Problema da Mochila 0-1 Bidimensional com Restrições de Disjunção

Thiago Alves de Queiroz,
Departamento de Matemática, UFG-CAC,
75704-020, Catalão, GO,
e-mail: th.al.qz@gmail.com.

Flávio Keidi Miyazawa
Instituto de Computação, IC, UNICAMP,
13084-971, Campinas, SP,
e-mail: fkm@ic.unicamp.br.

RESUMO

O problema da mochila 0-1 com restrições de disjunção (ou com grafo de conflitos) tem sido pouco explorado pela literatura. Além de ser um problema com aplicações práticas no contexto industrial, aparece como subproblema em outros problemas de otimização combinatória. Este trabalho lida com a versão bidimensional deste problema. Propomos uma heurística baseada em GRASP que, partindo de uma solução aleatória gulosa inicial, melhora-a iterativamente por meio de operações de perturbação e diversificação. O empacotamento dos itens é feito por um algoritmo que opera sobre pontos de cantos. Além disso, desenvolvemos um modelo de programação inteira que é resolvido pelo CPLEX. Os resultados computacionais mostram que a heurística é satisfatória, conseguindo soluções a $22,99\%$, na média, do ótimo, gastando pouco tempo computacional. Por outro lado, o modelo inteiro conseguiu soluções ótimas para 15 instâncias de pequeno a médio porte dado o limite de tempo de duas horas.

PALAVRAS-CHAVE. Problema da Mochila 0-1 Bidimensional, Restrições de Disjunção, Grafo de Conflitos, GRASP.

Área principal. Otimização Combinatória.

ABSTRACT

The 0-1 knapsack problem with disjunctive constraints (or with conflict graphs) have been little handled by specialized literature. This problem appears in practical scenarios as well as like subproblems when dealing with other combinatorial optimization problems. This paper deals with the two-dimensional version of this problem. We propose a GRASP based heuristic that generates an initial solution by a greedy randomized procedure and improves its using operations that perturb and diversify the solutions. To pack items, we use a heuristic that packs on corner points. Besides that, we also propose an integer linear model solved with CPLEX. The computational results show the competitiveness of the heuristic with solutions to 22.99%, on average, of optimal, and spending little CPU time. On the other hand, the integer model allowed optimal solutions for 15 small and medium size instances within the time limit of two hours.

KEYWORDS. Two-Dimensional 0-1 Knapsack Problem, Disjunctive Constraints, Conflict Graphs, GRASP.

Main area. Combinatorial Optimization.

1 Introdução

No problema da mochila 0-1 bidimensional (2D) com restrições de disjunção (2KPD) são dados um conjunto $V = \{1,2,\ldots,n\}$ de itens, cada item i (retangular) com largura l_i, comprimento c_i e valor/lucro v_i, e um recipiente B (mochila retangular) de largura L e comprimento C. Além disso, é dado um grafo de conflitos $G = (V,E)$ não-orientado, em que E é o conjunto de arestas, tais que $(i,j) \in E$ representa que os itens i e j são conflitantes e no máximo um deles pode ser empacotado na mochila, isto é, OU i está em B OU j está em B, mas não ambos. O objetivo do 2KPD é determinar como empacotar um subconjunto de itens de V em B, de forma a maximizar o valor total presente na mochila e o empacotamento seja viável.

Dizemos que um empacotamento em B é viável se, para quaisquer dois itens $i,j \in B$, tivermos: i e j empacotados de forma ortogonal, tal que seus lados são paralelos aos lados de B; i e j não podem ser rotacionados; i e j não se sobrepõem; e, $(i,j) \notin E$.

O 2KPD consiste em uma generalização do problema da mochila 0-1 bidimensional (2KP) que, por sua vez, generaliza o clássico problema da mochila 0-1 (1KP) que é NP-difícil (Garey e Johnson, 1979). Note que o 2KP é um caso especial do 2KPD quando E é vazio. O 2KPD é um problema com importantes aplicações práticas. Algumas aplicações do mundo real incluem a distribuição de alimentos organizados em paletes, em que alguns paletes não podem ser colocados no mesmo veículo. Além disso, aparece como subproblema quando se lida com o problema de empacotamento em *bins* (versão 2D) com conflitos ao usar estratégias como geração de colunas ou *branch-and-price* (Epstein et al., 2008; Sadykov e Vanderbeck, 2012).

Ao nosso conhecimento, não há trabalhos na literatura que tratam do 2KPD. Além disso, a literatura para a versão 1D deste problema (1KPD) não é muito extensa. Um dos primeiros autores a lidar com o 1KPD foi Yamada e Kataoka (2001); Yamada et al. (2002), que apresentaram heurísticas (gulosas) e algoritmos exatos. A principal heurística gera uma solução inicial gulosa que é melhorada por uma estratégia de busca em vizinhança. Esta solução é, então, usada como limitante inferior para o método exato, enquanto que limitantes superiores são obtidos pela relaxação do programa inteiro combinado com a relaxação das disjunções. Isto permitiu a estratégia exata resolver instâncias com até 1.000 itens e 10.000 restrições de disjunção. Hifi e Michrafy (2006) apresentaram um algoritmo de busca local reativa para resolver instâncias de grande porte. O algoritmo possui uma lista para evitar configurações repetidas, além de usar um princípio de degradação da solução para escapar de mínimos locais e diversificar a busca. Em Hifi e Michrafy (2007), os autores apresentaram três algoritmos exatos bem competitivos, se comparados ao CPLEX®. Tais algoritmos usam limitantes calculados através de heurísticas e da relaxação do modelo, além de restrições para a redução de domínio. Para lidar com instâncias de médio porte, Hifi e Michrafy (2007) reduziram o número de restrições de disjunção considerando um modelo equivalente em que, para cada item, todos os itens conflitantes a ele foram combinados em uma única restrição. Isto permitiu reduzir o número de restrições de disjunção para apenas o número de itens.

Pferschy e Schauer (2009) desenvolveram algoritmos pseudo-polinomiais para o 1KPD considerando classes especiais de grafos de conflitos, como árvores ou grafos cordais. Estes autores também derivaram esquemas de aproximação de tempo polinomial completo (FPTAS) partindo dos algoritmos iniciais. Recentemente, Akeb et al. (2011) lidaram com o 1KPD propondo algoritmos baseados na técnica de *local branching*. O objetivo destes autores era resolver instâncias de grande porte do problema em questão. Um primeiro algoritmo considera uma solução inicial obtida por um procedimento especializado de arredondamento, que trabalha na solução da relaxação do modelo inteiro. O segundo algoritmo, por sua vez, considera um procedimento de duas fases embutido no *local branching*. Por fim, um terceiro algoritmo combina o segundo algoritmo com uma estratégia de diversificação. Por outro lado, Hifi e Otmani (2012) propuseram dois algoritmos baseados na meta-heurística de busca dispersa (*scatter search*), com os quais conseguiram resolver várias instâncias em pouco tempo computacional.

Como comentado, existem outros problemas combinatórios que incluem grafos de conflitos e estão relacionados com o problema da mochila. Um destes é o problema de empacotamento em *bins* com conflitos, que foi introduzido por Jansen e Öhring (1997) e vem sendo investigado deste então, como em:

Gendreau et al. (2004); Khanafer et al. (2010); Muritiba et al. (2010); Khanafer et al. (2012). Algoritmos exatos e heurísticas para outras variantes do problema da mochila podem ser encontrados, por exemplo, em Birgin et al. (2012); Queiroz et al. (2012).

Neste trabalho, uma heurística baseada em GRASP (*Greedy Randomized Adaptive Search Procedure*) é proposta para o 2KPD. Uma solução inicial é obtida através de um procedimento que opera sobre uma lista de itens candidatos. Em seguida, um procedimento de busca local explora, através de perturbações, a vizinhança desta solução inicial. Diferentes estratégias para gerar novas soluções (vizinhos) são propostas e combinadas para diversificar as soluções e, consequentemente, escapar de mínimos locais. Também apresentamos um modelo de programação inteira, resolvido via CPLEX, que é usado para efeito de comparação dos resultados.

O trabalho está organizado da seguinte forma. Apresentamos na próxima seção a heurística desenvolvida. Começamos pela procedimento que gera a solução inicial, depois discutimos o algoritmo de busca local e o algoritmo usado para empacotar os itens na mochila. Por fim, detalhamos a heurística proposta. Na Seção 3 apresentamos o modelo de programação inteira do 2KPD usado nos testes comparativos. Os testes, por sua vez, são discutidos na Seção 4. Foram geradas 48 instâncias aleatórias, com diferentes tipos de itens, tamanhos de mochila e densidade do grafo de conflitos. Por fim, as conclusões e direções para trabalhos futuros são discutidos na Seção 5.

2 Heurística baseada em GRASP

Dado o problema 2KPD, representamos uma instância sua por $I = (B,V,G)$, tal que: $B = (L,C)$ corresponde a mochila de dimensões (L,C); $V = (l_{1,\ldots,n}, c_{1,\ldots,n}, v_{1,\ldots,n})$ é o conjunto de itens, sendo o item i de dimensões (l_i,c_i) e valor v_i; e, $G = (V,E)$ o grafo de conflitos. A representação de um empacotamento ocorre no plano Cartesiano $\mathbb{R}^2$. A mochila tem seu canto inferior esquerdo no ponto $(0,0)$ e o seu canto superior direito no ponto (L,C). Um item i é empacotado na mochila através do seu canto inferior esquerdo. Sem perda de generalidade, assumiremos que todos os valores em I são inteiros não negativos.

Em linhas gerais, a heurística proposta aqui possui três fases: (i) construção da solução inicial, (ii) busca local (exploração de vizinhos e melhora da solução inicial), que caracterizam uma estratégia GRASP; e, (iii) algoritmo de empacotamento bidimensional. Denominaremos esta heurística por G3F (Grasp com três Fases).

Na fase (i), uma solução inicial é gerada por um procedimento (aleatório) guloso que, ao avaliar um conjunto de itens da lista de candidatos restritos (*Restricted Candidate List - RCL*), a qual foi obtida da lista de candidatos, busca selecionar o item de melhor custo, segundo uma função de avaliação gulosa. Este procedimento finaliza após todos os itens da lista de candidatos terem sido avaliados.

A fase (ii), de busca local, explora soluções vizinhas à atual com o intuito de obter soluções melhores. Este processo consiste em gerar novos vizinhos por meio de operações que *perturbam* a solução atual, como trocar itens de posição e excluir/inserir um item. Ao gerar todos os vizinhos, obtém-se aquele de melhor benefício, tornando-o a solução atual. Como a fase (ii) pode ser iterada diversas vezes, uma estratégia simples para evitar convergência prematura para um ótimo local consiste em perturbar a melhor solução atual. Isto também diversifica as regiões a serem exploradas no espaço de soluções.

As fases (i) e (ii) basicamente trabalham sobre duas listas de itens: lista LP dos possíveis itens a serem empacotados pelo algoritmo de empacotamento; e, lista LN dos itens não presentes na solução final. A fase (iii) recebe os itens da LP como entrada e, então, executa um algoritmo para empacotar um subconjunto de tais itens em pontos de canto, seguindo os objetivos do 2KPD.

2.1 Construindo a Solução Inicial

Para obter a solução inicial, usamos um algoritmo (aleatório) guloso, Algoritmo 1, que preenche inicialmente as listas LP e LN com os itens de I. Os itens em LP, no final do algoritmo, formam uma solução inicial para o 2KPD.

Na verdade, o termo solução usado não é adequado, pois uma solução (mesmo que inicial) para o 2KPD consiste, além de uma lista de itens, de um empacotamento viável 2D destes itens (ou de um subconjunto destes itens). De qualquer forma, usaremos o termo solução, visto que um empacotamento 2D viável com os itens da lista será obtido no final do G3F.

Algoritmo 1: Sub-rotina para obter a solução inicial - SBegin

Entrada: Instância $I = (B, V, G)$ do 2KPD; limitante superior UB; tamanho q da lista de candidatos.
Saída: Listas LP e LN contendo os itens de I.

```
1.1  LP ← { }; LN ← { }; L ← V; valorI ← 0.
1.2  enquanto L possuir itens a serem avaliados faça
1.3      flag ← false; bestV ← 0.
1.4      para i ← 1 até q faça
1.5          se L está vazia então
1.6              retorna (LP; LN).
1.7          senão
1.8              j ← item escolhido aleatoriamente de L.
1.9              se valorI + v_j > UB OU bestV > v_j OU ∃(k, j) ∈ E para k ∈ LP então
1.10                 LN ← LN ∪ {j}.
1.11                 L ← L − {j}.
1.12             senão
1.13                 bestV ← v_j; item ← j; flag ← true.
1.14     se flag então
1.15         LP ← LP ∪ {item}.
1.16         L ← L − {item}.
1.17         valorI ← valorI + v_item.
1.18 retorna (LP; LN).
```

O Algoritmo 1 realiza no laço enquanto a avaliação de todos os itens da instância I. A cada iteração deste laço, uma lista de candidatos contendo q itens é criada. Estes itens são escolhidos aleatoriamente entre aqueles em L (lista com os itens ainda não avaliados), na linha 1.8. Então, para cada item j nesta lista de candidatos, escolhemos aqueles (laço das linhas 1.4-1.13) com o melhor valor/lucro v para formar a lista de candidatos restritos RLC. Entre os itens na RLC corrente, o item de melhor valor é inserido em LP e excluído de L (veja nas linhas $1.15 - 1.16$), enquanto os demais itens da lista de candidatos, excetuando-se aqueles na RLC corrente, são excluídos de L, por já terem sido avaliados negativamente e não contribuírem para a solução corrente (linhas $1.9 - 1.11$).

Quando L estiver vazia, o Algoritmo 1 retorna com as listas LP e LN. Note que em LP há um conjunto de itens não conflitantes (ou seja, um conjunto independente do grafo G) gerado de forma aleatória, porém com base na melhor avaliação gulosa de cada RLC. Além disso, LP contém um número de itens que não necessariamente cabe em uma única mochila B.

2.2 Busca Local e Geração de Vizinhos

De posse da solução gerada inicialmente, buscamos iterativamente melhorá-la usando um algoritmo de busca local. As operações que permitem gerar novos vizinhos são chamadas de operações de perturbação e são caracterizadas por inserções, trocas e exclusões de itens nas listas LP e LN. Listamos abaixo os tipos (forma de se obter um novo vizinho) considerados, em que cada tipo t possui probabilidade p_t de

ser escolhido.

- T_1: Se LP está vazia, chama-se T_3, senão retiramos aleatoriamente um item de LP e inserimos em LN;
- T_2: Se LP está vazia, chama-se T_3, senão escolhemos aleatoriamente $j \in LP$ e, se LN está vazia, chama-se T_1, senão fazemos a escolha aleatória de um item $k \in LN$. Então, se existir conflito entre k e qualquer outro item em $(LP - \{j\})$, retornamos sem vizinho, senão fazemos a troca de j por k.
- T_3: Se LN está vazia, retornamos sem vizinho, senão escolhemos aleatoriamente $k \in LN$. Então, se existir conflito entre k e qualquer outro item em LP, retornamos sem vizinho, senão, inserimos k em LP, retirando-o de LN.
- T_4: Se LP está vazia, chama-se T_3, senão escolhemos aleatoriamente $j \in LP$ e, se LN está vazia, chama-se T_1, senão fazemos a escolha pseudo-aleatória de um item $k \in LN$, tal que $v_k > v_j$ **E** $(\sum_{i \in LP} v_i) - v_j + v_k \leq UB$ **E** k não esteja em conflito com qualquer outro item em $(LP - \{j\})$. Se não existir tal item k, chama-se T_1, senão finalmente fazemos a troca de j por k. UB é um limitante superior para o valor na mochila.
- T_5: Se LN está vazia, chama-se T_1, senão fazemos a escolha pseudo-aleatória de um item $k \in LN$, tal que $(\sum_{i \in LP} v_i) + v_k \leq UB$ **E** k não esteja em conflito com qualquer outro item em LP. Se não existir tal item k, chama-se T_1, senão finalmente inserimos k em LP, retirando-o de LN.
- T_6: Sorteamos aleatoriamente um valor m entre 1 e a quantidade de itens em LP. Então, retiramos os m primeiros itens de LP. Em seguida, inserimos (até) m itens de LN em LP, desde que para cada item $j \in LN$ a ser inserido em LP, j não gere conflito com quaisquer dos itens presente em LP.

O Algoritmo 2 implementa a sub-rotina de busca local, a qual explora a vizinhança da melhor solução atual usando os tipos de vizinhos previamente discutidos. Com isso, esperamos obter novas listas LP (e LN) melhoradas segundo o objetivo do 2KPD.

Algoritmo 2: Sub-rotina que realiza a busca local - LSearch

Entrada: Instância $I = (B, V, G)$; listas LP e LN; número de vizinhos *nviz*; probabilidade p_t para gerar um vizinho do tipo $t = \{1,2,3,4,5,6\}$.
Saída: Listas LP e LN atualizadas.

2.1 $vA \leftarrow 0$; $vM \leftarrow \sum_{j \in LP} v_j$.
2.2 **enquanto** $vM > vA$ **faça**
2.3 **para** $i \leftarrow 1$ **até** *nviz* **faça**
2.4 Selecione de forma aleatória um tipo t observando a probabilidade p_t.
2.5 Gera um vizinho LP_{viz} sobre LP aplicando o tipo T_t.
2.6 **se** $\sum_{j \in LP_{viz}} v_j > vA$ **então**
2.7 $vA \leftarrow \sum_{j \in LP_{viz}} v_j$.
2.8 $vAux \leftarrow vA$; $vA \leftarrow vM$.
2.9 **se** $vAux > vM$ **então**
2.10 $vM \leftarrow vAux$.
2.11 $LP \leftarrow LP_{viz}$.
2.12 **retorna** *(LP; V-LP).*

Observe que o Algoritmo 2 é executado enquanto existir solução melhor que a melhor solução atual (laço das linhas $2.2 - 2.11$). A cada iteração deste laço enquanto, *nviz* vizinhos são gerados de forma

que, aquele cuja soma dos valores dos itens é máxima, será considerado como melhor solução (LP) atual (linha 2.11). Ao final, o algoritmo retorna a lista LP (possivelmente melhorada) contendo os itens não conflitantes que perfazem o maior valor entre todos os vizinhos explorados.

2.3 Empacotando Itens 2D

O algoritmo usado para empacotar os itens da lista LP na mochila é uma adaptação do algoritmo OneBin proposto por Martello et al. (2000), que chamaremos de OneBin-M. OneBin é um algoritmo exato do tipo *branch-and-bound* que encontra o melhor preenchimento para um único recipiente (mochila), dados os itens da entrada, usando o conceito de pontos de cantos.

Em linhas gerais, durante a execução do OneBin, alguns conjuntos são mantidos para limitar a ramificação da árvore e podar ramos que conduzem à soluções inviáveis ou não promissoras. Então, seja P e N os conjuntos de itens empacotados e não empacotados, respectivamente. Denotamos por $C(P)$ o conjunto de pontos de canto dado os itens em P, $A(P)$ é a área do envelope (parte não usada da mochila delimitada pelos pontos de canto e relacionada com os itens em P) e F é a área do melhor empacotamento já obtido. Assim, cada item $j \in N$ é empacotado em cada ponto de canto $c \in C(P)$, tal que o algoritmo é chamado recursivamente e os conjuntos $C(P)$ e $A(P)$ são atualizados adequadamente. Quando não couber mais itens na mochila, F pode ter seu valor atualizado e o *backtracking* ocorre. O *backtracking* também ocorre sempre que $\sum_{i \in P} a_i - (A(P) - V) \leq F$. O trabalho original de Martello et al. (2000) apresenta maiores detalhes.

O Algoritmo 3 descreve o OneBin-M. A sub-rotina OneBin tem sua execução limitada de acordo com um valor de tempo limite imposto na entrada. Quando este tempo for alcançado, a melhor solução obtida até aquele momento é reportada. Vale mencionar que usamos (e adaptamos) o código disponibilizado no *website* de um dos autores de Martello et al. (2000) para implementar o algoritmo em questão.

Algoritmo 3: OneBin-M

Entrada: Instância $I = (B, V, G)$ do 2KPD; lista LP de itens; tempo limite TM.
Saída: Empacotamento S de um subconjunto de itens em LP.

3.1 Faça $a_i = l_i * c_i$ para todo $i \in LP$.
3.2 Ordene de forma decrescente os itens em LP segundo a razão $\frac{v_i}{a_i}$, tal que $(\frac{v_1}{a_1} \geq \frac{v_2}{a_2} \geq \ldots \geq \frac{v_n}{a_n})$. Desempates são feitos observando o menor valor de a_i.
3.3 $S \leftarrow OneBin(LP, TM)$.
3.4 **retorna** (S).

Note que no Algoritmo 3 os itens são inicialmente ordenados de forma decrescente observando a razão valor por área de cada item (linha 3.2). Então, o OneBin recebe a entrada já ordenada e não faz qualquer outro tipo de ordenação. No final, o algoritmo retorna um empacotamento S de um subconjunto de itens em LP. Visto que o algoritmo OneBin pode ser extremamente lento, como reporta Martello et al. (2000), seria interessante limitar a quantidade de itens em LP de forma que $\sum_{i \in LP} (l_i * c_i) \leq L * C$.

2.4 Heurística para o 2KPD

O Algoritmo 4 apresenta o G3F, fazendo uso das sub-rotinas descritas anteriormente para cada fase.

Inicialmente, computamos um valor para o limitante superior UB (linha 4.2) que será usado durante a geração de vizinhos (linha 4.8) e como um critério de parada (linha 4.24). Para tanto, usamos a sub-rotina $knapsack1d_{value}(I)$ que: (i) transforma a instância I do 2KPD em uma instância do 1KP fazendo $W \leftarrow L * C$, $(w_i, v_i) \leftarrow (l_i * c_i, v_i)$ para todo item $i \in I$ e ignora o grafo de conflitos G; (ii) aplica o clássico algoritmo (pseudo-polinomial) de programação dinâmica que resolve o 1KP na instância gerada; e, (iii) retorna o valor da solução encontrada, como limitante superior.

Algoritmo 4: G3F

Entrada: Instância $I = (B,V,G)$ do 2KPD; tamanho q da lista de candidatos; número de vizinhos $nviz$; número de iterações $iterG$ para as fases (i)-(iii); número total $nvezes$ de iterações; tempo TM para o OneBin-M; probabilidade p_t de gerar um vizinho do tipo $t = \{1,2,3,4,5,6\}$.

Saída: Solução para I.

4.1 $vB \leftarrow 0$; $rr \leftarrow 1$; $cc \leftarrow 1$; $LP_{best} \leftarrow \{\}$; $S \leftarrow \{\}$.
4.2 $UB \leftarrow knapsack1d_{value}(I)$.
4.3 **para** $i \leftarrow 1$ **até** $nvezes$ **faça**
4.4 **para** $j \leftarrow 1$ **até** $iterG$ **faça**
4.5 **se** $rr < 4$ **então**
4.6 **se** $cc < 2$ **então**
4.7 $LP \leftarrow \{\}$; $LN \leftarrow \{\}$.
4.8 $(LP;LN) \leftarrow SBegin(I,UB,q)$.
4.9 **senão**
4.10 Diversificar LP:{Para cada $j \in LP$, sorteamos um valor vv em {1,2}. Se vv for 1, então retiramos j de LP e inserimos em LN. Caso contrário, nada é feito}.
4.11 $cc \leftarrow 1$.
4.12 **senão**
4.13 Diversificar LP: {Sorteamos vv aleatoriamente entre 1 e $\frac{n}{2}$ (n é o número total de itens em I). Para cada item k de vv até n, em I, fazemos: sorteamos um valor mm em {1,2} e, se mm for 1, então inserimos k em LN, senão, inserimos k em LP. Por fim, para cada item k em LP, verificamos se k está em conflito com qualquer outro item de LP, se sim, retiramos k de LP (e inserimos em LN), repetindo este processo para a lista LP atualizada}.
4.14 $rr \leftarrow 1$.
4.15 $(LP;LN) \leftarrow LSearch(I,LP,LN,nviz,\{p_1,p_2,p_3,p_4,p_5,p_6\})$.
4.16 $iC \leftarrow \sum_{k \in LP} v_k$.
4.17 **se** $iC = vB$ **então**
4.18 $rr \leftarrow rr+1$; $cc \leftarrow cc+1$.
4.19 **senão**
4.20 **se** $iC > vB$ **então**
4.21 $LP_{best} \leftarrow LP$; $vB \leftarrow iC$.
4.22 $LP \leftarrow knapsack1d(I,LP)$.
4.23 $S_c \leftarrow \text{OneBin} - \text{M}(LP,TM)$.
4.24 **se** $\sum_{k \in S_c} v_k = UB$ **então**
4.25 **retorna** (S_c).
4.26 **senão**
4.27 **se** $\sum_{k \in S_c} v_k > \sum_{k \in S} v_k$ **então**
4.28 $S \leftarrow S_c$.
4.29 **retorna** (S).

O laço externo, das linhas $4.3 - 4.28$, ocorre até $nvezes$ com o objetivo de obter diferentes empacotamentos para o 2KP e, então, fazer o melhor deles (máximo valor) como solução final. Para isto, o laço interno, das linhas $4.4 - 4.21$, realiza um número de iteração $iterG$ das duas primeiras fases do G3F em busca de uma lista LP que gere uma boa solução ao executar o OneBin-M. Note que antes de chamar OneBin-M, executamos a sub-rotina $knapsack1d(I,LP)$ (linha 4.22) que: (i) transforma os itens de LP em uma instância do 1KP fazendo $W \leftarrow L*C$ e $(w_i,v_i) \leftarrow (l_i*c_i,v_i)$ para todo item $i \in LP$; (ii) aplica o

clássico algoritmo que resolve o 1KP em tal instância; e, (iii) retorna os itens presentes na solução final (não o valor resultante).

Além disso, observe que duas operações de diversificação são feitas no G3F: uma na linha 4.10 e outra em 4.13. Ambas buscam diversificar os itens em LP quando a soma do valor dos itens *iC* da LP corrente alcançar duas e quatro vezes, respectivamente, o melhor valor *vB* já obtido.

3 Modelo de Programação Inteira

A formulação inteira para o 2KPD considera a mochila B discretizada em uma malha de pontos. Herz (1972) apresenta os pontos de discretização e mostra que estes podem ser usados sem perda generalidade ao considerar um problema de empacotamento 2D que, por exemplo, visa minimizar o desperdício da placa retangular. Isto torna possível estender, sem perda de generalidade, os resultados de Herz (1972) para o 2KPD. Note que as restrições de disjunção não impõem restrições quanto ao modo de empacotar os itens, mas apenas com relação à quais itens devem ser empacotados (ou não).

A partir disto, dada a malha formada com somente os pontos de discretização de Herz, denominamos por $\mathcal{C}$ o conjunto que representa as linhas na direção do comprimento (eixo y), enquanto $\mathcal{L}$ contém as linhas na direção da largura (eixo x). Por $\mathcal{S}$ representamos o conjunto dos pontos $p = (a,b)$, tal que $a \in \mathcal{L}$ e $b \in \mathcal{C}$. Já $\mathcal{S}_i$ representa o conjunto de pontos onde o item i pode ser empacotado na mochila. Por outro lado, denotamos por $\mathcal{D}_{ijp}$ o conjunto de pontos, tal que se o item j for empacotado em algum destes pontos, então j irá sobrepor o item i que foi empacotado em p. As únicas variáveis de decisão na formulação para o 2KPD são binárias, tal que $w_{ip} = 1$ indica que o item i foi empacotado no ponto p e, $w_{ip} = 0$, caso contrário. Por fim, o modelo inteiro descrito na eq. (1) formula o 2KPD.

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} \sum_{p \in \mathcal{S}_i} v_i * w_{ip} \\
\textit{sujeito a}: & \\
(i) \quad & \sum_{p \in \mathcal{S}_i} w_{ip} \leq 1 && \forall\, i = 1, \ldots, n. \\
(ii) \quad & \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{q \,\in\, \mathcal{D}_{ijp}} w_{jq} \leq (1 - w_{ip})n && \forall\, i = 1, \ldots, n;\ \forall\, p \in \mathcal{S}_i. \\
(iii) \quad & w_{ip} + w_{jq} \leq 1 && \forall\, (i,j) \in E;\ \forall\, p \in \mathcal{S}_i;\ \forall\, q \in \mathcal{S}_j. \\
(iv) \quad & w_{ip} \in \{0,1\} && \forall\, i = 1, \ldots, n;\ \forall\, p \in \mathcal{S}_i.
\end{aligned}
\tag{1}
$$

A função objetivo da formulação (1) busca maximizar o valor carregado na mochila, enquanto que as restrições: (i) garantem que cada item será empacotado no máximo uma vez; (ii) evitam a sobreposição entre quaisquer pares de itens i e j, caso o item i esteja empacotado no ponto p; (iii) satisfazem as restrições de disjunção impostas pelo grafo de conflitos G; e, (iv) garantem que as variáveis são binárias.

Note que o número de variáveis e restrições na formulação (1) é pseudo-polinomial, no pior caso, visto dependerem da quantidade de pontos na malha, que, por sua vez, depende das dimensões da mochila e dos itens. Maiores detalhes sobre como obter o pontos de discretização de Herz, ou conjuntos mais especializados de pontos, como os *reduced raster points* de Scheithauer (1997), podem ser obtidos no trabalho recente de Queiroz et al. (2012).

4 Testes Computacionais

Fizemos a codificação do G3F e das respectivas sub-rotinas na linguagem *C*. Por sua vez, o modelo de programação inteira da eq. (1) foi resolvido pelo ILOG® CPLEX® 12 Callable Library com seus

parâmetros padrões, exceto o tempo máximo de resolução que foi limitado a 7200 segundos. Mais detalhes sobre o CPLEX estão em IBM (2009). Os testes computacionais ocorreram em uma máquina com sistema operacional *Linux*, processador Intel® Xeon® X3430 2.4 GHz e 8 GB de memória RAM.

As instâncias foram geradas de forma aleatória baseando-se no procedimento descrito por Yamada et al. (2002) e Martello e Vigo (1998), visto não encontrarmos instâncias para o 2KPD na literatura. Como visamos comparar a solução do G3F com a exata, dado o limite de 7200 segundos no resolvedor, as instâncias criadas são de pequeno até médio porte. Basicamente, as instâncias seguem a seguinte formação: (i) número de itens fixado em 25; (ii) mochila B com $L = C$ e sendo $\{30,40,50\}$; (iii) densidade do grafo G sendo $\{7\%, 10\%, 13\%, 16\%\}$ do valor $\frac{n(n-1)}{2}$, sendo $n = 25$ o número de itens; e, (iv) itens de quatro tipos, que são:

- Tipo I: l_i obtido aleatoriamente em $[\frac{2}{3}L, L]$; c_i obtido aleatoriamente em $[1, \frac{1}{2}C]$;
- Tipo II: l_i obtido aleatoriamente em $[1, \frac{1}{2}L]$; c_i obtido aleatoriamente em $[\frac{2}{3}C, C]$;
- Tipo III: l_i obtido aleatoriamente em $[\frac{1}{2}L, L]$; c_i obtido aleatoriamente em $[\frac{1}{2}C, C]$;
- Tipo IV: l_i obtido aleatoriamente em $[1, \frac{1}{2}L]$; c_i obtido aleatoriamente em $[1, \frac{1}{2}C]$;

A partir disto, um total de 48 instâncias foram criadas, de forma que em cada instância há um Tipo predominante. Ou seja, a probabilidade de itens daquele tipo aparecer é de 70%, enquanto que dos demais é de 10%. Além disso, as instâncias estão disponíveis através de solicitação por *e-mail*.

Os parâmetros usados no G3F foram determinados através de testes de calibração, sendo os mais promissores usados para a obtenção dos resultados aqui presentes. Nestes testes, buscamos por valores de parâmetros que balanceassem a exploração do espaço de busca com a qualidade da solução e o tempo computacional requerido. Com isso, o valor dos parâmetros usados foram: nvezes = 50; iterG = 20; TM = 10 (segundos); q = 20% do número de itens em I; nviz = 20; $p_1 = 5\%$; $p_2 = 20\%$; $p_3 = 20\%$; $p_4 = 10\%$; $p_5 = 25\%$; $p_6 = 20\%$. Vale mencionar que valores maiores para os números de iterações e tempo TM foram testados, porém não ocorreram melhoras na solução final, apenas o tempo resultante foi maior.

A Tabela 1 mostra os resultados obtidos pela heurística G3F e pelo CPLEX para cada instância considerada. Cada linha da tabela apresenta: nome da instância; dimensões da mochila B; tipo predominante de itens; densidade do grafo G; resultado do G3F: tempo médio sobre as *nvezes* iterações, tempo total e o valor da melhor solução; resultado do resolvedor: tempo gasto, valor da solução e se a solução é a ótima. Caso não seja ótima, o valor apresentado corresponde ao valor da melhor solução viável obtida dentro do tempo limite imposto. Por outro lado, se aparecer a entrada "–", significa que o resolvedor não encontrou qualquer solução viável dentro deste tempo limite.

Analisando os dados presentes na Tabela 1, o tempo médio e o total requerido pelo G3F foi de $0,27$ e $13,72$ segundos, na média. O resolvedor, por sua vez, consumiu $5387,37$ segundos, na média, representando um diferença enorme ao se comparar com o tempo total (na média) do G3F. Observe que algumas instâncias do G3F tiveram tempo computacional elevado, como a $2kpd20$. Isto aconteceu pois o OneBin-M parou somente ao atingir o tempo limite em quase todas as iterações. Vale mencionar que o algoritmo OneBin-M comportou-se bem, já que atingiu o tempo limite de 10 segundos para poucas instâncias (note que o tempo total do G3F foi bem abaixo de 10 segundos para 37 das 48 instâncias).

Por outro lado, o resolvedor retornou solução ótima, dado o tempo limite de 7200 segundos, para 15 das 48 instâncias. Uma justificativa para o baixo número de soluções ótimas dentro deste tempo está no tipo dos itens presente na instância. Isto é, se as dimensões dos itens forem pequenas comparadas a da mochila, o número de pontos na malha cresce rapidamente visto que a formulação (1) tem uma quantidade pseudo-polinomial de variáveis e restrições.

Note que o G3F não conseguiu solução ótima para qualquer das instâncias resolvidas à otimalidade pelo CPLEX. No entanto, obteve soluções satisfatórias, com diferença percentual de $22,99\%$, na média,

Tabela 1: Resultados para as instâncias geradas aleatoriamente.

nome	(L, C)	tipo	dens.	G3F			resolvedor CPLEX		
				T. Médio (s)	T. Total (s)	M. Valor	T. Total (s)	Valor	Ótima?
2kdc1	(30, 30)	I	0,07	0,07	3,33	4798	7200,00	6203	NÃO
2kdc2	(30, 30)	II	0,07	0,60	30,09	4638	7200,00	5568	NÃO
2kdc3	(30, 30)	III	0,07	0,00	0,08	2256	6,77	2354	SIM
2kdc4	(30, 30)	IV	0,07	0,21	10,37	6085	7200,00	–	NÃO
2kdc5	(30, 30)	I	0,10	0,02	0,90	3651	7200,00	4245	NÃO
2kdc6	(30, 30)	II	0,10	0,00	0,10	2893	7200,00	4516	NÃO
2kdc7	(30, 30)	III	0,10	0,00	0,03	2298	395,73	2793	SIM
2kdc8	(30, 30)	IV	0,10	0,20	10,09	5373	7200,00	–	NÃO
2kdc9	(30, 30)	I	0,13	0,03	1,36	4401	7200,00	4577	NÃO
2kdc10	(30, 30)	II	0,13	0,01	0,37	3502	7200,00	3826	NÃO
2kdc11	(30, 30)	III	0,13	0,00	0,13	1978	115,06	3764	SIM
2kdc12	(30, 30)	IV	0,13	0,00	0,13	4159	7200,00	7108	NÃO
2kdc13	(30, 30)	I	0,16	0,00	0,08	3935	3346,21	4649	SIM
2kdc14	(30, 30)	II	0,16	0,00	0,08	3325	7200,00	4754	NÃO
2kdc15	(30, 30)	III	0,16	0,00	0,04	1691	14,25	3350	SIM
2kdc16	(30, 30)	IV	0,16	0,80	40,09	4928	7200,00	–	NÃO
2kdc17	(40, 40)	I	0,07	0,00	0,05	5119	7200,00	8138	NÃO
2kdc18	(40, 40)	II	0,07	0,00	0,07	5579	1428,18	6923	SIM
2kdc19	(40, 40)	III	0,07	0,00	0,08	4342	124,31	6560	SIM
2kdc20	(40, 40)	IV	0,07	8,60	430,02	9118	7200,00	–	NÃO
2kdc21	(40, 40)	I	0,10	0,00	0,07	5972	4268,85	11209	SIM
2kdc22	(40, 40)	II	0,10	0,00	0,23	5160	1621,27	6199	SIM
2kdc23	(40, 40)	III	0,10	0,00	0,04	3033	28,96	4207	SIM
2kdc24	(40, 40)	IV	0,10	0,00	0,11	7149	7200,00	–	NÃO
2kdc25	(40, 40)	I	0,13	0,00	0,08	7596	7200,00	9552	NÃO
2kdc26	(40, 40)	II	0,13	0,00	0,23	7045	7200,00	7045	NÃO
2kdc27	(40, 40)	III	0,13	0,00	0,09	2849	43,22	4555	SIM
2kdc28	(40, 40)	IV	0,13	0,00	0,12	8269	7200,00	–	NÃO
2kdc29	(40, 40)	I	0,16	0,01	0,42	4600	7200,00	5677	NÃO
2kdc30	(40, 40)	II	0,16	0,20	10,16	5019	7200,00	8402	NÃO
2kdc31	(40, 40)	III	0,16	0,00	0,05	4144	7200,00	–	NÃO
2kdc32	(40, 40)	IV	0,16	0,60	30,11	6847	7200,00	–	NÃO
2kdc33	(50, 50)	I	0,07	0,00	0,07	15426	7200,00	15516	NÃO
2kdc34	(50, 50)	II	0,07	0,17	8,42	12447	7200,00	–	NÃO
2kdc35	(50, 50)	III	0,07	0,00	0,07	6221	2793,14	7545	SIM
2kdc36	(50, 50)	IV	0,07	0,20	10,06	15893	7200,00	–	NÃO
2kdc37	(50, 50)	I	0,10	0,00	0,09	10688	5814,96	13512	SIM
2kdc38	(50, 50)	II	0,10	0,00	0,11	11423	7200,00	16011	NÃO
2kdc39	(50, 50)	III	0,10	0,00	0,12	5219	50,15	6647	SIM
2kdc40	(50, 50)	IV	0,10	0,40	20,11	14436	7200,00	–	NÃO
2kdc41	(50, 50)	I	0,13	0,00	0,12	8563	7200,00	9529	NÃO
2kdc42	(50, 50)	II	0,13	0,00	0,12	8175	7200,00	–	NÃO
2kdc43	(50, 50)	III	0,13	0,00	0,16	8713	942,66	9387	SIM
2kdc44	(50, 50)	IV	0,13	0,60	30,11	13716	7200,00	–	NÃO
2kdc45	(50, 50)	I	0,16	0,00	0,16	7821	7200,00	–	NÃO
2kdc46	(50, 50)	II	0,16	0,39	19,30	12154	7200,00	–	NÃO
2kdc47	(50, 50)	III	0,16	0,00	0,07	6176	7200,00	–	NÃO
2kdc48	(50, 50)	IV	0,16	0,00	0,08	10030	7200,00	–	NÃO

comparado ao CPLEX, considerando todas as instâncias com valor de solução, inclusive as não resolvidas à otimalidade. O G3F obteve solução igual à retornada pelo CPLEX (não provada ótima) apenas para a instância $2kdc26$. A pior solução do G3F ocorreu para a instância $2kpc15$ cuja diferença no valor da solução comparado ao valor ótimo foi de $49,52\%$. Apesar disto, a heurística G3F tem potencial para resolver o 2KPD ao considerar instâncias de grande porte, ao contrário do CPLEX para a formulação dada. Além disso, os resultados são promissores para alicercear investigações futuras, ainda mais pela carência de estratégias para problema em questão.

Acreditamos que a combinação do G3F com o resolvedor permitiria a resolução de instâncias maiores por este último dentro do tempo limite imposto. A ideia é usar a solução obtida pelo G3F como ponto de partida para o CPLEX. Além disso, relaxar algumas restrições de disjunção e adicioná-las através de planos de corte. Também, acreditamos que a combinação do G3F com uma tabela para evitar configurações repetidas, assim como feito em Hifi e Michrafy (2006), permitiria explorar ainda mais o espaço de soluções, consequentemente retornando soluções melhores. Outro fator decisivo seria deixar de manipular listas (LP e LN) e manipular o empacotamento diretamente, ou seja, gerando também as coordenadas dos itens dentro do empacotamento nas fases (i) e (ii).

5 Conclusões e Direções para Trabalhos Futuros

Neste trabalho, a versão bidimensional do problema da mochila 0-1 com restrições de disjunção (ou com grafo de conflitos) foi tratada. Foi proposta uma heurística baseada em GRASP e uma formulação inteira para resolver o problema em questão.

A heurística possui três fases principais que incluem: gerar uma solução inicial (lista com itens não conflitantes), melhorar iterativamente esta solução, pela exploração de sua vizinhança, e realizar o empacotamento dos itens da lista final por um algoritmo que trabalha sobre pontos de canto. Já a formulação inteira considera uma malha de pontos discretizada sobre os pontos de discretização de Herz (1972).

Os resultados computacionais mostraram que a heurística proposta é satisfatória, já que obteve soluções distando $22,99\%$, na média, do valor ótimo e consumindo apenas $13,72$ segundos, na média. A formulação comportou-se bem, permitindo resolver 15 das 48 instâncias à otimalidade dentro do tempo limite de 7200 segundos.

Por fim, as estratégias propostas são consistentes para o 2KPD, apesar que melhorias podem ser feitas. Uma direção de estudo futuro inclui aperfeiçoar a formulação inteira com a inserção de desigualdades válidas e desenvolver estratégias exatas com bons limitantes para resolver instâncias de médio a grande porte dentro de um tempo computacional aceitável.

Agradecimentos

Os autores agradecem o apoio financeiro recebido pelo CNPq.

Referências

H. Akeb, M. Hifi e M. E. O. A. Mounir. (2011), Local branching-based algorithms for the disjunctively constrained knapsack problem. *Computers & Industrial Engineering*, 60:811–820.

E. G. Birgin, R. D. Lobato e R. Morabito. (2012), Generating unconstrained two-dimensional non-guillotine cutting patterns by a recursive partitioning algorithm. *Journal of the Operational Research Society*, 63(2):183–200.

L. Epstein, A. Levin e R. Van Stee. (2008), Two-dimensional packing with conflicts. *Acta Informatica*, 45(3):155–75.

M. R. Garey e D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. San Francisco: Freeman, 1979.

M. Gendreau, G. Laporte e F. Semet. (2004), Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research*, 31:347–358.

J. C. Herz. (1972), A recursive computational procedure for two-dimensional stock-cutting. *IBM Journal of Research Development*, páginas 462–469.

M. Hifi e M. Michrafy. (2006), A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society*, 57:718–726.

M. Hifi e M. Michrafy. (2007), Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & Operations Research*, 34:2657–2673.

M. Hifi e N. Otmani. (2012), An algorithm for the disjunctively constrained knapsack problem. *International Journal of Operational Research*, 13:22–43.

IBM. *ILOG® CPLEX® V12.1 - User's Manual for CPLEX®*. IBM Corporation, 2009.

K. Jansen e S. Öhring. (1997), Approximation algorithms for time constrained scheduling. *Information and Computation*, 132(2):85–108.

A. Khanafer, F. Clautiaux e E-G. Talbi. (2010), New lower bounds for bin packing problems with conflicts. *European Journal of Operational Research*, 206(2):281–288.

A. Khanafer, F. Clautiaux e E-G. Talbi. (2012), Tree-decomposition based heuristics for the two-dimensional bin packing problem with conflicts. *Computers & Operations Research*, 39:54–63.

S. Martello e D. Vigo. (1998), Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399.

S. Martello, D. Pisinger e D. Vigo. (2000), The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267.

A. E. F. Muritiba, M. Iori, E. Malaguti e P. Toth. (2010), New lower bounds for bin packing problems with conflicts. *INFORMS Journal on Computing*, 22(3):401–415.

U. Pferschy e J. Schauer. (2009), The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications*, 13(2):233–249.

T. A. Queiroz, F.K. Miyazawa, Y. Wakabayashi e E.C. Xavier. (2012), Algorithms for 3D guillotine cutting problems: unbounded knapsack, cutting stock and strip packing. *Computers & Operations Research*, 39:200–212.

R. Sadykov e F. Vanderbeck. (2012), Bin packing with conflicts: A generic branch-and-price algorithm. *INFORMS Journal on Computing*, doi: 10.1287/ijoc.1120.0499.

G. Scheithauer. (1997), Equivalence and dominance for problems of optimal packing of rectangles. *Ricerca Operativa*, 27:3–34.

T. Yamada e S. Kataoka. (2001), Heuristic and exact algorithms for the disjunctively constrained knapsack problem. In *Presented at EURO 2001: Rotterdam, The Netherlands*.

T. Yamada, S. Kataoka e K. Watanabe. (2002), Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Information Processing Society of Japan Journal*, 43:2864–2870.