alta relação benefício-custo, facilitando a eficácia na busca por soluções de qualidade. No contexto do KPF, a LCR é formada por uma seleção dos itens de maior densidade, onde a densidade reflete o benefício de adicionar um determinado item à solução, considerando seu lucro, peso e as penalidades associadas. Para um determinado item $i \in I$, seja $F_i \subseteq F$ o conjunto contendo todos os pares de penalidades que envolvem o item i . A densidade de um item i é inicialmente calculada como $\text{densidade}_i = \frac{(\frac{p_i}{w_i})}{CP_i}$, onde o coeficiente CP é determinado por $CP_i = \frac{\sum_{f \in F_i} d_f}{|F_i|} + 1$.

A razão da escolha para essa medida de densidade é avaliar um item não apenas com base em sua densidade tradicional $(\frac{p_i}{w_i})$, mas também considerando os pares de penalidades dos quais ele faz parte. Isso ocorre porque, geralmente, é mais vantajoso escolher um item com poucas penalidades em vez de um que tenha muitas. Portanto, o coeficiente de penalidade de um item i é calculado como a razão entre a soma das penalidades de todos os pares de penalidades contendo o item i , e a quantidade de pares de penalidade em que o item i está envolvido.

Após a inserção de um item $j \in I$ a solução, para todo item $i \in I$ que ainda não foi analisado e que forma um par de penalidade com o item j , ou seja, $f = \{i, j\} \in F_j$, temos que seu lucro p_i é reduzido pelo custo da penalidade d_f associado ao par, expresso pela equação $p_i = p_i - d_f$. Essa ação ocorre porque a inclusão do item j na solução resultará no pagamento da penalidade associada ao par $\{i, j\} \in F$ caso o item i também seja selecionado. Além disso, o par $\{i, j\}$ é removido do conjunto F_i , uma vez que o item j já foi incluído na solução.

No KPF, a LCR é composta por uma porcentagem α dos itens de maior densidade, em que α é um parâmetro do algoritmo. Em cada iteração da fase de construção, um item i contido na LCR é adicionado à solução atual S se sua inclusão não exceder a capacidade da mochila, ou seja, $(\sum_{i \in S} w_i x_i) + w_i \leq B$, além de sua adição incrementar o valor da solução, de modo que a *densidade* _{i} > 0 . O Algoritmo 1 apresenta o pseudocódigo do procedimento de construção da heurística GRASP-KPF.

Algoritmo 1: Pseudocódigo do procedimento de construção do GRASP-KPF.

Entrada: I : Conjunto de itens.
 B : Capacidade da mochila.
 F : Pares de Penalidade.
Saída: Uma solução S viável.

```

1  $S \leftarrow \{\}$ 
2  $\text{peso\_corrente} \leftarrow 0$ 
3  $LCR \leftarrow \text{construirLCR}(I, \alpha)$ 
4 while  $\text{peso\_corrente} < B$  and  $|I| > 0$  do
5   Seleccione aleatoriamente um item  $i \in LCR$ 
6   if  $\text{peso\_corrente} + w_i \leq B$  and densidade $i$   $> 0$  then
7      $S \leftarrow S \cup \{i\}$ 
8      $\text{peso\_corrente} \leftarrow \text{peso\_corrente} + w_i$ 
9     for  $f = \{i, j\} \in F_i$  do
10       $p_j \leftarrow p_j - d_f$ 
11       $F_j = F_j \setminus f$ 
12    $I = I \setminus \{i\}$ 
13    $LCR \leftarrow \text{construirLCR}(I, \alpha)$ 
14 return  $S$ 
  
```

Nas duas primeiras linhas do Algoritmo 1 é feita a inicialização do algoritmo, de modo que a solução S inicialmente encontra-se vazia e o *peso_corrente* da mochila é 0. Na linha 3 a função *construirLCR*(I, α) é responsável por construir a LCR, de modo que a função retorna uma porcentagem α dos itens de maior densidade. O laço das linhas 4-13 é executado enquanto tivermos capacidade livre na mochila e não tenhamos analisado todos os itens. Selecionamos então na linha 5 de maneira aleatória um item i da LCR, de modo que o item será incluído à solução S caso sua adição não exceda a capacidade da mochila (linha 6). Caso o item seja inserido à solução, nas linhas 7 e 8 incluímos o item i à solução S e adicionamos ao *peso_corrente* o peso w_i do item i , respectivamente. Já no laço das linhas 9-11, atualizamos o lucro de todos os itens $j \in I$ tal que $f = \{i, j\} \in F$, visto que a adição do item i a solução acarretará no pagamento da penalidade d_f . Em seguida, retiramos o item i do conjunto I de itens e reconstruímos a LCR nas linhas 12 e 13,

respectivamente. Por fim, na linha 14 a solução S construída é retornada.

Após a fase de construção descrita, um procedimento de busca local é realizado na solução S gerada pela etapa de construção. Nesse procedimento, foi utilizada uma estrutura de vizinhança $N(2)$, em que dois movimentos são realizados, de modo que um movimento consiste na inserção ou remoção de um item da solução atual. No primeiro tipo de movimento, todos os itens $i \in I$ são considerados para possível modificação na solução. No segundo tipo de movimento, um item j associado ao item i do primeiro movimento, de modo que $\{i, j\} \in F$, é escolhido para possível alteração na solução. Essa abordagem permite uma exploração ampla de soluções vizinhas, permitindo mudanças tanto em itens individuais quanto em pares de itens de penalidades. O procedimento de busca local adota a estratégia de primeira melhoria (*first improvement*). Seja S' a primeira solução vizinha encontrada de modo que $f(S') > f(S)$, onde $f(\cdot)$ é uma função que retorna o lucro de uma solução S caso esta seja viável, realizamos então uma nova busca local em S' . Essa iteração recursiva persiste até que um ótimo local obtido a partir da solução S seja alcançado.

Algoritmo 2: Pseudocódigo do procedimento de busca local do GRASP-KPF.

Entrada: Uma solução S viável.

I : Conjunto de itens.

B : Capacidade da mochila.

F : Pares de Penalidade.

Saída: Um ótimo local obtido a partir da solução S .

```

1   $S' \leftarrow S$ 
2  for  $i \in I$  do
3      if  $i \in S'$  then
4           $S' = S' \setminus \{i\}$ 
5      else
6           $S' = S' \cup \{i\}$ 
7      if  $f(S') > f(S)$  then
8           $S \leftarrow \text{buscaLocal}(S')$ 
9          return  $S$ 
10     else
11         for  $\{i, j\} \in F$  do
12              $S'' \leftarrow S'$ 
13             if  $j \in S''$  then
14                  $S'' = S'' \setminus \{j\}$ 
15             else
16                  $S'' = S'' \cup \{j\}$ 
17             if  $f(S'') > f(S)$  then
18                  $S \leftarrow \text{buscaLocal}(S'')$ 
19                 return  $S$ 
20 return  $S$ 

```

A busca local apresentada no Algoritmo 2 recebe como parâmetro de entrada uma solução viável previamente gerada pela fase de construção e retorna um ótimo local obtido a partir da solu-

ção S utilizando-se da estrutura de vizinhança $N(2)$. Na linha 1 inicializaremos S' com a solução S inicial. Já no laço das linhas 2 a 19, percorremos todos os itens do conjunto I , de modo que nas condicionais das linhas 3 a 6, removemos o item i da solução S se ele já fizer parte dela, ou adicionamos o item i à solução se ele ainda não estiver incluído. Caso esse movimento resulte em uma solução S' de maior lucro que a solução inicial S , repetimos o procedimento de busca local a partir da nova solução S' (linhas 7-9). Do contrário, um segundo movimento é realizado nas linhas (11-19). Para todo item $j \in I$ em que $\{i, j\} \in F$, inicializamos na linha 12 uma solução S'' a partir da solução S' atual e removemos o item j da solução S'' se ele já fizer parte dela, ou adicionamos o item j à solução se ele ainda não estiver incluído (linhas 13-16). Caso esse movimento resulte em uma solução S'' de maior lucro que a solução inicial S , repetimos o procedimento de busca local a partir da nova solução S'' (linhas 17-19). Por fim, na linha 20 o algoritmo retorna um ótimo local S encontrado.

Após a descrição dos procedimentos de construção e busca local, apresentamos no Algoritmo 3 o pseudocódigo da heurística GRASP-KPF. Nas linhas 1 e 2 inicializamos o lucro f^* da melhor solução encontrada até o momento para 0, e a melhor solução S^* para vazia. O laço das linhas (3-8) é executado um total de $IterMAX$ vezes, de modo que a cada iteração, executamos nas linhas 4 e 5 os procedimentos de construção e busca local descritos pelos Algoritmos 1 e 2, respectivamente. Caso o lucro da solução S retornada pelo procedimento da busca local seja superior ao lucro f^* da melhor solução encontrada até o momento, temos então que S^* e o seu lucro f^* são atualizados para S e $f(S)$, respectivamente (linhas 6-8). Ao final das $IterMAX$ iterações, a melhor solução S^* encontrada é retornada na linha 9.

Algoritmo 3: Pseudocódigo da heurística GRASP-KPF.

Entrada: I : Conjunto de itens.

B : Capacidade da mochila.

F : Pares de Penalidade.

Saída: S^* : Uma solução viável.

```

1  $f^* \leftarrow 0$ 
2  $S^* \leftarrow \{\}$ 
3 for  $Iter = 1, 2, 3, \dots, IterMAX$  do
4    $S \leftarrow \text{construção}(I, B, F, \alpha)$ 
5    $S \leftarrow \text{buscaLocal}(S)$ 
6   if  $f(S) > f^*$  then
7      $S^* \leftarrow S$ 
8      $f^* \leftarrow f(S)$ 
9 return  $S^*$ 

```

Neste trabalho, o valor de α , que determina o tamanho da LCR foi estabelecido em 0.1, isto é, a LCR contém 10% dos itens com maior densidade. Além disso, o número de iterações executadas pelo algoritmo GRASP foi fixado em $IterMAX = 5000$ iterações, de modo que ambos os parâmetros foram definidos experimentalmente.

5. Path-relinking

O Path Relinking é uma estratégia de busca em problemas de otimização combinatória, utilizada comumente para aprimorar soluções obtidas por algoritmos heurísticos como o GRASP [Laguna e Marti, 1999; Alvarez-Valdes et al., 2007; Gunawan et al., 2023]. Essa técnica envolve

a combinação de duas soluções e a geração de uma sequência de soluções intermediárias através de movimentos locais, de modo a favorecer a exploração de regiões mais diversas e potencialmente mais promissoras.

Uma estratégia fundamental empregada no Path Relinking é a utilização de uma memória de longo prazo conhecida como conjunto elite. Durante um processo denominado de *Relinking*, as soluções intermediárias são geradas combinando uma solução S com uma solução aleatória selecionada do conjunto elite. Essa abordagem permite que o algoritmo explore não apenas as soluções recentemente encontradas, mas também soluções de alta qualidade previamente armazenadas na memória de longo prazo.

Estruturamos o algoritmo GRASP-KPF de maneira que a cada iteração do algoritmo, um procedimento de *Relinking* é realizado entre a solução S retornada pela busca local daquela iteração e uma solução S^E do conjunto elite escolhida de maneira aleatória, de modo que o procedimento retornará a melhor solução encontrada no caminho de S até S^E . Caso a solução retornada pelo procedimento tenha um lucro superior a melhor solução S^* encontrada até o momento pela heurística GRASP-KPF, a solução S^* é atualizada. Ademais, a solução retornada é então inserida no conjunto elite até que este atinja seu limite de armazenamento. A partir desse ponto, novas soluções geradas têm seu lucro comparado com a solução de menor lucro no conjunto elite, sendo inseridas no conjunto elite apenas se seu lucro for superior ao da pior solução presente. Nesse caso, removemos do conjunto elite a solução S' que apresente a maior diferença simétrica com a solução recém-inserida. A diferença simétrica de duas soluções S_1 e S_2 é o conjunto dos itens que estão em S_1 ou em S_2 , mas não em ambos. Em outras palavras, é a medida dos elementos que diferem entre as duas soluções. Essa estratégia visa garantir que o conjunto elite contenha tanto soluções de alta qualidade quanto diversificadas.

O pseudocódigo do Algoritmo 4 ilustra o procedimento de *Relinking* aplicado a um par de soluções S e S^E . Na linha 1, computamos a diferença simétrica $\Delta(S, S^E)$ entre as duas soluções, representando o conjunto de movimentos necessários para alcançar S^E a partir de S . Nas linhas 2 e 3, inicializamos f^* e S^* , sendo f^* o lucro e S^* a melhor solução entre S e S^E . O laço das linhas (4-10) é executado enquanto a solução S não tenha alcançado S^E , ou seja $\Delta(S, S^E) \neq 0$. A cada iteração, o procedimento investiga todos os movimentos $m \in \Delta(S, S^E)$ da solução atual S e seleciona aquele que resulta na solução de maior lucro (linha 5), sendo aquele que maximiza $f(S \oplus m)$, em que $S \oplus m$ é a melhor solução resultante da aplicação do movimento m na solução S . Em seguida, o conjunto de movimentos disponíveis é atualizado e o melhor movimento m^* é aplicado a solução S corrente (linhas 6 e 7). Já na condicional das linhas (8-10), se a solução S obtiver um lucro superior à melhor solução S^* encontrada até o momento, atualizamos os valores do lucro f^* e da melhor solução S^* . Por fim, a melhor solução S^* é retornada na linha 11.

Neste trabalho, definimos experimentalmente o tamanho máximo do conjunto elite como 15 soluções.

6. Experimentos Computacionais

Os experimentos computacionais relatados nesta seção avaliam o desempenho do algoritmo ILS de [Moura et al., 2021], do algoritmo genético de [Capobianco et al., 2022], do algoritmo de *branch-and-bound* do CPLEX baseado na formulação (1)-(5), e do algoritmo GRASP proposto na Seção 4. Nos referimos a esses algoritmos como ILSForfeits, GA-CG, B&B e GRASP-KPF, respectivamente. Esses algoritmos foram implementados em C++ e compilados com o GNU *gcc* versão 6.3. A formulação de PLI foi resolvida pelo IBM CPLEX versão 12.6 utilizando configurações de parâmetros padrão e com um tempo de execução de 3 horas. Todos os experimentos foram realizados em um único núcleo de um processador Intel® Core™ i7-10700K @ 3.80GHz com 32 GB

Algoritmo 4: *Pseudo-código do procedimento de Relinking.*

Entrada: S : Solução inicial.
 S^E : Solução elite alvo.
Saída: S^* : Melhor solução no caminho de S até S^E .

- 1 Compute a diferença simétrica $\Delta(S, S^E)$
- 2 $f^* \leftarrow \max\{f(S), f(S^E)\}$
- 3 $S^* \leftarrow \operatorname{argmax}\{f(S), f(S^E)\}$
- 4 **while** $\Delta(S, S^E) \neq 0$ **do**
 - 5 $m^* \leftarrow \operatorname{argmax}\{f(S \oplus m) : m \in \Delta(S, S^E)\}$
 - 6 $\Delta(S \oplus m^*, S^E) \leftarrow \Delta(S, S^E) \setminus \{m^*\}$
 - 7 $S \leftarrow S \oplus m^*$
 - 8 **if** $f(S) > f^*$ **then**
 - 9 $f^* \leftarrow f(S)$
 - 10 $S^* \leftarrow S$
- 11 **return** S^*

de memória RAM.

Para esses experimentos, utilizamos o conjunto de instâncias descrito em [Moura et al., 2021], sendo este dividido em 4 subconjuntos, cada um composto por oito instâncias, onde o número de itens n varia entre 500, 700, 800 e 1000. Em cada instância, o número de pares de penalidade é equivalente a $6n$, sendo cada um deles composto por um par aleatório de itens, enquanto a capacidade B da mochila é igual a $3n$. Pesos, lucros e custos das penalidades são escolhidos aleatoriamente dentro dos intervalos $[3, \dots, 20]$, $[5, \dots, 25]$ e $[2, \dots, 15]$, respectivamente. Nas avaliações experimentais realizadas, comparamos os lucros das soluções heurísticas com o lucro da melhor solução obtida pelo algoritmo de B&B do CPLEX. Para essa comparação, definimos o *gap* da solução como $100 - 100 \times (\frac{Sol_{heu}}{Sol_{cplex}})$, onde Sol_{heu} consiste no lucro da solução da heurística utilizada e Sol_{cplex} no lucro da melhor solução obtida pelo algoritmo de B&B.

Na Tabela 1 são apresentados os resultados das instâncias contendo 500 e 700 itens, de modo que nesses conjuntos de instâncias o algoritmo de B&B do CPLEX foi capaz de obter a solução ótima para todas as instâncias avaliadas. Já na Tabela 2 apresentamos os valores médios do *gap* em relação ao lucro da melhor solução do algoritmo de B&B, e o tempo médio de execução das 8 instâncias de cada conjunto. Para essas instâncias, podemos observar através das Tabelas 1 e 2 que a heurística GRASP-KPF foi capaz de obter um *gap* inferior as demais heurísticas em todas as instâncias, obtendo um *gap* médio de 0.53%, enquanto as heurísticas GA-CG e ILSForfeits obtiveram um *gap* médio de 1.64% e 1.86%, respectivamente. Com relação ao tempo de execução, a heurística GRASP-KPF apresentou um tempo médio de 114.69 segundos, ficando entre os tempos médios da heurística ILSForfeits e da heurística GA-CG, que obtiveram um tempo médio de 54.66 e 334.21 segundos, respectivamente.

A Tabela 3 contém os resultados das instâncias contendo 800 e 1000 itens. Para essas instâncias, não foi possível encontrar a solução ótima através do CPLEX em um tempo limite de 3 horas, de modo que o *gap* é calculado baseado na melhor solução obtida pelo CPLEX. Na Tabela 4, apresentamos os valores médios do *gap* em relação ao custo da melhor solução encontrada e o tempo médio de execução das instâncias de cada conjunto. Podemos observar através das Tabelas 3 e 4 que o *gap* médio da heurística GRASP-KPF foi de 0.83%, enquanto as heurísticas GA-CG

Instância		GRASP-KPF		ILSForfeits		GA-CG		B&B
Nº itens	Id	Gap(%)	Tempo(s)	Gap(%)	Tempo(s)	Gap(%)	Tempo(s)	Solução
500	1	0.42	109.10	3.88	25.1	2.21	165.32	2626
	2	0.26	88.81	2.22	21.01	1.47	159.89	2660
	3	0.60	66.76	2.07	28.57	1.51	162.82	2516
	4	0.63	74.74	2.19	25.68	1.60	167.60	2556
	5	0.34	114.90	0.80	44.73	1.64	166.19	2625
	6	0.69	80.11	1.95	39.59	2.22	155.08	2615
	7	0.08	124.34	0.30	23.17	0.95	163.76	2627
	8	0.51	67.33	1.02	25.69	1.33	162.94	2556
700	1	0.33	136.81	2.20	44.48	2.17	517.19	3589
	2	0.63	128.71	1.58	95.05	1.22	507.37	3679
	3	1.04	123.86	2.24	55.56	1.61	495.49	3664
	4	0.66	118.58	2.25	49.21	0.77	494.02	3647
	5	0.44	137.05	1.11	148.9	1.20	498.85	3596
	6	0.23	155.70	3.13	72.49	2.71	513.84	3542
	7	0.72	152.99	1.55	57.11	2.04	501.05	3619
	8	0.82	155.34	1.31	118.23	1.59	515.93	3652

Tabela 1: *Gap* e tempo de execução das heurísticas avaliadas para as instâncias com 500 e 700 itens.

Nº itens	GRASP-KPF		ILSForfeits		GA-CG	
	Gap(%)	Tempo(s)	Gap(%)	Tempo(s)	Gap(%)	Tempo(s)
500	0.44	90.76	1.80	29.19	1.62	162.95
700	0.61	138.63	1.92	80.13	1.66	505.47
média	0.53	114.69	1.86	54.66	1.64	334.21

Tabela 2: Média do *gap* e do tempo de execução para instâncias com 500 e 700 itens.

Instância		GRASP-KPF		ILSForfeits		GA-CG		B&B
Nº itens	Id	Gap(%)	Tempo(s)	Gap(%)	Tempo(s)	Gap(%)	Tempo(s)	Solução
800	1	0.33	174.18	1.46	113.69	1.41	789.00	4184
	2	1.03	191.95	2.07	88.77	1.45	793.28	4065
	3	0.41	200.15	2.88	98.30	2.10	796.44	4104
	4	1.60	174.65	1.85	86.79	2.37	828.26	4056
	5	0.56	216.25	0.73	59.84	1.10	803.99	4086
	6	0.73	179.46	2.45	58.36	1.53	792.54	4249
	7	0.95	205.95	1.60	61.83	2.43	803.40	4121
	8	0.64	202.47	0.76	77.48	1.08	782.48	4063
1000	1	1.05	264.63	1.54	266.59	2.15	1597.80	4940
	2	0.52	252.75	1.53	117.52	1.53	1564.61	4969
	3	0.95	291.12	1.87	114.41	2.07	1645.26	5177
	4	0.60	236.00	1.59	244.69	1.52	1553.59	5143
	5	0.90	273.39	1.09	197.92	1.69	1526.05	5136
	6	0.95	233.57	0.87	215.63	2.50	1589.67	5078
	7	1.05	318.36	0.86	187.60	1.68	1585.42	5119
	8	1.04	252.32	1.56	87.28	2.16	1572.44	5183

Tabela 3: Gap e tempo de execução das heurísticas avaliadas para as instâncias com 800 e 1000 itens.

Nº itens	GRASP-KPF		ILSForfeits		GA-CG	
	Gap(%)	Tempo(s)	Gap(%)	Tempo(s)	Gap(%)	Tempo(s)
800	0.78	193.13	1.72	80.63	1.68	798.67
1000	0.88	265.27	1.36	178.96	1.91	1579.36
média	0.83	229.20	1.54	129.79	1.79	1174.01

Tabela 4: Média do gap e do tempo de execução para instâncias com 800 e 1000 itens.

e ILSForfeits apresentaram *gaps* médios de 1.79% e 1.54%, respectivamente, obtendo então um *gap* médio 46% inferior a heurística ILSForfeits. Com relação ao tempo de execução, a heurística GRASP-KPF apresentou um tempo médio de 229.20 segundos, ficando novamente entre os tempos das heurísticas ILSForfeits e GA-CG, que obtiveram um tempo médio de 129.79 e 1174.01 segundos, respectivamente.

Considerando todas as instâncias avaliadas, podemos observar a partir dos resultados obtidos que a heurística GRASP-KPF foi capaz de encontrar soluções mais próximas a solução ótima que as demais heurísticas em 30 das 32 instâncias avaliadas, sendo superado apenas pela heurística ILSForfeits em duas instâncias do subconjunto com 1000 itens.

7. Conclusões

Neste trabalho, foi proposto uma nova heurística para uma variante do Problema da Mochila, conhecida como o Problema da Mochila com Penalidades. Nesta variante, uma penalidade é aplicada ao lucro final toda vez que dois itens que compõem um par de penalidades são adicionados a solução. Uma heurística GRASP em conjunto com a técnica de *Path-relinking* é proposta para o problema, apresentando um *gap* médio em relação a melhor solução obtida pelo algoritmo de *branch-and-bound* de 0.53% para as instâncias com 500 e 700 itens, sendo aproximadamente 67% menor a heurística GA-CG, que obteve o segundo melhor resultado. Já para as instâncias com 800

e 1000 itens, a heurística GRASP obteve um *gap* médio de 0.83%, sendo 46% inferior a heurística ILSForfeits, que obteve o segundo melhor resultado para esse subconjunto de instâncias.

Como trabalhos futuros, podemos investigar o comportamento de novos métodos exatos para o problema. Isso inclui abordagens baseadas em programação por restrição e na análise poliédrica do problema, visando explorar suas propriedades estruturais. Desenvolver métodos de resolução eficientes que tirem proveito dessas características pode contribuir significativamente para avançar no entendimento e na solução do problema.

8. Agradecimentos

Esse trabalho foi financiado parcialmente pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), código de financiamento 001, pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPQ), e pela Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG).

Referências

- Alvarez-Valdes, R., Martí, R., Tamarit, J. M., e Parajón, A. (2007). Grasp and path relinking for the two-dimensional two-stage cutting-stock problem. *INFORMS Journal on Computing*, 19(2): 261–272.
- 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*, 26:1–14.
- Cerrone, C., Cerulli, R., e Golden, B. (2017). Carousel greedy: A generalized greedy algorithm with applications in optimization. *Computers & Operations Research*, 85:97–112.
- Cerulli, R., D'Ambrosio, C., Raiconi, A., e Vitale, G. (2020). The knapsack problem with forfeits. In *Combinatorial Optimization*, p. 263–272, Cham. Springer International Publishing.
- Feo, T. A. e Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *J. Glob. Optim.*, 6(2):109–133.
- Gunawan, A., Nguyen, D. V. A., Nguyen, P. K. M., e Vansteenwegen, P. (2023). Grasp solution approach for the e-waste collection problem. In *Computational Logistics*, p. 260–275, Cham. Springer Nature Switzerland.
- Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. e Thatcher, J. W., editors, *Proceedings of a symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, p. 85–103.
- Laguna, M. e Marti, R. (1999). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS J. Comput.*, 11(1):44–52.
- Lourenço, H. R., Martin, O. C., e Stützle, T. (2003). *Iterated Local Search*, p. 320–353. Springer US, Boston, MA.
- Moura, A. F. C., Noronha, T., e Bogue, E. T. (2021). *Uma heurística ILS para o Problema da Mochila com Penalidades*. Anais do Simpósio Brasileiro de Pesquisa Operacional, João Pessoa, PB.
- Yamada, T., Kataoka, S., e Watanabe, K. (2002). Heuristic and exact algorithms for the disjunctively constrained knapsack problem. In *Journal of Information Processing (JIP)*, volume 43(9), p. 2864–2870.