MÉTODOS EXATOS PARA O PROBLEMA DE EMPACOTAMENTO COM DEPENDÊNCIAS

Renatha Capua
Instituto de Computação - Universidade Federal Fluminense - UFF
R. Passo da Pátria, 156, São Domingos, Niterói-RJ, 24210-240, Brasil
rcapua@ic.uff.br

Ruslan Sadykov
INRIA Bordeaux – Sud-Ouest
200 avenue de la Veille Tour, 33405 Talence, França
ruslan.sadykov@inria.fr

Thibaut Vidal
Departamento de Informática - Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio
R. Marquês de São Vicente, 225, Gávea, Rio de Janeiro - RJ, 22451-900, Brasil
vidalt@inf.puc-rio.br

Yuri Frota
Instituto de Computação - Universidade Federal Fluminense - UFF
R. Passo da Pátria, 156, São Domingos, Niterói-RJ, 24210-240, Brasil
yuri@ic.uff.br

Luiz Satoru Ochi
Instituto de Computação - Universidade Federal Fluminense - UFF
R. Passo da Pátria, 156, São Domingos, Niterói-RJ, 24210-240, Brasil
satoru@ic.uff.br

RESUMO

Este artigo apresenta o Problema de Empacotamento com Dependências, que surge na otimização da alocação de tarefas a CPUs em aplicações de computação em nuvem. Para resolver o problema são propostos uma formulação compacta com desigualdades para eliminação de simetria, e um algoritmo de *Branch-and-price* que utiliza o resolvedor BaPCode. Para acelerar a resolução dos problemas de *pricing*, foi desenvolvida uma heurística de *pricing* baseada no algoritmo de busca em profundidade. Os experimentos computacionais em 40 instâncias, com itens variando de 120 a 250, demonstram a alta performance da abordagem de *Branch-and-price*, que resolve todas as instâncias em até 37 minutos. Já a resolução da formulação matemática, utilizando-se o resolvedor CPLEX, produz apenas 10 soluções ótimas em um tempo limite de uma hora.

PALAVRAS CHAVE. Problema de Empacotamento com Dependências. Heurística de Precificação. Branch-and-price.

Área Principal: Programação Matemática

ABSTRACT

In this paper, we introduce the Bin Packing Problem with Dependencies, which appears when optimizing task allocations to CPUs in cloud computing applications. We propose a compact

formulation of the problem with symmetry breaking inequalities, and a branch-and-price algorithm built in the BaPCode framework. To speed up the resolution of the *pricing* problems, we design a specialized *pricing* heuristic based on a depth-first search. Our computational experiments on 40 benchmark instances with 120 to 250 items demonstrate the high performance of the branch-and-price approach, which solves all instances in up to 37 minutes. In comparison, the direct resolution of the compact formulation, with CPLEX, led only to 10 optimal solutions with a time limit of one hour.

KEYWORDS. Bin Packing Problem with Dependencies. Pricing Heuristic. Branch-and-price.

Main Area: Mathematical Programming

1. Introdução

O *Problema de Empacotamento (PE)* está entre os problemas mais estudados na área de Otimização Combinatória. A grande quantidade de trabalhos existentes na literatura que tratam deste problema demonstram a relevância de se obter soluções eficientes para sua resolução. Uma revisão detalhada dos métodos pode ser vista em [Delorme et al., 2016]. Dentre os fatores que levaram a isso, está principalmente a simplicidade da definição do problema, de fácil entendimento, e as diversas aplicações práticas em que pode ser utilizado.

Dependendo da aplicação em que está sendo empregado, o Problema de Empacotamento possui restrições adicionais que devem ser incorporadas ao problema clássico e que trazem determinadas características que dificultam sua resolução utilizando os métodos existentes. Esse fator motiva o continuo estudo de novas técnicas que possam contribuir na resolução dessas outras variantes do Problema de Empacotamento. Neste contexto, este trabalho aborda uma variante do Problema de Empacotamento chamada de *Problema de Empacotamento com Dependências (PED)*.

O Problema de Empacotamento com Dependências é um problema novo mas que é relacionado com três outros problemas encontrados na literatura. Assim como no Problema de Empacotamento clássico, neste problema considera-se um conjunto de itens e *bins* idênticos, onde todos os itens devem ser alocados ao menor número possível de *bins*. A diferença é que no PED, cada item pode estar associado a uma ou mais vizinhanças (um conjunto de outros itens). A alocação de um item a um *bin* implica em alocar junto no mesmo *bin* ao menos um outro item pertencente a cada uma de suas vizinhanças, caso existam.

O PED possui uma estrutura diferente do Problema de Empacotamento, de modo que uma simples realocação de um item em outro *bin* necessita de realocações simultâneas de possivelmente muitos outros itens em conjunto. Sem a inclusão de novas vizinhanças avançadas baseadas em cadeias de realocações de itens inter-dependentes, esta tarefa pode tornar-se difícil e computacionalmente custosa em uma abordagem baseada em metaheurísticas, que utilizam estruturas de vizinhanças. Deste modo, para produzir resultados iniciais para o PED, considera-se o uso de resolvedores de programação inteira, tais como o CPLEX para resolver a formulação compacta desenvolvida para o problema. Porém, tal formulação apresenta simetria, com diversas restrições equivalentes, e sua relaxação linear é fraca. Como uma alternativa a esses métodos e, beneficiando-se do desenvolvimento recente de *frameworks Branch-and-price*, considera-se ainda o desenvolvimento de métodos de programação matemática no qual as dependências podem ser mais intuitivamente integradas com o subproblema de precificação ou *pricing*.

Portanto, neste trabalho, para resolver o PED são utilizados métodos exatos de programação inteira, mais especificamente, o algoritmo de *Branch-and-Price*, que utiliza uma implementação chamada BaPCod, proposta por Sadykov e Vanderbeck [2013]. São apresentados ainda, uma formulação matemática para o problema e novas instâncias utilizadas nos testes.

O artigo é organizado como descrito a seguir. Na Seção 2 é feita uma definição formal do problema, onde é apresentado um modelo matemático de programação inteira desenvolvido. Na Seção 3 são revisados alguns trabalhos relacionados existentes na literatura. Na Seção 4 é descrito um método exato para a resolução do PED e a seguir, na Seção 6, são exibidos os resultados computacionais obtidos com o algoritmo desenvolvido. Finalmente, na Seção 7 são apresentadas as conclusões.

2. Definição do Problema

No Problema de Empacotamento com Dependências (PED), considere um conjunto finito de *bins* de capacidade idêntica Q, um conjunto de itens $\mathcal{V} = \{1, \ldots, n\}$ com pesos $w_1, \ldots, w_n$. Cada item $i \in \mathcal{V}$ possui um conjunto de vizinhanças $N(i) = \{N_1(i), \ldots, N_{L_i}(i)\}$, onde cada vizinhança é composta por um conjunto de itens.

O objetivo do Problema de Empacotamento com Dependências é alocar cada item a um *bin* minimizando o número de *bins* utilizados. Esta alocação deve respeitar a capacidade máxima dos *bins* e as restrições de vizinhanças, onde uma solução é viável se, para cada item i, e para cada

vizinhança $N_l(i)$, existe pelo menos um item $j \in N_l(i)$ alocado ao mesmo *bin* que em que i está alocado.

Um modelo de programação linear inteira mista para o problema (PLIM) é dado pela Formulação $F1$ a seguir:

$$
\begin{aligned}
(F1) \quad & \min \sum_{k=1}^{m} y_k && && (1)\\
& \text{s.a.} \sum_{i=1}^{n} w_i x_{ik} \leqslant Q y_k && k = 1, \ldots, m && (2)\\
& \sum_{k=1}^{m} x_{ik} = 1 && i = 1, \ldots, n && (3)\\
& x_{ik} \leqslant \sum_{j \in N_l(i)} x_{jk} && \forall i \in 1, \ldots, n, \forall k \in 1, \ldots, m, \forall l \in 1, \ldots, L_i && (4)\\
& y_k \in \{0, 1\} && k = 1, \ldots, n && (5)\\
& x_{ik} \in \{0, 1\} && i = 1, \ldots, n, k = 1, \ldots, m && (6)
\end{aligned}
$$

Neste modelo, a cada variável binária x_{ik} é atribuído o valor 1, se e somente se, o item i é alocado ao *bin* k, e cada variável binária y_k recebe o valor 1, se e somente se, o *bin* k está sendo utilizado. A função objetivo (1) minimiza a soma do número de *bins* utilizados. As restrições (2) garantem que o número de itens alocados a cada *bin* não ultrapasse sua capacidade máxima. O conjunto de restrições (3) asseguram que cada item deve ser atribuído a somente um *bin*. As restrições (4) garantem que pelo menos um vizinho de cada vizinhança de um item deve ser alocado ao mesmo *bin* em que este foi alocado. As restrições (5) e (6) definem o domínio das variáveis de decisão do problema.

Em $F1$, existe uma grande quantidade de simetria, uma vez que existem diversas soluções nas quais o conjunto de itens estão alocados exatamente com a mesma configuração mas em *bins* diferentes. Para acelerar o tempo de computação e melhorar o comportamento do modelo matemático, reduzindo por exemplo a simetria, foram adicionadas os seguintes cortes no modelo:

$$
\begin{aligned}
& y_k \geqslant y_{k+1} && \forall k = 1, \ldots, m-1 && (7)\\
& x_{ik} = 0 && \forall i = 1, \ldots, m-1, \forall k = i+1, \ldots, m && (8)\\
& x_{ik} \leqslant y_k && \forall k = 1, \ldots, m, \forall i = 1, \ldots, n && (9)
\end{aligned}
$$

O conjunto de desigualdades em (7) foram adicionadas para reduzir o tamanho da árvore de enumeração do *Branch-and-bound*. Nestes cortes, os *bins* são utilizados de acordo com a ordem de seus índices. O conjunto de desigualdades em (8) elimina diversas soluções simétricas considerando que há uma solução ótima onde o item 1 foi alocado ao *bin* 1, o item 2, ao *bin* 1 ou 2, e assim sucessivamente. Para melhorar a relaxação linear, foi adicionado o conjunto de restrições (9), que determinam que itens com valores inteiros não podem ser alocados a *bins* com valores fracionários. Essas desigualdades já são bem conhecidas e utilizadas para o Problema de Empacotamento [Delorme et al., 2016].

3. Revisão da Literatura

O PED é um problema novo, no entanto existem três problemas relacionados a ele, o *Problema da Mochila 1-vizinho* (*1-neighbour knapsack problem*), o *Problema da Mochila Todos-vizinhos* (*all-neighbour knapsack problem*) e o *Problema de Realocação de Máquinas*.

O Problema da Mochila 1-vizinho e o Problema da Mochila Todos-vizinhos pertencem a uma classe de problemas chamados de Problema da Mochila Limitado. Em ambos, existe uma mochila que suporta um peso máximo k, um grafo $G = (V, E)$, onde cada vértice $i \in V$ representa um objeto ou item que pode ser levado na mochila, com peso w_i e um lucro p_i. O objetivo de cada um dos problemas é encontrar um conjunto de vértices que maximizam o lucro obtido e que juntos possuem um peso máximo k. No problema da Mochila 1-vizinho, se um item possui um vizinho, ele pode ser selecionado somente se pelo menos um dos seus vizinhos também for selecionado. Já no Problema da Mochila Todos-vizinhos, quando um item é selecionado, todos os seus vizinhos também são selecionados.

Problemas da Mochila limitados possuem aplicações em diversas áreas tais como escalonamento, gerenciamento de ferramentas, estratégias de investimento, armazenamento de banco de dados e *network formation* [Borradaile et al., 2012].

No trabalho de [Borradaile et al., 2012], os autores desenvolveram um algoritmo polinomial para resolver o Problema da Mochila 1-vizinho considerando os valores do pesos e do lucro uniformes e o grafo não-direcionado. Os autores mostram que o problema é fortemente NP-difícil quando esse é direcionado e uniforme. Este trabalho apresenta ainda algoritmos aproximados para o problema com peso e lucro arbitrários com grafos direcionados e não-direcionados.

Para o Problema da Mochila Todos-vizinhos, [Borradaile et al., 2012] demonstram que o problema direcionado e uniforme é fortemente NP-completo. Para o problema uniforme e direcionado, os autores apresentam uma redução para o Problema da Mochila 0-1, que é resolvido em tempo polinomial.

O Problema de Realocação de Máquinas (PRM) foi proposto pelo Google como um desafio no ROADEF/EURO [Roadef, 2012]. O objetivo do PRM é otimizar a utilização de um conjunto de máquinas heterogêneas M por um conjunto de processadores P. Inicialmente cada processo $p \in P$ está alocado a uma determinada máquina mas eles podem ser movidos entre as máquinas. Esse movimento deve respeitar restrições de capacidade, dependência e conflito. O problema considera também um conjunto de recursos R onde um processo $p \in P$ pode requerer $R_{p,r}$ unidades de cada recurso $r \in R$. A capacidade de uma máquina $m \in M$ com relação a um recurso $r \in R$ é chamado $C_{m,d}$. No PRM, a utilização de um recurso em uma máquina não pode exceder sua capacidade. A utilização U de um recurso r em uma máquina m é definida como $U(m, r) = \sum_{p \in P | M(p)=m} R(p, r)$, onde $M(p) = m$ significa que o processo p está executando na máquina m. Os processos são particionados em um conjunto s de serviços, e cada processo pertence somente a um serviço $s \in S$ e todos os processos do mesmo serviço deve executar em máquinas diferentes. Seja L, o conjunto de localidades, onde cada localidade $l \in L$ é um conjunto de máquinas, o número de localidades distintas onde os processos pertencentes ao mesmo serviço devem executar devem respeitar uma dada restrição de espalhamento. Adicionalmente, as máquinas são agrupadas também em vizinhanças denotadas por N. Um serviço pode depender de outros serviços, portanto, se um serviço s executar em uma maquina na vizinhança $n \in N$, pelo menos um processo de cada serviço pertence ao mesmo conjunto de serviços de que s depende, deve executar em uma máquina na vizinhança de n.

Vários trabalhos na literatura tratam o PRM: no trabalho de Masson et al. [2013], os autores desenvolvem uma formulação matemática e um algoritmo baseado na metaheurística *Multi-start Iterated Local Search* para o PRM. Gavranović et al. [2012] propuseram limites inferiores para o problema e uma abordagem baseada no *Variable Neighborhood Search*, onde os autores tentam realocar os processos grandes primeiro. O procedimento de perturbação do VNS de Gavranović et al. [2012] é feito por uma busca local considerando uma função objetivo diferente. Lopes et al. [2015] desenvolveram quatro heurísticas ILS, onde duas delas usam uma perturbação baseada em programação inteira. No trabalho de Portal et al. [2015], os autores propuseram uma heurística baseada no metaheurística *Simulated Annealing*, bem como desenvolveram estruturas de dados especiais para acelerar o tempo de execução. Mais recentemente, Wang et al. [2016] desenvolveram

um método que usa diferentes estruturas de vizinhanças, duas deles já utilizadas na literatura para o Problema de Alocação Generalizado, permitindo movimentos inviáveis na busca local e um mecanismo de busca de particionamento da vizinhança.

O Problema de Empacotamento com Dependências pode ser visto como um caso específico do PRM considerando apenas a restrição de dependência. O PED também está intimamente relacionado com as variantes do Problema da Mochila citadas, a principal diferença está na função objetivo que envolve minimizar o número de *bins* ao invés de maximizar o benefício obtido de um único *bin* (mochila), e onde ainda todos os itens devem ser alocados ao invés de somente um subgrupo.

4. Método de Resolução exato para o PED

Mesmo com a adição dos cortes, o modelo $F1$ não pode ser resolvido de maneira eficiente com os resolvedores de programação matemática para diversas instâncias do PED. Portanto, adicionalmente a $F1$, outra formulação alternativa para o PED pode ser obtida. Tal formulação, chamada de $F2$, é baseada na formulação do problema de cobertura de conjuntos e pode ser vista nas Equações de (10) a (12) a seguir.

$$(F2) \qquad \min \quad \sum_{j \in D} \lambda_j \tag{10}$$

$$\text{s.a.} \quad \sum_{j \in D} a_{ij} \lambda_j \geqslant 1 \qquad \forall i \in \mathcal{V} \tag{11}$$

$$\lambda_j \in \{0, 1\} \qquad \forall j \in D \tag{12}$$

Considere D um conjunto composto por todos os sub-conjuntos de itens que podem ser alocados em um *bin* e que respeitam a definição de dependência entre vizinhanças do problema. A função objetivo em (10) minimiza a soma do número de conjuntos selecionados, ou seja, o número de *bins* que serão utilizados. O conjunto de restrições em (11) definem que cada item i deve estar associado a pelo menos um item que estará na solução. Cada variável binária λ_j será igual a 1 se o subconjunto de itens j é selecionado para fazer parte da solução e 0, caso contrário. Associado a cada variável tem-se ainda um a_{ij}, que será igual a 1 se o item i está no sub-conjunto j.

Tal formulação acima envolve um número exponencial de variáveis λ. No entanto, ela possui uma alternativa para sua resolução, que é a utilização do algoritmo de *Branch-and-price* para resolvê-la. Esse algoritmo iterativamente resolve esta formulação (chamada de Problema Mestre) em um conjunto restrito de variáveis (colunas), resolvendo o subproblema para encontrar novas colunas que deverão fazer parte do problema mestre (problema de *pricing*). Essa técnica é muito conhecida e com grande sucesso na resolução de diversos problemas. A seguir será fornecida uma descrição de como esse procedimento funciona e logo após, como essa abordagem foi aplicada ao PED.

5. Algoritmo de *Branch-and-price*

Para resolver o PED, foi desenvolvido um algoritmo de *Branch-and-price (BP)* [Barnhart et al., 1998]. O BP é uma técnica que combina o algoritmo de *Branch-and-bound* com o método de geração de colunas para resolver problemas com uma grande quantidade de variáveis.

Parte-se de uma formulação do problema com um número exponencial de variáveis. Se esta não estiver disponível, pode por exemplo ser obtida aplicando-se a decomposição de Dantzig-Wolfe [Dantzig e Wolfe, 1960]. O problema é subdividido em outros problemas, um chamado de Problema Mestre e outro chamado de Subproblema, que por sua vez pode ser composto por um ou mais problemas escravos. Devido a grande quantidade de colunas presente no Problema Mestre, algumas podem ser excluídas deste, resultando no que chamamos de *Problema Mestre Reduzido (PMR)*. O BP irá resolver o PMR a cada nó da árvore de *Branch-and-bound* através de geração de

colunas. Iterativamente, através da resolução da Relaxação linear do PMR, novos valores duais para as variáveis do PMR são encontradas, esses valores são utilizados para resolver o que chamamos de problema de *pricing*: o subproblema é resolvido a fim de encontrar colunas com custo reduzido negativo. Se estas forem encontradas, elas são adicionadas ao PMR. O processo termina se a solução ótima para o PMR for uma solução ótima também para o Problema Mestre.

No caso do *Branch-and-price* desenvolvido para o PED, a decomposição de Dantzig-Wolfe é aplicada a formulação $F1$ e, assim como no problema de empacotamento clássico, resulta no problema de cobertura de conjuntos como problema mestre [Sadykov e Vanderbeck, 2013]. A formulação resultante para o problema mestre é a mesma já apresentada em $F2$. O subproblema obtido, de modo análogo, é o *Problema da Mochila com Dependências (PMD)*, que pode ser formulado por $F3$ como:

$$(F3) \quad \max \sum_{i=1}^{n} \Phi_i x'_i \tag{13}$$

$$\text{s.a.} \sum_{i=1}^{n} w_i x'_i \leqslant Q \tag{14}$$

$$x'_i \leqslant \sum_{j \in N_l(i)} x'_j \qquad \forall i \in 1,\ldots,n, \forall l \in 1,\ldots,L_i \tag{15}$$

$$x'_i \in \{0,1\} \qquad i = 1,\ldots,n \tag{16}$$

Para o problema de *pricing*, considere Φ_i, onde $i \in V$, como a solução dual corrente do PMR de $F2$.

Para resolver o problema de *pricing*, neste trabalho foi desenvolvido um algoritmo heurístico. Se este algoritmo falhar em encontrar colunas com custo reduzido, o subproblema é resolvido de modo exato. Os dois métodos são detalhados nas próximas seções.

5.1. Pricing usando heurística

Foi proposto um algoritmo heurístico para resolução do problema de *pricing*. Essa abordagem utiliza o algoritmo de Busca em Profundidade (ou *Depth First Search (DFS)*) para auxiliar na escolha dos itens que devem ser incluídos na mochila. O pseudo-código é apresentado nos Algoritmos 1 e 2.

No Algoritmo 1, considere como entrada o conjunto D de itens. Cada item possui um status que indica se o item foi fixado com 1, com 0 ou se é livre. Seja V, o conjunto de todos os itens, w_i o peso de cada item, $Q_{restante}$ o espaço disponível na mochila e, S o conjunto de itens já alocados na mochila.

Algorithm 1 Heurística para o *Pricing*

```
1:  while (Q_restante > 0) e (|S| < |V|) do
2:    Escolha um item i ∈ V − S;
3:    if w_i < Q_restante then
4:      DFS(i,Q_restante,G,S);
5:      S = S + i;
6:      if DFS encontrou um nó folha, um ciclo ou item já adicionado then
7:        Adiciona itens da árvore de DFS;
8:      end if
9:    end if
10: end while
```

Algorithm 2 DFS(i,$Q_{restante}$,G,S)

Marca i como visitado;
$pre(i) + +$;
if ($w_i < Q_{restante}$) e ($i.status \neq 0$) **then**
  if $Adj(i) = \emptyset$ **then**
    encontrou um nó folha;
  else
    Ordena $Adj(i)$ pelos custos reduzidos;
    for $v \in Adj(i)$ **do**
      if $v \in S$ **then**
        encontrou um item já adicionado na mochila;
        break;
      else
        if v não foi visitado **then**
          Insere aresta (i, v) na árvore de DFS;
          DFS(v,$Q_{restante}$,G,S);
        else
          if $pos(v)$ indefinido **then**
            encontrou um ciclo;
            break;
          end if
        end if
      end if
    end for
  end if
end if
$pos(i) + +$;

Enquanto a mochila não está cheia, o algoritmo tenta inserir itens na mochila. Inicialmente, na linha 2, um item i é escolhido para ser alocado na mochila. Essa escolha é realizada de forma gulosa, de acordo com o status do item, o número de dependências que o item resolve se alocado na mochila e o seu benefício. O benefício é calculado pelo valor do custo reduzido do item dividido pelo seu peso. Se o há espaço para alocar o item na mochila, o mesmo é alocado. Além disso, usa-se o algoritmo de DFS (linha 4), para alocar outros itens juntos na mochila e satisfazer as restrições de dependências.

O algoritmo de DFS (Algoritmo 2) percorre o grafo examinando seus nós, o item i é considerado como o nó raiz. A busca pela árvore tem por objetivo encontrar um caminho na árvore que forme um ciclo que inclua o item i (linha 18) ou outro item já adicionado na mochila (linha 10), ou um nó folha (linha 5), que não possua nós dependentes.

5.2. Pricing usando programação inteira

Para a resolução do problema de *pricing* uma opção mais direta e simples é utilizar o modelo matemático existente para o problema da mochila com dependências. Isto é feito com o uso de um resolvedor de programação matemática, tal como o CPLEX. Como será visto posteriormente, essa solução foi suficiente apenas para gerar soluções ótimas em instâncias de tamanho pequeno e médio, mas para a resolução do problema de *pricing* em instâncias grandes ainda é utilizada uma significante quantidade de tempo.

6. Experimentos Computacionais

Os algoritmos propostos neste trabalho foram codificados em C++ e executados em uma máquina Intel i7 com 3,4 GHz e 16 Gb de memória RAM utilizando o sistema Operacional Linux 64 bits. Foi utilizado o CPLEX como resolvedor matemático e o resolvedor de *Branch-and-Price* BaPCode. Todos os testes com o modelo matemático utilizaram apenas uma *thread*.

Na próxima seção são descritas como as instâncias utilizadas nos testes foram geradas. A seguir, são exibidos os resultados encontrados por cada método proposto.

6.1. Instâncias

Como não haviam instâncias para o PED, para a realização dos testes, foram geradas instâncias inspiradas no problema da Mochila 1-vizinho como descritas a seguir. Como ponto de partida foram utilizadas algumas instâncias propostas anteriormente para o *Problema de Empacotamento com Conflitos (PEC)* [Gendreau et al., 2004]. Mais especificamente, foram utilizadas 40 instâncias da classes u (uniforme) propostas por Muritiba et al. [2010], com *bins* de capacidade 150 e número de itens iguais a $n_u = \{120, 250\}$. Os grafos de conflito foram removidos das instâncias e a capacidade dos *bins* foi dobrada. Para cada instância considerada, uma solução viável do Problema de Empacotamento foi produzida e a partir desta, as dependências foram criadas. Essa solução foi utilizada para garantir que houvesse uma solução viável para o problema com dependências.

A solução viável para o Problema de Empacotamento foi gerada selecionando-se a cada iteração um item escolhido aleatoriamente, o item é alocado ao primeiro *bin* que possui espaço residual suficiente. Não considerando itens que possuem mais de 80% de seu espaço ocupado.

Nesta solução criada, os itens foram separados nas vizinhanças conforme sua alocação aos *bins*. As características do grafo criado levaram em considerações três parâmetros: α – porcentagem de itens sem vizinhanças; β – número de itens em cada vizinhança; $ciclo$ – existência ou não de ciclo no grafo de dependências. O procedimento para criar as vizinhanças foi descrito abaixo.

1. $\alpha\%$ dos itens são escolhidos para serem "sem vizinhança": um item é escolhido aleatoriamente em cada *bin* da solução de referência e marcado como "sem vizinhança"até que o valor de α seja atingido;

2. Para cada item i que não está marcado como "sem vizinhança", selecione β itens para pertencer a sua vizinhança, começando com pelo menos um item escolhido aleatoriamente entre os itens pertencentes ao *bin* onde o item está alocado na solução de referência.

Após alguns testes empíricos, foram considerados os valores de $\alpha = \{30\%, 50\%\}$ e $\beta = \{2, 3, 5, 10, 20\}$. Foram criadas, instâncias com ciclo e sem ciclo. Essas configurações combinadas geraram 120 instâncias.

6.2. Resultados

Foram realizados testes comparativos para avaliar a eficiência dos métodos propostos. Em ambos os métodos, cada instância foi executada com um tempo limite de 1 hora. Os resultados encontrados são exibidos nas tabelas a seguir, que foram separadas de acordo com o tipo de grafo de dependência. Na Tabela 1 encontram-se os resultados para as instâncias com grafos de dependências sem ciclos, enquanto na Tabela 2 são exibidos os resultados para as instâncias que possuem grafos de dependências com ciclos. Em ambas as tabelas, as quatro primeiras colunas exibem informações das instâncias, onde as colunas *Inst.*, *NItens*, α e β, indicam respectivamente a instância testada, o número de itens que esta possui e os valores de α e $beta$ utilizados no procedimento de geração da instância.

Para cada método proposto, são reportados os valores do melhor limite inferior e do melhor limite superior, identificados nas colunas *LB* e *UB* respectivamente, encontrados na execução de cada instância. Ainda, a coluna *GAP* exibe a diferença percentual entre o limite inferior e superior, calculado como $100(UB - LB)/UB$. A coluna *NNós* mostra o número de nós e a coluna *T(s)*, exibe o tempo de CPU em segundos gasto pela execução do método.

Inst.	NItens	α	β	Modelo F1					Branch-and-price			
				LB	UB	GAP	NNós	T(s)	LB	UB	GAP	T(s)
i1	120	30	2	24	25	4,00%	56020	3600,05	24	24	0,00%	229,64
i2	120	30	3	25	25	0,00%	10908	450,54	25	25	0,00%	414,69
i3	120	30	5	23	24	4,17%	65994	3600,05	23	23	0,00%	20,31
i4	120	30	10	25	25	0,00%	1557	57,54	25	25	0,00%	16,11
i5	120	30	20	25	25	0,00%	5904	172,83	25	25	0,00%	11,34
i6	120	50	2	24	24	0,00%	54446	1665,31	24	24	0,00%	29,57
i7	120	50	3	24	25	4,00%	102912	3600,04	24	24	0,00%	13,20
i8	120	50	5	25	25	0,00%	11116	62,20	25	25	0,00%	12,05
i9	120	50	10	25	26	3,85%	161509	3600,05	25	25	0,00%	9,24
i10	120	50	20	23	24	4,17%	173636	3600,07	23	23	0,00%	8,85
i11	250	30	2	50	52	3,85%	95719	3600,17	50	50	0,00%	1532,68
i12	250	30	3	50	52	3,85%	11717	3600,17	50	50	0,00%	2138,91
i13	250	30	5	51	173	70,52%	4948	3600,19	51	51	0,00%	184,79
i14	250	30	10	50	194	74,23%	4411	3600,24	50	50	0,00%	82,51
i15	250	30	20	51	52	1,92%	10883	3600,34	51	51	0,00%	60,53
i16	250	50	2	51	52	1,92%	14424	3600,16	51	51	0,00%	129,74
i17	250	50	3	51	52	1,92%	23749	3600,16	51	51	0,00%	71,41
i18	250	50	5	52	53	1,89%	21184	3600,18	52	52	0,00%	45,83
i19	250	50	10	53	54	1,85%	176836	3600,21	53	53	0,00%	43,30
i20	250	50	20	51	52	1,92%	1591	3600,27	51	51	0,00%	70,69

Tabela 1: Resultados Computacionais para as Instâncias sem ciclo.

O modelo matemático $F1$, juntamente com os cortes apresentados para este modelo, resolvidos pelo CPLEX, chegou até a otimalidade em apenas 10 instâncias dentro do tempo determinado. Nas instâncias restantes, o valor do GAP permaneceu alto, chegando a um valor superior a 70% em algumas delas.

Enquanto isso, o método de *Branch-and-price* foi bem sucedido na resolução de todas as instâncias geradas. Em todos os casos, o valor do limite inferior obtido depois da geração de

Inst.	NItens	α	β	Modelo F1					Branch-and-price			
				LB	UB	GAP	NNós	T(s)	LB	UB	GAP	T(s)
i21	120	30	2	24	24	0,00%	16304	700,35	24	24	0,00%	147,69
i22	120	30	3	25	25	0,00%	975	30,37	25	25	0,00%	93,49
i23	120	30	5	23	24	4,17%	45044	3600,04	23	23	0,00%	19,73
i24	120	30	10	25	25	0,00%	1	39,01	25	25	0,00%	12,22
i25	120	30	20	25	25	0,00%	1	23,35	25	25	0,00%	12,07
i26	120	50	2	24	25	4,00%	166401	3600,04	24	24	0,00%	20,86
i27	120	50	3	24	25	4,00%	149253	3600,05	24	24	0,00%	13,73
i28	120	50	5	25	25	0,00%	1	7,92	25	25	0,00%	11,74
i29	120	50	10	25	26	3,85%	206954	3600,05	25	25	0,00%	9,78
i30	120	50	20	23	24	4,17%	199896	3600,06	23	23	0,00%	9,71
i31	250	30	2	50	52	3,85%	13023	3600,16	50	50	0,00%	2221,91
i32	250	30	3	50	52	3,85%	19546	3600,17	50	50	0,00%	178,09
i33	250	30	5	51	53	3,77%	3179	3600,20	51	51	0,00%	112,38
i34	250	30	10	50	52	3,85%	11490	3600,24	50	50	0,00%	70,30
i35	250	30	20	51	53	3,77%	1019	3600,33	51	51	0,00%	66,72
i36	250	50	2	51	52	1,92%	29080	3600,15	51	51	0,00%	74,62
i37	250	50	3	51	52	1,92%	27622	3600,16	51	51	0,00%	52,90
i38	250	50	5	52	53	1,89%	11968	3600,18	52	52	0,00%	41,29
i39	250	50	10	53	54	1,85%	12948	3600,22	53	53	0,00%	39,89
i40	250	50	20	51	52	1,92%	33083	3600,29	51	51	0,00%	67,03

Tabela 2: Resultados Computacionais para as Instâncias com ciclo.

colunas é forte o suficiente para resolver o problema apenas no nó raiz. Deste modo, não foi necessário realizar *branching* em nenhuma das instâncias. No entanto, o tempo de execução do *Branch-and-price* varia significantemente de uma instância para outra. Dependendo da instância, foram geradas entre 538 e 2908 colunas, e todo o processo levou entre 8,85 e 2221,91 segundos.

Todas essas instâncias foram resolvidas até a otimalidade, em contraste com a implementação da formulação, que não obteve sucesso em muitos casos. Isto demonstra o potencial de métodos baseados em geração de colunas para estes tipos de problemas e instâncias.

7. Conclusões

Neste artigo foi introduzido um novo problema na área de Otimização Combinatória, o Problema de Empacotamento com Dependências (PED). O problema proposto foi formalmente descrito e para sua resolução foram desenvolvidos um modelo matemático e um algoritmo de *Branch-and-price* (BP). Foram criadas ainda 40 instâncias para o problema. O modelo matemático descrito foi melhorado com a introdução de alguns cortes de redução de simetria existentes na literatura para o Problema de Empacotamento Clássico. No entanto, este foi capaz de resolver até a otimalidade apenas 10 instâncias. Já o algoritmo de *Branch-and-price*, resolveu até a otimalidade todas as instâncias criadas.

Como trabalhos futuros, pretende-se continuar o estudo do problema e estender os métodos desenvolvidos com a criação de novos cortes (para o desenvolvimento de um *Branch-and-cut*), e o aprimoramento da heurística de *pricing* para o BP já desenvolvido. Serão elaboradas ainda novas instâncias com um grande número de itens por *bin*, em média, que tendem a ser mais desafiadoras para métodos de BP. Finalmente, pretende-se expandir o problema proposto incorporando características adicionais inspiradas nas restrições reais existente no Problema de Realocações de Máquinas.

Referências

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., e Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329.

Borradaile, G., Heeringa, B., e Wilfong, G. (2012). The knapsack problem with neighbour constraints. *Journal of Discrete Algorithms*, 16:224–235.

Dantzig, G. B. e Wolfe, P. (1960). Decomposition Principle for Linear Programs. *Operations Research*, 8(1):101–111.

Delorme, M., Iori, M., e Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20.

Gavranović, H., Buljubašić, M., e Demirović, E. (2012). Variable Neighborhood Search for Google Machine Reassignment problem. *Electronic Notes in Discrete Mathematics*, 39:209–216.

Gendreau, M., Laporte, G., e Semet, F. (2004). Heuristics and Lower Bounds for the Bin Packing Problem with Conflicts. *Computers & Operations Research*, 31(3):347–358.

Lopes, R., Morais, V. W. C., Noronha, T. F., e Souza, V. a. a. (2015). Heuristics and matheuristics for a real-life machine reassignment problem. *International Transactions in Operational Research*, 22(1):77–95.

Masson, R., Vidal, T., Michallet, J., Penna, P. H. V., Petrucci, V., Subramanian, A., e Dubedout, H. (2013). An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems. *Expert Systems with Applications*, 40(13):5266–5275.

Muritiba, A., Iori, M., Malaguti, E., e Toth, P. (2010). Algorithms for the Bin Packing Problem with Conflicts. *INFORMS Journal on Computing*, 22(3):401–415.

Portal, G. M., Ritt, M., Borba, L. M., e Buriol, L. S. (2015). Simulated annealing for the machine reassignment problem. *Annals of Operations Research*, p. 1–22.

Roadef, 2012. ROADEF/EURO challenge 2012: Machine reassignment. `http://challenge.roadef.org/2012/en/index.php`. Acessado: 06.09.2016.

Sadykov, R. e Vanderbeck, F. (2013). Bin Packing with Conflicts: A Generic Branch-and-price Algorithm. *INFORMS Journal on Computing*, 25(2):244–255.

Wang, Z., Lü, Z., e Ye, T. (2016). Multi-neighborhood local search optimization for machine reassignment problem. *Computers and Operations Research*, 68:16–29.