Matheurísticas para o problema da mochila com restrição de conflitos

Adriano Rodrigues Alves
Universidade Federal de Mato Grosso do Sul//Faculdade de Computação
Av. Costa e Silva,s/n. Cidade Universitária. Campo Grande-MS.
adriano.alves@ufms.br

Edna A. Hoshino
Universidade Federal de Mato Grosso do Sul//Faculdade de Computação
Av. Costa e Silva, s/n. Cidade Universitária. Campo Grande-MS.
edna.hoshino@ufms.br

RESUMO

Neste trabalho, três classes de matheurísticas são avaliadas para a construção de heurísticas para o problema da mochila com restrição de conflitos. As classes de matheurísticas avaliadas são: métodos de decomposição, métodos baseados na relaxação linear e métodos de melhoria. Para a heurística de decomposição, foram propostas heurísticas gulosas e aleatórias para resolver os conflitos seguidas por métodos exatos para resolver a mochila. A heurística baseada na relaxação linear proposta foi a *Relax&Fix* e a de melhoria foi o *Large Neighbourhood Search* (LNS). Testes computacionais foram conduzidos em instâncias da literatura para avaliar a qualidade das soluções. A combinação entre *Relax&Fix* e LNS obteve melhores resultados em 57% das instâncias, quando comparado com as demais, e o tempo gasto, na média, foi de 116 segundos. Em relação à literatura, o algoritmo proposto obteve resultados próximos do ótimo na maioria das instâncias com baixa densidade, mas não foi competitivo nas instâncias densas.

PALAVRAS CHAVE. Heurísticas, Programação Linear Inteira, Relax&Fix.

Tópicos: Metaheurísticas, Programação Matemática, Otimização Combinatória

ABSTRACT

In this work, three classes of matheuristics are analyzed for the construction of heuristics for the disjunctively constrained knapsack problem. The classes of matheuristic analyzed are: decomposition methods, methods based on linear relaxation, and improvement methods. For the decomposition heuristic, greedy and randomized heuristics were proposed to solve the conflicts and then, exact approaches are used to solve the knapsack problem. The heuristic based on linear relaxation proposed was *Relax&Fix* and the improvement heuristic was Large Neighbourhood Search (LNS). Computational experiments were conducted on instances of the literature to evaluate the quality of the solutions. The combination of *Relax&Fix* and LNS obtained better results in 57% of the instances, when compared to others, and the spent time, on average, was 116 seconds. With respect to the literature, the proposed algorithm obtained results close to optimal in almost of instances with small density, however it was not competitive in dense instances.

KEYWORDS. Heuristics, Integer Linear Programming, Relax&Fix.

Paper topics: Metaheuristics, Mathematical Programming, Combinatorial Optimization

1. Introdução

Como uma variante do problema da mochila, **o problema da mochila com restrição de conflitos**, do inglês *disjunctive constrained knapsack problem*, (DCKP) é definido da seguinte forma: dados um conjunto de n itens $I = \{1, \cdots, n\}$, em que cada item $i \in I$ possui um valor v_i e um peso p_i, ambos inteiros positivos, um conjunto de conflitos E, com $(i, j) \in E$ indicando que os itens i e j têm conflito, ou seja são **conflitantes**, e uma capacidade $C > 0$, o objetivo consiste em encontrar um subconjunto S de I que maximize a soma dos valores dos itens selecionados, respeitando as restrições de conflito e capacidade. A restrição de conflito impõe que nenhum par de itens em S tenha conflito enquanto a restrição de capacidade exige que a soma dos pesos dos itens em S não ultrapasse a capacidade C.

Um modelo de programação linear inteira (PLI) para o **DCKP** considera uma variável binária x_i, para cada item i, de modo que $x_i = 1$ se e somente se o item i é selecionado em S e é descrito a seguir.

(DCKP) Maximize

$$\sum_{i=1}^{n} v_i x_i \tag{1}$$

Sujeito a

$$\sum_{i=1}^{n} p_i x_i \leq C \tag{2}$$

$$x_i + x_j \leq 1, \forall (i,j) \in E. \tag{3}$$

$$x_i \in \{0, 1\}, i = 1, \cdots, n \tag{4}$$

A função objetivo (1) tenta maximizar o valor do subconjunto de itens selecionados a partir de I. A restrição (2) é utilizada para verificar a soma dos pesos dos itens selecionados para não ultrapassar a capacidade máxima da mochila. A restrição (3) é a restrição de conflitos, ou seja, itens conflitantes não podem estar, simultaneamente, na mochila. Por fim, a restrição (4) é a restrição de integralidade. Ela indica que os itens devem ser escolhidos de forma binária, ou seja, ou o item será selecionado ou não.

O DCKP é classificado como um problema NP-difícil, ou seja, não existe um algoritmo que resolve o problema em tempo polinomial, a menos que P=NP. Devido a isso e à importância deste problema, foram propostos vários algoritmos exatos, aproximados e heurísticas para a solução do problema, como constataremos na Seção 2.

Neste trabalho serão avaliadas matheurísticas para a obtenção de solução do DCKP, em tempo razoável, abdicando-se da otimalidade. Matheurísticas nada mais são do que metaheurísticas baseadas em modelos matemáticos. Segundo a classificação de matheurísticas definida em Ball [2011], neste trabalho, avaliamos três classes de matheurísticas: métodos de decomposição, métodos baseados na relaxação linear e métodos de melhoria. Métodos de decomposição seguem a premissa de dividir o problema em problemas menores e resolver um ou mais desses problemas menores de forma exata e os demais de forma heurística para construir uma solução, não necessariamente ótima, para o problema original. Para o DCKP, podemos dividi-lo em dois problemas menores, o problema da mochila 0-1 e o problema do conjunto independente. Para a solução do problema do conjunto independente, foram propostas heurísticas gulosas e aleatórias. Já o problema da mochila 0-1 será solucionado por métodos exatos.

Métodos baseados em algoritmos exatos se baseiam na ideia de alterar um ou mais passos de um método exato para acelerar a construção de uma solução, abdicando-se da otimalidade da

solução. Já os métodos baseados na relaxação são aqueles que utilizam a relaxação do problema para construir uma solução sub-ótima. Neste caso, foi utilizada uma heurística chamada *Relax&Fix*, que iterativamente usa a relaxação linear do problema e fixação de variáveis para construir uma solução parcial. Nesta matheurística, cria-se uma partição $P = (P_1, P_2, \ldots, P_k)$ das variáveis inteiras de um modelo de PLI do problema e, iterativamente, resolvem-se k subproblemas, cada um deles associado a uma das partes da partição. Na iteração i, somente as variáveis da parte P_i são consideradas inteiras, ou seja, as variáveis das demais partes são contínuas. Adicionalmente, cada variável das partes anteriores, isto é, em $P_1 \cup \ldots \cup P_{i-1}$ é fixada no valor da solução do subproblema correspondente em que foi considerada inteira. Desta forma, ao final das k iterações, todas as variáveis terão valores inteiros. A vantagem deste método é que cada subproblema é muito mais fácil de ser resolvido do que o problema original, muito embora isso dependa da escolha do tamanho de cada parte.

Métodos de melhoria são heurísticas que utilizam métodos exatos para completar, modificar ou melhorar uma solução inicial, usualmente gerada por outras heurísticas. A heurística de melhoria avaliada neste trabalho é a *Large Neighbourhood Search* que consiste em dois passos: o *Destroy* e o *Repair*. No passo do *Destroy*, parte da solução inicial é destruída. Por exemplo, no caso do DCKP, alguns itens são removidos da solução. No passo do *Repair*, um método exato é usado para completar a solução, destruída no passo anterior.

Para os métodos exatos, utilizamos algoritmos *branch-and-bound*. Nos experimentos computacionais realizados em instâncias da literatura, foi utilizada a biblioteca de otimização SCIP Gamrath et al. [2016] com o resolvedor Cplex IBM [2019][1].

Também será apresentada uma forma alternativa de se lidar com os conflitos. A forma de representar os conflitos na restrição (3) é denominada de **Modelo de Arestas**. Neste modelo, cada um dos conflitos é representado por dois "vértices", que seriam os itens, e uma "aresta", que seria o conflito. Nessa forma alternativa, será feito um tratamento dos conflitos e, a partir disso, será criado um **Modelo de Cliques** que, posteriormente, substituirá a restrição de modelo de arestas. Uma clique, que é um termo vindo da teoria dos grafos, é um subconjunto de vértices tais que cada um dos vértices do subconjunto é adjacente a todos os outros. Traduzindo para o nosso problema, uma clique será um subconjunto de itens com conflitos entre si, ou seja, são dois a dois conflitantes. Considere $\mathcal{C}$ um conjunto de cliques que cobrem todos os conflitos E, ou seja, para cada conflito (i, j) deve existir uma clique em C que contém ambos i e j. O modelo de cliques é descrito da seguinte forma:

$$\sum_{i \in C} x_i \leq 1, \forall C \in \mathcal{C}. \tag{5}$$

A restrição (5) nos diz o mesmo que a restrição (3). A diferença é que o número de itens por restrição pode ser maior, dependendo do tamanho das cliques.

O restante deste trabalho será dividido da seguinte forma. Na Seção 2 será feita uma revisão bibliográfica sobre o DCKP. Na Seção 3 serão apresentados os algoritmos propostos neste trabalho. A seção 4 apresenta e faz uma análise dos resultados obtidos pelos algoritmos. Por fim, a Seção 5 discute as contribuições do trabalho e apresenta algumas perspectivas de trabalhos futuros.

2. Trabalhos Relacionados

Fazendo uma breve revisão dos trabalhos relacionados ao tema, em 2002, Yamada et al. [2002] foi o trabalho que introduziu o problema, além de propor uma heurística gulosa com uma

[1] Ambas as ferramentas foram utilizadas sob licenças acadêmicas

busca local para melhorar a solução inicial. Nesse trabalho, os autores também propuseram um algoritmo de enumeração implícita e um método de redução intervalar para resolver o DCKP na otimalidade.

Em 2006, Hifi e Michrafy [2006] propuseram um algoritmo baseado em busca local reativa, onde se utiliza uma solução inicial gerada por dois algoritmos gulosos, com um processo de degradação para diversificar a busca. Neste trabalho, eles geraram um conjunto de 50 instâncias do DCKP com 500 e 1000 itens, que seriam usados em testes de trabalhos futuros.

Em 2007, Hifi e Michrafy [2007] propuseram três versões para um algoritmo exato baseado em estratégias de redução local.

Em 2011, Akeb et al. [2011] investigaram o uso de técnicas de *local branching* para a solução do DCKP. Os autores propuseram três versões do algoritmo.

Em 2017, Salem et al. [2017] propõem uma heurística de busca tabu probabilística com múltiplas estruturas de vizinhança. Essa heurística utiliza um procedimento de busca local com memória adaptativa para criar buscas mais flexíveis.

Em 2021, Wei e Hao [2021] apresentam um algoritmo memético baseado em pesquisa de limiar para resolver o DCKP, que combina a estrutura memética com pesquisa de limiar para encontrar soluções de alta qualidade.

3. Matheurísticas para o DCKP

Nesta seção será mostrado em detalhes o funcionamento dos algoritmos propostos para cada uma das classes de matheurísticas.

3.1. Métodos de decomposição

Para utilizarmos este método, foi necessário dividir o problema do DCKP em dois subproblemas: o problema do conjunto independente, do inglês *independent set problem*, (**ISP**) e o problema da mochila 0-1, do inglês *knapsack problem*, (**KP**). Para o ISP foram propostas duas heurísticas, uma gulosa e outra aleatória, descritos pelos Algoritmos 1 e 2. Na heurística gulosa, o conjunto de itens é ordenado de três maneiras: ordenado, de forma decrescente, pelo valor de cada item, ordenado, de forma crescente, pelo peso de cada item e ordenado, de forma crescente, pelo número de conflitos que cada item possui. Após a ordenação, um item é escolhido na ordem apresentado na lista e, após feita a escolha, é removido da lista de itens todos os itens que possuem conflito com o item escolhido. Isso é feito até que não tenha mais itens na lista. Para a heurística aleatória, os itens selecionados são escolhidos de forma aleatória, e, após a escolha de cada item, são removidos da lista todos os itens que possuem conflito com o item selecionado.

Algoritmo 1 Heurística Gulosa

Require: Instância I, Matriz de Conflitos, ordenação
Ensure: Vetor de itens sem conflitos

```
1: ordena(I, ordenação)
2: for cada item i ∈ I do
3:     seleciona item i
4:     for cada item j ∈ I do
5:         if existe conflito entre i e j then
6:             remove item j
7:         end if
8:     end for
9: end for
10: resolvePLI(I)
```

Algoritmo 2 Heurística Aleatória

Require: Instância I, Matriz de Conflitos
Ensure: Vetor de itens sem conflitos

```
while houver itens na lista I do
    seleciona item i aleatoriamente
    for cada item j ∈ I do
        if existe conflito entre i e j then
            remove item j
        end if
    end for
end while
resolvePLI(I)
```

Após a execução do Algoritmo 1 ou 2, é adquirido um subconjunto de itens onde não há conflitos entre si. Com isso, resolveremos agora o KP utilizando o método exato. Primeiro, é necessário escrever o modelo matemático que descreva o KP. Como foi anteriormente definido o modelo matemático para o DCKP, se removermos a restrição referente aos conflitos, reduzimos o modelo para o modelo que descreve o KP. Assim, o modelo fica da seguinte forma:

(KP) Maximize

$$\sum_{i=1}^{n} v_i x_i \tag{6}$$

Sujeito a

$$\sum_{i=1}^{n} p_i x_i \leq C \tag{7}$$

$$x_i \in \{0,1\}, i = 1, \cdots, n \tag{8}$$

Este modelo matemático serve tanto para a heurística gulosa quanto para a heurística aleatória. Durante os testes, a heurística gulosa foi executada utilizando os três tipos de ordenação, por valor, por peso e por número de conflitos, e a heurística aleatória foi executada cinco vezes para encontrar o melhor resultado. O método exato foi executado com o tempo máximo de 60 segundos.

3.2. Método baseado na relaxação linear

Para este método utilizamos uma matheurística denominada *Relax&Fix*, que funciona da seguinte forma: O conjunto I de itens é dividido em n partes menores de tamanho K, sendo que K é calculado levando-se em conta uma porcentagem dos itens do problema. Por exemplo, se a porcentagem for 10% e o total de itens do problema for 500, então 10% de 500 é 50, dando o número de partes que será criado para a matheurística. Portanto, o tamanho de cada parte será o total de itens dividido pelo total de partes, dando, no exemplo, partes de tamanho 10. Assim, o conjunto de partições será gerado de três formas: partição por conjunto independente, partição aleatória e partição sequencial.

Na construção por partição aleatória, os itens são escolhidos aleatoriamente para cada parte. Na construção sequencial, é feita uma ordenação dos itens pelo valor e os itens são escolhidos na sequência em que eles se apresentam na lista. Na construção por conjunto independente, os itens são escolhidos de modo a criar subconjuntos independentes em cada parte da partição.

O objetivo do *Relax&Fix* é resolver n problemas menores fazendo a escolha dos itens a partir da solução de um subproblema de PLI. Por exemplo, se o programa estiver executando a

k-ésima parte, as variáveis em n_k são restringidas a serem binárias, enquanto a integralidade das variáveis em $n_{k+1} \cup \cdots \cup n_K$ é relaxada e as variáveis em $n_1 \cup \cdots \cup n_{k-1}$ já estão fixadas.

Após a criação da partição, para cada uma das partes, é resolvido o subproblema de PLI de forma exata. Em seguida, é recuperado o valor que as variáveis adquiriram e, em seguida, elas serão fixadas para a iteração da próxima parte.

O pseudocódigo que descreve a ideia geral do método *Relax&Fix* aplicado ao DCKP é apresentado no Algoritmo 3.

Algoritmo 3 *Relax&Fix*

Require: Instância I, Conflitos C, porcentagem, tipo
Ensure: Vetor de itens selecionados

```
1: Ordenação(I)                          ▷ Ordena a lista de candidatos pelo valor de cada item
2: partes ← ∅                            ▷ Cria uma lista de partes
3: fixed ← ∅                             ▷ Cria uma lista de itens fixados
4: n ← totalItens                        ▷ Número total de itens
5: K ← (porcentagem * n)                 ▷ Número de partes
6: tamanho ← teto(n/K)
7: criaPartição(partes, tamanho, tipo, C)
8: for cada parte ∈ partes do
9:     for cada item i ∈ I do
10:        if item i pertence a parte atual then
11:            tornaBinário(i)
12:        end if
13:    end for
14:    resolvePLI(I, partes, fixed)
15:    for cada item i ∈ I do
16:        fixed[i] = valor[i]
17:    end for
18: end for
```

3.3. Método de melhoria

Para o método de melhoria, foi utilizada a heurística *Large Neighbourhood Search* (LNS). Essa heurística foi implementada utilizando o conceito de *Destroy&Repair*. Primeiro é necessário ter uma solução inicial, a qual é construída usando-se uma das soluções geradas pelas demais matheurísticas. Em seguida, ordenando-se os itens da solução inicial pelo peso e selecionam-se alguns itens da solução inicial para serem removidos. O número de itens que será removido depende de qual é a porcentagem do *Destroy* que será aplicado. Após feita a destruição de uma parte da solução inicial, é utilizado um método exato para reconstruir a solução, que corresponde à fase de reconstrução. Nesta fase, os itens que ainda permaneceram na mochila após a fase da destruição são fixados para o valor 1.0. Para a reconstrução, resolve-se o próprio DCKP, mas restrito aos itens que ficaram de fora da mochila e considerando a capacidade residual da mochila, ou seja, $C - \sum_{i \in S} p_i$, sendo S o conjunto dos itens que estavam na solução inicial e não foram removidos na fase de destruição. Desta forma, quanto menor for o percentual de destruição, menor será o problema DCKP a ser resolvido na fase de reconstrução.

O Algoritmo 4 descreve os passos principais do LNS proposto para o DCKP.

Algoritmo 4 LNS

Require: Instância I, Solução Inicial SI, porcentagem
Ensure: Vetor de itens sem conflitos
1: $ordena(SI)$
2: $removeItens(SI, porcentagem)$
3: **for** $i \in SI$ **do**
4: $\quad fixed[i] = 1.0$
5: **end for**
6: resolvePLI(I, $fixed$)

4. Resultados computacionais

Os algoritmos foram implementados na linguagem de programação C utilizando o gcc 4.9.2 dentro do frame-work SCIP (v3.2.1) usando o resolvedor de programação linear da IBM Cplex (v12.6.1.0) (CPLEX, 2019). Os experimentos foram realizados em um computador Desktop contendo uma Intel(R) Core(TM) i7-4790 (3.60 GHz) equipado com 32 GB RAM executando sobre o sistema operacional Linux onde o *multi-thread* estava desabilitado.

As matheurísticas propostas foram avaliadas em instâncias da literatura[2] e, posteriormente, comparadas com os resultados do estado-da-arte do problema. As instâncias são dividas em dois conjuntos, *set I* e *set II*. O *set I* é composto por 100 instâncias denominadas de jIk, onde $j \in \{1, \cdots, 20\}$ e $k \in \{1, \cdots, 5\}$. As 50 primeiras instâncias ($1Ik$ até $10Ik$) foram introduzidas em 2006 por Hifi e Michrafy [2006]. As outras 50 instâncias ($11Ik$ até $20Ik$) foram introduzidas em 2017 por Quan e Wu [2017]. O *set II* é composto por 930 instâncias denominadas $C15$, $R15$, SR e SC. Esse conjunto de instâncias foi introduzido em 2017 por Bettinelli et al. [2017] e expandido em 2021 por Coniglio et al. [2021]. Os dados das instâncias estão definidas na Tabela 1, são estes: o número de itens (n), a capacidade da mochila (C), a densidade dos conflitos (d) e o total de instâncias.

instance_name	n	C	d	#total
1Ik a 10Ik	{500,1000}	{1800,2000}	[0.05,0.1]	50
11Ik a 20Ik	{1500,2000}	{4000}	[0.04,0.2]	50
R15	$120 \leq n \leq 500$	15000	[0.1,0.9]	180
C15	$120 \leq n \leq 500$	15000	[0.1,0.9]	270
SC	$500 \leq n \leq 1000$	$1000 \leq C \leq 2000$	[0.001,0.05]	240
SR	$500 \leq n \leq 1000$	$1000 \leq C \leq 2000$	[0.001,0.05]	240

Tabela 1: Dados das instâncias

Na Tabela 2 são apresentadas as configurações das matheurísticas utilizadas. Nela está descrito, além da heurística e do método, a porcentagem para calcular o tamanho das partes para o *Relax&Fix*, o critério de ordenação utilizado em cada método e a porcentagem utilizada para o *Destroy*. A partir da Tabela 3 serão apresentados os resultados obtidos com os testes computacionais utilizando-se as diferentes matheurísticas. Cada uma das entradas das tabelas representa o número de vezes que determinada matheurística obteve o melhor resultado para determinado grupo de instâncias.

[2]As instâncias estão disponíveis no repositório *Mendeley Data* em: https://data.mendeley.com/datasets/gb5hhjkygd/1

heurística	método	tam partição	critério	*%Destroy*
A-PLI	Aleatório+Mochila PLI	-	-	-
G-PLI-v	Guloso+Mochila PLI	-	valor	-
G-PLI-p	Guloso+Mochila PLI	-	peso	-
G-PLI-c	Guloso+Mochila PLI	-	conflitos	-
RF-IS	*Relax&Fix*– Partição IS	{5,10,20}	-	-
RF-G	*Relax&Fix*– Partição Gulosa	{5,10,20}	valor	-
LNS	*Large Neighbourhood Search*	-	-	{10,30,50}

Tabela 2: Configurações das matheurísticas

Primeiro foram feitos os testes utilizando os métodos de decomposição. Para a heurística aleatória, o Algoritmo 2 foi executado para cada uma das porcentagens do LNS cinco vezes, e, dos cinco valores encontrados, foi escolhido o maior valor. Pela Tabela 3, a heurística aleatória com LNS de 50% foi a que produziu melhores resultados. O objetivo de se fazer os testes com a heurística aleatória foi avaliar qual a melhor porcentagem de LNS. Para atingir este objetivo, os testes foram restritos às instâncias do grupo 1Ik - 10Ik.

Heurística	1Ik - 10Ik
A-PLI+LNS(10)	4
A-PLI+LNS(30)	6
A-PLI+LNS(50)	**40**

Tabela 3: Resultados da Heurística Aleatória com LNS

Para a heurística gulosa, primeiro foram executados testes com o Algoritmo 1 sem LNS para verificar qual o critério de ordenação gera os melhores resultados.

Heurísticas	1Ik - 10Ik	11Ik - 20Ik
G-PLI-c	**35**	**50**
G-PLI-p	15	0
G-PLI-v	15	0

Tabela 4: Resultado da Heurística Gulosa - Critério de Ordenação

Analisando os resultados da Tabela 4, a heurística gulosa com o critério de ordenação por conflitos foi a que gerou os melhores resultados tanto para as instâncias de 1Ik - 10Ik quanto para as instâncias de 11Ik - 20Ik.

Após definido o critério de ordenação, o próximo passo foi executar o Algoritmo 1 com LNS para comparar as porcentagens da destruição. Os resultados deste teste estão apresentados na Tabela 5.

	Total		melhoria	
Heurísticas	1Ik - 10Ik	11Ik - 20Ik	1Ik - 10Ik	11Ik - 20Ik
G-PLI-c+LNS(10)	0	0	1.45	1.87
G-PLI-c+LNS(30)	0	0	4.03	3.30
G-PLI-c+LNS(50)	**50**	**50**	**6.75**	**4.20**

Tabela 5: Resultado da Heurística Gulosa - Conflitos + LNS

Pela Tabela 5, pode-se observar que o LNS com 50% foi o que obteve os melhores resultados, gerando uma melhoria média em torno de 6.75% para as instâncias de 1Ik - 10Ik e 4.20% para as instâncias de 11Ik - 20Ik.

Para o *Relax&Fix*, os testes foram conduzidos da mesma forma que para a heurística gulosa. Primeiro testamos as duas formas de se criar a partição, sequencial ou por *independent set*, com as diferentes porcentagens para o tamanho das partes. Depois, foi feita a comparação pela forma de criar a partição e pelo tamanho das partes, assim, o sequencial com 5% foi comparado com o IS com 5% e assim por diante. Os resultados das comparações podem ser vistas na Tabela 6.

Heurísticas	Partição	1Ik - 10Ik	11Ik - 20Ik
RF-IS	0.05	14	21
RF-IS	0.1	20	13
RF-IS	0.2	25	**34**
RF-G	0.05	**39**	**30**
RF-G	0.1	**32**	**37**
RF-G	0.2	**29**	16

Tabela 6: Resultados do *Relax&Fix* por construção da partição e tamanho das partes

Na Tabela 6, podemos notar que o *Relax&Fix* com partição sequenciais foi o que obteve os melhores resultados. Um detalhe é que para o sequencial com tamanho das partes de 20%, os resultados não foram tão bons. Isso se deve, pois as instâncias a partir da $17I1$ são instâncias grandes com uma densidade de conflitos alta, assim o *solver* teve dificuldades em encontrar a solução no limite de tempo proposto.

Por fim, pela conclusão que chegamos na heurística gulosa, aplicamos o LNS de 50% para melhorar os resultados do *Relax&Fix* e decidir qual o melhor tamanho para as partes.

Heurísticas	Partição	1Ik - 10Ik	11Ik - 20Ik
RF-G+LNS(50)	0.05	15	15
RF-G+LNS(50)	0.1	12	10
RF-G+LNS(50)	0.2	**30**	**31**

Tabela 7: Resultados do *Relax&Fix* - Partição Gulosa + LNS

Pela Tabela 7, o *Relax&Fix* com partição sequencial de tamanho 20% com LNS de 50% foi o que obteve os melhores resultados. Mesmo que no teste anterior o *Relax&Fix* sequencial com 20% tenha dado alguns resultados ruins, o LNS acabou corrigindo esses resultados.

Ao final dos testes utilizando o *set I* de instâncias, foram comparados os melhores resultados gerados pelas heurísticas. A Tabela 8 nos mostra essa comparação entre elas.

Heurística	1Ik - 10Ik	11Ik - 20Ik
A+LNS(50)	5	-
G-PLI-c+LNS(50)	19	**33**
RF-G(0.2)+LNS(50)	**33**	24

Tabela 8: Comparativo entre as matheurísticas

Pelos resultados da Tabela 8, tanto o *Relax&Fix* quando a heurística gulosa parecem produzir bons resultados. Assim, compararemos os resultados dessas duas matheurísticas no *set II* de

instâncias com os resultados alcançados por Wei e Hao [2021] utilizando o algoritmo **TSMBA**. Na Tabela 9 é mostrado essa comparação para as instâncias $C15$, $R15$, SC e SR.

Instância	TSMBA	RF-G(0.2)+LNS(50)	G-PLI-c+LNS(50)
C15	**270**	22	1
R15	**180**	35	1
SR	40	**205**	7
SC	**234**	210	75

Tabela 9: Comparação entre as heurísticas e o TSMBA

Observando-se a Tabela 9, nota-se que a heurística *Relax&Fix* com LNS usando o percentual de 50% produziu melhores resultados que a heurística de decomposição Gulosa com LNS. Também podemos observar que a primeira obteve resultados melhores ou competitivos com o da literatura nas instâncias esparsas, dadas pelas instâncias SR e SC.

5. Conclusão

Neste trabalho foram avaliadas três classes de matheurísticas: métodos de decomposição, métodos baseados na relaxação linear e métodos de melhoria. Cada um destes métodos foi submetido a um conjunto de instâncias fornecidos pela literatura a fim de se determinar o método que geraria os melhores resultados. Para isso, foi utilizado o conjunto de instâncias *set I* que contêm instâncias nomeadas de 1Ik - 10Ik e 11Ik - 20Ik, com $k \in \{1, \cdots, 5\}$. A partir dos resultados encontrados, tanto a heurística gulosa quanto o *Relax&Fix* produziram resultados satisfatórios, assim sendo candidatos a serem comparados com os resultados do estado-da-arte.

A comparação com os resultados do estado-da-arte foi possível graças aos autores em Wei e Hao [2021] disponibilizarem os seus resultados para o *set II* de instâncias. Assim, feita os testes e obtendo-se os resultados, podemos verificar que o *Relax&Fix* apresentou resultados competitivos e melhores que os resultados gerados pelo TSMBA para instâncias muito esparsas, enquanto a heurística gulosa obteve resultados não ótimos.

Como trabalhos futuros sugere-se avaliar outras classes de matheurísticas, como a classe dos métodos baseados em algoritmos exatos, que consiste em realizar modificações em um ou mais passos de algoritmos exatos e desta forma acelerar a construção da solução evitando comprometer a qualidade da solução. Tendo em vista que a qualidade das soluções geradas pelas matheurísticas propostas não foi boa em instâncias densas, outro trabalho futuro é avaliar os diferentes parâmetros das matheurísticas usando-se ferramentas de calibração automática como o *Irace*, López-Ibáñez et al. [2016]. Além disso, também é possível explorar outras formas de se decompor o problema e resolvê-lo. Uma forma seria manter a decomposição em ISP e KP, mas utilizar o *Relax&Fix* para resolver o problema do ISP e depois utilizar o método exato para resolver o problema da mochila. Também é possível resolver o problema da mochila por programação dinâmica, já que, para as matheurística onde houve a necessidade de se resolver o problema da mochila, o conjunto de itens era reduzido, o que favorece ainda mais o uso dessa técnica.

Referências

Akeb, H., Hifi, M., e Mounir, M. E. O. A. (2011). Local branching-based algorithms for the disjunctively constrained knapsack problem. *Computers & Industrial Engineering*, 60.

Ball, M. O. (2011). Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science*, 16(1):21–38. ISSN 1876-7354. URL https://www.sciencedirect.com/science/article/pii/S1876735410000036.

Bettinelli, A., Cacchiani, V., e Malaguti, E. (2017). A branch-and-bound algorithm for the knapsack problem with conflict graph. *INFORMS Journal on Computing*, 29.

Coniglio, S., Furini, F., e Segundo, P. S. (2021). A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. *European Journal of Operational Research*, 289.

Gamrath, G., Fischer, T., Gally, T., Gleixner, A. M., Hendel, G., Koch, T., Maher, S. J., Miltenberger, M., Müller, B., Pfetsch, M. E., Puchert, C., Rehfeldt, D., Schenker, S., Schwarz, R., Serrano, F., Shinano, Y., Vigerske, S., Weninger, D., Winkler, M., Witt, J. T., e Witzig, J. (2016). The scip optimization suite 3.2. Technical Report 15-60, ZIB, Takustr. 7, 14195 Berlin.

Hifi, M. e Michrafy, M. (2006). A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society*, 57.

Hifi, M. e Michrafy, M. (2007). Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & Operations Research*, 34.

IBM. *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual, Version 12 Release 6*. IBM Corp., 2019.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., e Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58. ISSN 2214-7160. URL https://www.sciencedirect.com/science/article/pii/S2214716015300270.

Quan, Z. e Wu, L. (2017). Cooperative parallel adaptive neighbourhood search for the disjunctively constrained knapsack problem. *Engineering Optimization*, 49.

Salem, M. B., Hanafi, S., Taktak, R., e Abdallah, H. B. (2017). Probabilistic tabu search with multiple neighborhoods for the disjunctively constrained knapsack problem. *RAIRO Operations Research*, 51.

Wei, Z. e Hao, J.-K. (2021). A threshold search based memetic algorithm for the disjunctively constrained knapsack problem. *Computers & Operations Research*, 136.

Yamada, T., Kataoka, S., e Watanabe, K. (2002). Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *IPSJ Journal*, 43.