Pesquisa Operacional para o Desenvolvimento
Vol. 18, e18001, p. 1-20, 2024

ISSN 1984-3534

UM ALGORITMO DE BRANCH-AND-CUT PARA O PROBLEMA DO EMPACOTAMENTO BIDIMENSIONAL EM CONTÊINERES NA PRESENÇA DE CONFLITOS[1]

Ana Clara N. dos Santos[a], Charbel D. Boulos[b], Mário C. San Felice[c], Pedro Hokama[a*]

[a]Instituto de Matemática e Computação
Universidade Federal de Itajubá, Itajuba-MG, Brasil

[b]Instituto de Engenharia de Sistemas e Tecnologias da Informação
Universidade Federal de Itajubá, Itajuba-MG, Brasil

[c]Departamento de Computação
Universidade Federal de São Carlos, São Carlos-SP, Brasil

Recebido 31/05/2023, aceito 04/03/2024

RESUMO

Neste artigo apresentamos o Problema do Empacotamento Bidimensional em Contêineres na Presença de Conflitos, que consiste em alocar um conjunto de itens de tamanhos variados no menor número possível de recipientes idênticos, respeitando as restrições de conflito entre os itens. Apresentamos um algoritmo de *Branch-and-Cut* que utiliza um modelo em Programação Linear Inteira, além de propormos melhorias envolvendo limitantes, pré-processamentos, cortes e quebras de simetrias. Para encontrar cortes violados, precisamos resolver o problema de decidir se um conjunto de itens pode ser empacotado em um contêiner. Para tanto, utilizamos um modelo de Programação por Restrições aprimorado usando padrões adaptados da literatura, em particular, Padrões Boschetti e *Meet in the Middle*. Testes computacionais foram realizados em instâncias adaptadas da literatura e os resultados foram muito significativos. As melhorias implementadas reduziram o tempo de execução de algumas instâncias de 3600 segundos para 0,02 segundos.

Palavras-chave: Programação linear inteira, Programação por restrições, Conjunto independente.

ABSTRACT

In this paper, we present the Two-dimensional Bin Packing Problem With Conflicts, which involves allocating a set of items of varying sizes into the fewest possible identical containers while respecting the conflicts between the items. We introduce a Branch-and-Cut algorithm that uses an Integer Linear Programming model, we also propose enhancements involving bounds, preprocessing, cuts and symmetry-breaking techniques. To find violated cuts, we need to solve the problem of deciding whether a set of items can be packed into a container. To do so, we use a Constraint Programming model improved with patterns adapted from the literature, in particular, Boschetti and Meet in the Middle Patterns. Computational tests were conducted on instances adapted from the literature, and the results were highly significant. The proposed improvements reduced the running time for some instances from 3600 seconds to 0.02 seconds.

Keywords: Integer linear programming, Constraint programming, independent set.

[*] Autor para correspondência. E-mail: hokama@unifei.edu.br
DOI: https://doi.org/10.4322/PODes.2024.003

[1]*Todos os autores assumem a responsabilidade pelo conteúdo do artigo.*

1. Introdução

Em empresas dos mais diversos ramos existem processos logísticos que visam a organização de produtos de forma compacta, melhorando a utilização do espaço em seu transporte e armazenamento e, consequentemente, reduzindo custos. Há ainda casos em que esse transporte ou armazenamento envolvem produtos que não podem ser alocados juntos, devido ao risco de contaminação, ou de combinar materiais explosivos, inflamáveis, corrosivos ou tóxicos (Capua et al., 2015).

O Problema do Empacotamento Bidimensional em Contêineres na Presença de Conflitos (*Two-dimensional Bin Packing Problem With Conflicts* - 2BPPC) consiste em alocar um conjunto de itens retangulares de tamanhos variados no menor número possível de recipientes idênticos, respeitando as restrições de conflito entre os itens. Esse problema é NP-difícil, ou seja, ele não pode ser solucionado em tempo polinomial a menos que $P = NP$. Isso decorre do 2BPPC generalizar dois problemas NP-Completos clássicos. O primeiro é o tradicional Problema do Empacotamento em Contêineres (*Bin Packing Problem* - BPP), que na versão unidimensional tem o objetivo de agrupar os itens no menor número de contêineres possível. O segundo é o problema da Coloração, em que se deseja particionar (colorir) vértices de um grafo de modo que vértices adjacentes tenham cores distintas. O 2BPPC também se relaciona com o Problema do Empacotamento Bidimensional Ortogonal (*Two-dimension Orthogonal Packing Problem - 2D-OPP)*, que verifica se há um empacotamento ou não em apenas um contêiner para um dado conjunto de itens. Este problema também é NP-Difícil, sendo possível reduzir o problema da Partição para o mesmo. Na redução, cada número corresponde à largura de um item e as alturas destes são iguais a 1; a altura do contêiner é 2 e a largura do mesmo é metade da soma dos números da entrada. Ao longo desse artigo usamos o termo *bin* para nos referirmos a um contêiner.

Gendreau et al. (2004) estão entre os primeiros a abordar o Problema do Empacotamento Unidimensional em Contêineres na Presença de Conflitos, utilizando heurísticas baseadas no algoritmo *First Fit Decreasing* (FFD), em procedimentos de coloração de grafos, em cliques e em limitantes inferiores. Muritiba et al. (2010), ao trabalharem com o mesmo problema, empregaram uma abordagem exata baseada na formulação do problema da cobertura por conjuntos. Capua et al. (2015) também abordaram este problema com um método baseado na metaheurística *Iterated Local Search*, além de um algoritmo *Large Neighborhood Search*.

O Problema do Empacotamento em Contêineres Bidimensional com Penalidades de Conflitos Variáveis é uma variante do 2BPPC que possui dois tipos de conflitos: (i) aqueles em que os itens não podem ser alocados no mesmo contêiner; e (ii) aqueles em que podem desde que incorra alguma penalidade, como distância de segurança ou proteção extra entre os itens. Este problema foi estudado por Li et al. (2014) utilizando uma heurística baseada no algoritmo *Iterative Maximal Area* (IMA) inicialmente proposto por El Hayek et al. (2008), juntamente com um procedimento de *Local Search* para melhorar as soluções. No trabalho de Queiroz et al. (2017) os autores abordaram o Problema da Mochila Bidimensional na Presença de Conflitos, utilizando formulações em programação inteira resolvidas com um algoritmo *branch-and-cut*, aprimorado com limitantes e cortes válidos. Além disso, propuseram uma heurística que combina os componentes das abordagens *Tabu search* e *Simulated Annealing*.

Khanafer et al. (2012) propuseram um *framework* baseado na decomposição em árvore para resolver o 2BPPC e apresentaram várias heurísticas para tornar o uso da decomposição eficaz. Epstein et al. (2008) também exploraram este mesmo problema e adotaram uma estratégia de resolução baseada em algoritmos de aproximação, centrada na coloração de grafos.

O presente trabalho propõe uma formulação em Programação Linear Inteira para o 2BPPC, e para fortalecer esta formulação, restrições baseadas em limitantes, pré-processamentos e cortes são apresentadas. Uma das famílias de restrições propostas elimina subconjuntos de itens que não conseguem ser empacotados juntos em um contêiner. Esta família apresenta um número exponencial de restrições, sendo inviável adicioná-las explicitamente no modelo. Dessa forma aplicamos a técnica de *Branch-and-Cut*, adicionando essas restrições apenas quando detectamos que uma foi violada.

Notadamente, para detectar se uma dessas restrições está violada é necessário resolver o 2D-OPP. Para tanto, apresentamos uma formulação em Programação por Restrições, e para diminuir o tempo de decisão do *solver* implementamos algumas melhorias, como o uso de Padrões. Em particular, Christofides e Whitlock (1977) e Herz (1972) demonstraram a eficiência do chamado Padrão Normal. Posteriormente Boschetti et al. (2002) apresentaram uma redução do Padrão Normal, que chamamos de Padrão Boschetti. Finalmente, Côté e Iori (2018) apresentaram o Padrão *Meet in the Middle*, que por sua vez, é uma redução do Padrão Boschetti.

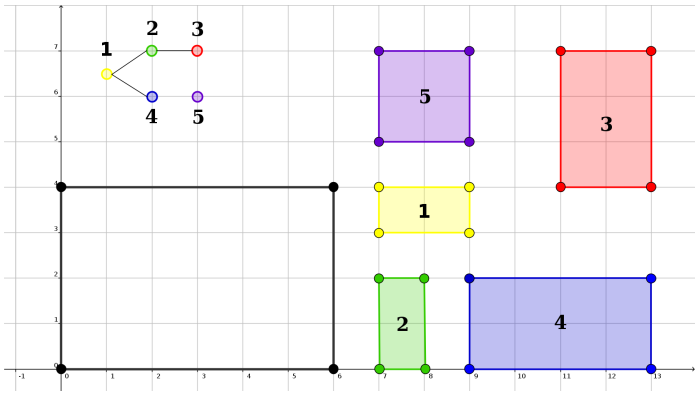
O restante deste artigo está organizado da seguinte forma: Na Seção 2 apresentamos a definição formal do 2BPPC, as notações que serão utilizadas e a formulação matemática do problema. Na Seção 3 apresentamos as melhorias desenvolvidas para tornar a resolução do modelo mais rápida. A Seção 4 apresenta o 2D-OPP, sua modelagem em Programação por Restrições e aborda os Padrões que restringem os domínios das entradas do problema. Os resultados computacionais dos experimentos com instâncias da literatura e a conclusão são encontrados nas Seções 5 e 6, respectivamente.

2. Descrição do Problema e Modelagem em Programação Linear Inteira

Nesta seção apresentamos a descrição formal do 2BPPC, e uma modelagem matemática em Programação Linear Inteira para o problema.

Definição 2.1 (2BPPC) - Problema do Empacotamento Bidimensional em Contêineres na Presença de Conflitos. *Seja V um conjunto de n itens, com cada $i \in V$ possuindo altura h_i e largura w_i, e seja Q um conjunto de bins idênticos de altura H e largura W. Todos os valores da entrada são inteiros não negativos. Considere também um grafo não direcionado $G = (V, E)$, no qual os vértices correspondem aos itens, e as arestas indicam conflitos entre itens. Em uma solução para este problema, cada item i deve ser alocado em exatamente um bin $k \in Q$, não sendo permitido que dois itens conflitantes sejam empacotados no mesmo bin. Além disso, a orientação de cada item é fixa, não sendo permitidas rotações e itens não devem se sobrepor ou exceder as bordas do bin. O objetivo é encontrar uma solução que minimize o número de bins utilizados.*

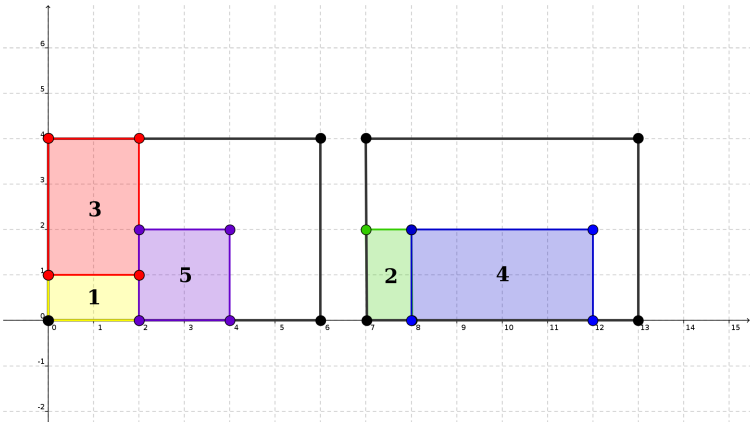
Figura 1: Exemplo de instância de entrada, na qual os cinco retângulos coloridos (lado direito) são os itens a serem empacotados em *bins* idênticos ao retângulo branco (canto inferior esquerdo), respeitando os conflitos do grafo (canto superior esquerdo).



Fonte: Elaborada pelos autores.

A Figura 1 apresenta um exemplo de uma instância com 5 itens $(1, 2, 3, 4, 5)$ com as dimensões (w_i, h_i), respectivamente, (2, 1), (1, 2), (2, 3), (4, 2) e (2, 2). Os *bins* tem largura $W = 6$ e altura $H = 4$. Acima do retângulo do contêiner, está uma representação do grafo de conflitos, em que cada vértice representa um item e existes três conflitos, sendo eles entre os itens 1 e 2; 1 e 4; 2 e 3.

Figura 2: Uma solução ótima utilizando dois *bins*.



Fonte: Elaborada pelos autores.

A Figura 2 apresenta uma possível solução ótima que utiliza 2 *bins* , em que os itens 1, 3 e 5 estão no primeiro bin, e os itens 2 e 4 estão no segundo *bin*. Note que nenhum dos pares de itens conflitantes estão no mesmo bin.

2.1. Modelagem Matemática

A formulação do 2BPPC apresentada a seguir é uma adaptação da proposta por Gendreau et al. (2004). Nela temos variáveis binárias dos tipos β_k e $\alpha_{i,k}$, em que para cada k, β_k indica se o *bin* k está sendo utilizado ($\beta_k = 1$) ou não ($\beta_k = 0$), e para cada par i e k, $\alpha_{i,k}$ indica se o item i está presente no *bin* k, assumindo valor 1 se está e 0 caso contrário. Considere $\mathcal{S}$ como a coleção de todos os subconjuntos de V que cabem simultaneamente em um *bin*.

$$\text{Minimize: } \sum_{k=1}^{n} \beta_k \tag{1}$$

$$\text{Sujeito à: } \sum_{i=1}^{n} w_i h_i \alpha_{i,k} \leq WH\beta_k, \qquad k \in \{1, \ldots, n\} \tag{2}$$

$$\sum_{k=1}^{n} \alpha_{i,k} = 1, \qquad i \in \{1, \ldots, n\} \tag{3}$$

$$\alpha_{i,k} + \alpha_{j,k} \leq 1, \qquad (i,j) \in E, k \in \{1, ..., n\} \tag{4}$$

$$\sum_{i \in S} \alpha_{i,k} \leq |S| - 1, \qquad S \notin \mathcal{S}, k \in \{1, ..., n\} \tag{5}$$

$$\beta_k \in \{0, 1\}, \qquad k \in \{1, ..., n\} \tag{6}$$

$$\alpha_{i,k} \in \{0, 1\}, \qquad i \in \{1, ..., n\}, k \in \{1, ..., n\} \tag{7}$$

O objetivo definido por (1) consiste em minimizar o número de *bins* utilizados. As restrições (2) garantem que a soma das áreas dos itens seja menor que a área do *bin* e que os itens só serão empacotados em *bins* que forem utilizados na solução, o conjunto de restrições (3) certifica que cada item será alocado exatamente em um recipiente, enquanto as restrições (4) asseguram que dois itens que possuem conflito (aresta entre eles) não podem ser alocados no mesmo *bin*. As restrições (5) garantem que os itens sejam empacotáveis, isto é, que os conjuntos de itens que

excedem os limites do *bin* quando empacotados juntos não serão alocados no mesmo *bin*. As restrições (6) e (7) apontam o domínio das variáveis binárias.

É importante ressaltar que o número de conjuntos não empacotáveis é exponencial, o que impossibilita que todas as restrições (5) sejam inseridas. Recorremos, então ao uso de *Lazy Constraints* para relaxar esse conjunto de restrições e adicioná-las sob demanda. Nesse ponto, surge o 2D-OPP, que verifica se o empacotamento de um conjunto de itens é factível ou não. Para resolvê-lo, utilizamos a técnica de Programação por Restrições. Tal metodologia será abordada com mais detalhes na Seção 4.

3. Melhorias

Foram desenvolvidas diversas melhorias para o modelo proposto, visando o fortalecimento do mesmo e a quebra de simetrias. Embora essas restrições não sejam estritamente necessárias para o funcionamento do modelo, sua inclusão faz com que as soluções sejam encontradas mais rapidamente. Cada melhoria recebeu uma sigla entre parênteses após o seu nome para que possa ser identificada com mais facilidade na seção de resultados computacionais.

Fixando O Primeiro Item (F): Partindo do princípio de que todos os itens deverão ser alocados em um, e somente um *bin*, o primeiro item da instância é pré-alocado no recipiente 1. Essa restrição elimina opções simétricas de empacotamento do item $i = 1$ no modelo, logo não são testadas as possibilidades deste item ser alocado nos outros *bins* disponíveis.

$$\alpha_{1,1} = 1 \tag{8}$$

Bins Consecutivos (B): Condiciona o uso do *bin* k, ao *bin* anterior $k - 1$ estar sendo utilizado. Essa restrição elimina simetrias, já que uma solução que utilizasse, por exemplo, os *bins* $\{1, 3, 7\}$ tem uma solução equivalente que utiliza apenas os *bins* $\{1, 2, 3\}$ já que os *bins* são idênticos.

$$\beta_{k-1} \geq \beta_k \qquad k \in \{2, ..., n\} \tag{9}$$

Restringindo Bins dos Itens (I): Essa restrição é uma generalização da melhoria F para os demais itens, restringindo os *bins* aos quais os itens além do primeiro podem ser alocados. Serão considerados apenas metade dos itens, para evitar que restrições fracas sejam adicionadas. Essa restrição obriga o item $i \in \{2, ..., n/2\}$ a estar nos um dos i primeiros *bins*.

$$\sum_{k=1}^{i} \alpha_{i,k} = 1 \qquad i \in \{2, ..., n/2\} \tag{10}$$

Restringindo Itens Largos ou Altos (R2): Decompõe os itens em dois conjuntos: altos e largos, em função da metade da altura e da largura do *bin*, isto é, $A_{altos} = \{i \in V : h_i > H/2\}$ e $A_{largos} = \{i \in V : w_i > W/2\}$. Note que, os itens largos ocupam mais da metade do *bin*, não sendo permitidos dois deles lado a lado, de modo que apenas uma pilha pode ser formada e deve ser limitada pela altura do *bin*. Assim, em cada *bin* a altura dos itens largos é acumulada e deve corresponder no máximo à altura do *bin* (11). Ocorre o mesmo para os itens altos, como são limitados a apenas uma fileira horizontal, suas larguras são somadas e restritas à no máximo a largura do *bin* (12). Essas restrições podam algumas combinações de itens que excederiam as dimensões do *bin*.

$$\sum_{i \in A_{\text{largos}}} \alpha_{i,k} h_i \leq H \qquad k \in \{1, ..., n\} \tag{11}$$

$$\sum_{i \in A_{\text{altos}}} \alpha_{i,k} w_i \leq W \qquad k \in \{1, ..., n\} \tag{12}$$

Restringindo Itens (R3): Assim como na restrição anterior, os itens são classificados em função de sua altura ou largura. Considerando o recipiente discretizado em 3 faixas de largura, são contabilizadas quantas faixas cada item excede usando os pesos definidos em (13). Note que, o ϵ que aparece em (13) é uma constante positiva pequena para garantir que a largura de um item excede $1/3$ ou $2/3$ da largura do *bin*. Após isso, em (15) são acumuladas as alturas dos itens classificados, e são limitadas a serem menores ou iguais a $2H$, já que por serem maiores que $1/3$ da largura do *bin*, só podem ser formadas no máximo duas pilhas lado a lado para os itens largos, sem que se exceda o tamanho do *bin*. Temos os pesos e restrições equivalentes para os itens altos em (14) e (16), respectivamente.

$$\text{pesoW}(i) = \left\lfloor \frac{w_i}{W/3 + \epsilon} \right\rfloor \qquad i \in \{1, ..., n\} \tag{13}$$

$$\text{pesoH}(i) = \left\lfloor \frac{h_i}{H/3 + \epsilon} \right\rfloor \qquad i \in \{1, ..., n\} \tag{14}$$

$$\sum_{i=1}^{n} \text{pesoW}(i)\alpha_{i,k}h_i \leq 2H \qquad k \in \{1, ..., n\} \tag{15}$$

$$\sum_{i=1}^{n} \text{pesoH}(i)\alpha_{i,k}w_i \leq 2W \qquad k \in \{1, ..., n\} \tag{16}$$

Adicionando Conflitos Por Incompatibilidade De Dimensões (D): É um pré-processamento que compara todos os pares de itens. Quando a soma da altura e da largura de um par excede as dimensões do recipiente, significa que não é possível acomodar os dois juntos. Portanto, uma aresta é adicionada ao grafo de conflitos. Estes novos conflitos evitam que empacotamentos infactíveis devido às restrições de tamanho do *bin* sejam testadas, podando o espaço de busca.

$$w_i + w_j > W \text{ e } h_i + h_j > H \text{ para } i, j \in V^2,\ i \neq j \implies E \leftarrow E \cup \{i, j\} \tag{17}$$

Adicionando Conflitos por Cliques (C): Nessa abordagem, foi utilizada a biblioteca Cliquer (Niskanen, 2002) adaptada para C++, que apresenta diversas funções para a busca de Cliques Maximais em grafos, baseadas em *Branch-and-Bound*. No 2BPPC, uma clique C representa um conjunto de itens que devem ser empacotados em $|C|$ *bins* distintos, já que possuem conflitos com todos os outros que também compõe a clique. Através das funções fornecidas pela biblioteca, cliques maximais de $G = (V, E)$ são encontradas e adicionadas a uma coleção de cliques $\mathcal{C}$. Então são inseridas as restrições de que itens da mesma clique não podem ser empacotados no mesmo recipiente, fortalecendo o modelo.

$$\sum_{i \in C} \alpha_{i,k} \leq 1 \qquad k \in \{1, ..., n\}, C \in \mathcal{C} \tag{18}$$

Restringindo Itens (R4): De modo semelhante às restrições R3, cada item recebe um peso em função de sua largura, dado por (19). Esse peso indica quantos quartos da largura do *bin* o item ultrapassa. Note que, substituindo cada item i por pesoW(i) itens de altura h_i e largura $W/4 + \epsilon$, temos uma relaxação da instância. Além disso, no máximo 3 desses itens podem ser colocados lado a lado. Assim, quando empilhados, estes itens devem respeitar 3 vezes a altura do *bin* (21). De modo semelhante, os itens recebem um peso em função de sua altura (20), e sua largura acumulada é limitada por (22). Essa restrição elimina soluções que não respeitariam as dimensões do *bin*.

$$\text{pesoW}(i) = \left\lfloor \frac{w_i}{W/4 + \epsilon} \right\rfloor \qquad i \in \{1, ..., n\} \tag{19}$$

$$\text{pesoH}(i) = \left\lfloor \frac{h_i}{H/4 + \epsilon} \right\rfloor \qquad i \in \{1, ..., n\} \tag{20}$$

$$\sum_{i=1}^{n} \text{pesoW}(i)\alpha_{i,k}h_i \leq 3H \qquad k \in \{1, ..., n\} \tag{21}$$

$$\sum_{i=1}^{n} \text{pesoH}(i)\alpha_{i,k}w_i \leq 3W \qquad k \in \{1, ..., n\} \tag{22}$$

Adicionando Pré Processamento (Rk): É uma generalização das melhorias R2, R3 e R4. Mas ao contrários delas, não trabalha com um número fixo de faixas para discretizar o *bin*. A quantidade de faixas que cada item ultrapassa é calculada em (23) para cada item i e número de faixas l. Essa melhoria verifica todas as divisões possíveis no intervalo estabelecido e escolhe o número de faixas l^* que resulte na maior altura acumulada possível em cada uma das $l^* - 1$ pilhas disponíveis (24). Em seguida a altura dos itens largos é acumulada e limitada (25). O mesmo processo é feito considerando a altura dos itens. Desse modo, combinações que ultrapassam o tamanho do *bin* são podadas.

$$\text{pesoW(i, l)} = \left\lfloor \frac{w_i}{W/l + \epsilon} \right\rfloor \qquad i \in \{1, ..., n\} \qquad l \in \{2, ..., W\} \tag{23}$$

$$l^* = \arg\max_{l} \left\{ \frac{\sum_{i=1} \text{pesoW(i, l)}h_i}{(l-1)} \right\} \tag{24}$$

$$\sum_{i=1}^{n} \text{pesoW(i, l}^*)\alpha_{i,k}h_i \leq (l^* - 1)H \qquad k \in \{1, ..., n\} \tag{25}$$

Adicionando Lower Bound (L1): Utilizamos o limitante inferior para o número de *bins* proposto por Carlier et al. (2007), no qual o conjunto dos itens V é decomposto em grandes, médios, altos, largos e pequenos em função dos inteiros p e q, tal que $1 \leq p \leq H/2$ e $1 \leq q \leq W/2$.

Seja:

- $A_{grandes} = \{i \in V : h_i > H - p \text{ e } w_i > W - q\}$
- $A_{med} = \{i \in V \backslash A_{grandes} : h_i > H/2 \text{ e } w_i > W/2\}$
- $A_p^a = \{i \in V : h_i > H/2 \text{ e } q \leq w_i \leq W/2\}$
- $A_p^l = \{i \in V : p \leq h_i \leq H/2 \text{ e } w_i > W/2\}$
- $A_p^p = \{i \in V : p \leq h_i \leq H/2 \text{ e } q \leq w_i \leq W/2\}$

Note que cada *bin* comporta no máximo um item grande ou médio, e que nenhum item alto, largo ou pequeno cabe junto de um grande. Após a classificação dos itens, é feito o cálculo do valor m" para cada item não grande. Para itens altos, largos ou pequenos, m" corresponde a quantos retângulos de altura p e largura q cada item ocupa. Para itens médios, m" indica (em valor negativo) quantos destes quadrados sobram no *bin* após colocar o item lá. Para cada combinação de p e q, o m" de todos os itens, exceto os grandes, são somados e usados para obter o número de *bins* necessários para os itens menores. Por fim, esse valor é somado ao número de itens grandes e médios. Ao final, o maior Lower Bound é adicionado ao modelo.

$$L_{4,BM}^{2D} = \max_{1 \leq p \leq H/2, 1 \leq q \leq W/2} \left\{ |A_{grande} \cup A_{med}| + \max \left\{ 0, \left\lceil \frac{(\sum_{a_i \in A \backslash\ A_{large}} m"(a_i, p, q))}{\lfloor H/p \rfloor \lfloor W/q \rfloor} \right\rceil \right\} \right\} \tag{26}$$

Onde:

- $m"(i,p,q) = \lfloor (W-w_i)/q \rfloor \lfloor (H-h_i)/p \rfloor - \lfloor (W-w_i)/q \rfloor \lfloor H/p \rfloor - \lfloor W/q \rfloor \lfloor (H-h_i)/p \rfloor$ se $i \in A_{med}$
- $m"(i,p,q) = \lfloor w_i/q \rfloor \lfloor H/p \rfloor - \lfloor w_i/q \rfloor \lfloor (H-h_i)/p \rfloor$ se $i \in A_p^a$
- $m"(i,p,q) = \lfloor W/q \rfloor \lfloor h_i/p \rfloor - \lfloor (W-w_i)/q \rfloor \lfloor h_i/p \rfloor$ se $i \in A_p^l$
- $m"(i,p,q) = \lfloor w_i/q \rfloor \lfloor h_i/p \rfloor$ se $i \in A_p^p$

Adicionando Upper Bound (L2): Para encontrar um limitante superior para o 2BPPC, utilizamos uma abordagem na qual todos os itens primeiramente são alocados em prateleiras, as quais possuem largura idêntica a do *bin*, altura igual à do primeiro item ali colocado, e herdam o conflito de cada um dos que ali estão. Alocamos um item por vez e, sempre que possível, o item corrente é colocado em uma prateleira já existente. Quando ocorrem conflitos ou as dimensões disponíveis na prateleira são excedidas, uma nova é alocada. Após todos os itens serem destinados a alguma prateleira, estas são empacotadas nos *bins*, seguindo as mesmas restrições de tamanho e compatibilidade. Ao final, temos uma solução viável, e portanto um limite para o número máximo de *bins* necessários para resolver o 2BPPC para aquela instância. Logo, não serão consideradas soluções que utilizem mais recipientes do que o número obtido como limitante.

Ordenação Decrescente por Conflitos (Oc) ou por Área (Oa): Tanto o número de conflitos de um item quanto sua área podem dificultar alocá-los em *bins* parcialmente preenchidos com outros itens. Por isso, nesta melhoria ordenamos os itens em ordem decrescente segundo um dos critérios anteriores. A ideia é que, tal ordenação junto da melhoria I impõe uma estratégia gulosa ao resolvedor visando priorizar itens mais difíceis de alocar.

4. Descrição do 2D-OPP e Modelagem em Programação por Restrições

Nesta seção será abordada a versão de decisão do 2D-OPP, como modelá-lo em programação por restrições e maneiras de restringir os domínios das variáveis do problema.

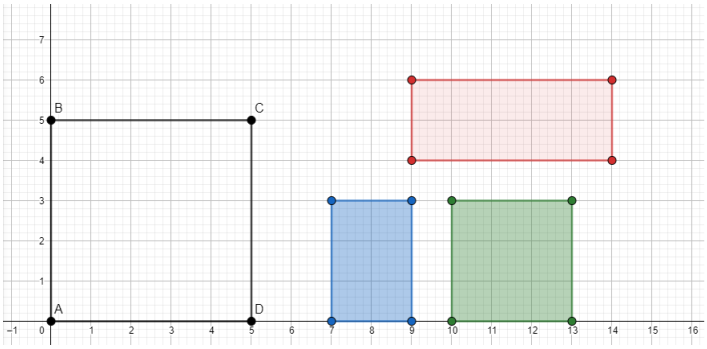
Definição 4.1 (2D-OPP) *- Problema do Empacotamento Bidimensional Uma instância $\mathcal{I}$ do 2D-OPP é formada por um contêiner C de altura H e largura W, e um conjunto de itens I, no qual cada item i possui sua respectiva altura h_i e largura w_i. Deve-se encontrar um empacotamento dos itens de I em C (ou provar que não existe), respeitando os limites de C e a não sobreposição dos itens, sendo que os itens não podem ser rotacionados e devem ter seus lados paralelos aos de C.*

A Figura 3 mostra o exemplo de uma instância do 2D-OPP com 3 itens, o item azul tem largura 2 e altura 3, o item verde é um quadrado com altura e largura iguais a 3 e o vermelho tem largura 5 e altura 2. O contêiner é um quadrado com $W = H = 5$. Já a Figura 4 apresenta uma solução para essa instância, em que de fato é possível empacotar os itens com o azul no ponto (0, 0) o verde à direita dele e o vermelho acima de ambos. Vale notar que o empacotamento deve ser possível sem rotacionar os itens e que nesse momento não existem conflitos.

4.1. Modelagem Matemática

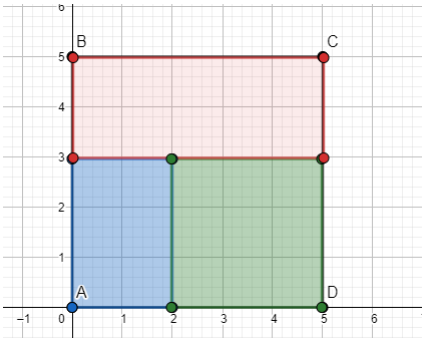
Dentre os diversos métodos para encontrar uma solução para o problema do empacotamento, neste trabalho será utilizada a programação por restrições (CP, do inglês *Constraint Programming*). Esse paradigma tem sido amplamente desenvolvido nas últimas décadas (Barták, 1999; Dechter, 2003; Rossi et al., 2006), e tem obtido resultados interessantes em diversos problemas de otimização, incluindo problemas de empacotamento (Clautiaux et al., 2008; Hokama et al., 2016; de Queiroz et al., 2017). Este é um paradigma de programação que trabalha com um

Figura 3: Exemplo de instância do problema do empacotamento, tendo o contêiner C altura e largura iguais a 5 e o conjunto I com 3 itens.



Fonte: Elaborada pelos autores.

Figura 4: Exemplo de solução para a instância da Figura 3.



Fonte: Elaborada pelos autores.

conjunto de variáveis, cada uma das quais podendo assumir valores de um conjunto finito, denominado domínio (nomenclatura D), e que utiliza restrições para relacionar os valores das variáveis e assim modelar diversos problemas.

Em nossa modelagem do 2D-OPP serão utilizadas duas variáveis para cada item i, X_i é a posição do item i no eixo x, ou seja, na direção da largura e Y_i é a posição do item i no eixo y, que corresponde à direção da altura. O domínio dessas variáveis será $D(X_i) = \{0, ..., W - w_i\}$ e $D(Y_i) = \{0, ..., H - h_i\}$ para cada i. Estes domínios garantem que nenhum item ultrapasse os limites de C. Na próxima subseção serão abordadas maneiras de reduzir estes domínios. A fim de garantir a não sobreposição dos itens, a seguinte restrição será adicionada para cada par de itens i e j:

$$[X_i + w_i \leq X_j] \text{ ou } [X_j + w_j \leq X_i] \text{ ou } [Y_i + h_i \leq Y_j] \text{ ou } [Y_j + h_j \leq Y_i] \tag{27}$$

Neste trabalho chamamos este modelo em que as variáveis possuem os domínios definidos anteriormente de “Sem Padrão”, em contraste com os próximos que usarão diferentes Padrões.

4.2. Padrões

Ao utilizar programação por restrições para modelar o problema do empacotamento bidimensional, uma dificuldade para encontrar uma solução ótima é o grande número de posições em que cada item pode ser empacotado. Uma possível solução para isso são os Padrões.

Definição 4.2 (Padrões) *Um Padrão é um subconjunto de pontos formado por diferentes critérios, que garantem a mesma viabilidade de uma instância $\mathcal{I}$ do 2D-OPP.*

Essa definição diz que uma instância $\mathcal{I}$ do 2D-OPP que tenha solução ótima no modelo geral (Sem Padrão) terá solução ótima de mesmo valor quando restringimos os domínios das variáveis ao padrão. Note que nem toda solução viável é preservada, mas alguma solução ótima ainda existe.

4.2.1. Padrão Normal

O Padrão Normal é a primeira restrição que será feita do domínio dos itens. Considerando apenas o eixo x e sendo 0 o início do contêiner, note que, dada uma solução qualquer, podemos deslizar os itens para a esquerda, a menos que outros itens já estejam lá. Assim, podemos focar no conjunto N^W que contém todas as combinações das larguras dos itens de I, que não ultrapassam a largura W do contêiner. Formalmente:

$$N^W = \left\{ x = \sum_{j \in S} w_j \colon 0 \leq x \leq W, S \in 2^I \right\} \tag{28}$$

O raciocínio análogo no eixo y permite obter os pontos N^H para restringir as posições verticais.

4.2.2. Padrão Boschetti

O Padrão Boschetti (Boschetti et al., 2002) é uma versão mais restrita do Padrão Normal. No Padrão Boschetti cada item i possui um conjunto B_i^W que contêm os pontos formados pela combinação das larguras dos demais itens e que somados à w_i não ultrapassem a largura de C. O conjunto de pontos do Padrão Boschetti B^W é formado pela união de todos os B_i^W. Formalmente:

$$B_i^W = \left\{ x = \sum_{j \in S} w_j \colon 0 \leq x \leq W - w_i, S \in 2^{I \setminus \{i\}} \right\} \tag{29}$$

$$B^W = \bigcup_{i \in I} B_i^W \tag{30}$$

Por exemplo, a instância apresentada na Figura 5 contém 6 itens com dimensões (2, 1), (2, 1), (3, 3), (1, 4), (3, 1) e (2, 2). Nessa situação observe que tanto no modelo sem padrão quanto no Padrão Normal o ponto 1 (do eixo horizontal) faz parte do $D(X_4)$, ou seja, o resolvedor pode atribuir valor 1 à variável X_4. Entretanto, observe que se o item 4 for colocado no ponto 1 nenhum outro item caberia a sua esquerda, de forma que qualquer solução que seja viável com $X_4 = 1$, também é viável com $X_4 = 0$. Portanto, o ponto 1 não precisa ser considerado para essa variável. Dessa forma $B_4^W = \{0, 2, 3, 4, 5\}$.

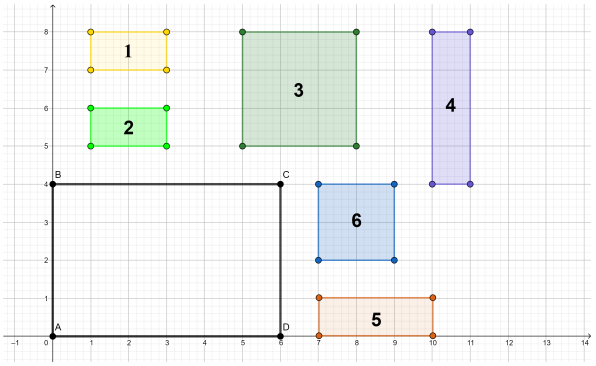
No entanto, note que ao unir os conjuntos formando B_W e fazermos $D(X_4) = B^W$ o ponto 1 volta a ser uma opção para o item 4. Por isso separamos o Padrão Boschetti em duas variações. A primeira é utilizando como um padrão geral fazendo $D(X_i) = B^W$, para todo $i \in I$, ou seja, todas as possibilidades encontradas. Neste caso temos que adicionar uma restrição $X_i + w_i \leq W$ para cada item i. A segunda é utilizando um domínio específico para cada item, onde terá apenas as possibilidades viáveis do item, ou seja, $D(X_i) = B_i^W$. Chamamos esta forma de Padrão Boschetti Individual.

Note que, de maneira análoga, o mesmo pode ser feito considerando altura.

4.2.3. Padrão Meet in the Middle

Por fim, o Padrão Meet in the Midle (MiM) tem o objetivo de diminuir a quantidade de pontos do Padrão Boschetti. De fato, o número de pontos do MiM não ultrapassa o número do Boschetti, embora um não seja subconjunto do outro (Côté e Iori, 2018), mas essa prova foge ao escopo deste artigo. O conjunto de uma dimensão deste padrão, diferente dos anteriores, é

Figura 5: Exemplo de uma instância do problema de empacotamento com C de altura 4 e largura 6, e um conjunto I com 6 itens.



Fonte: Elaborada pelos autores.

formado pela junção de outros dois conjuntos e utiliza um parâmetro de limiar denominado pela letra t (do inglês, threshold).

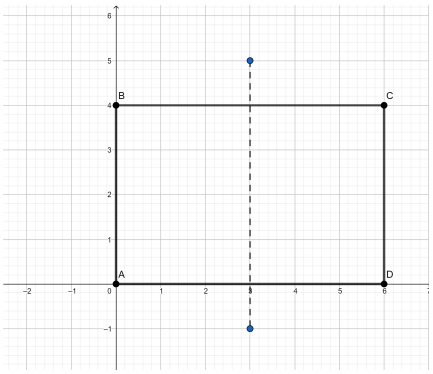
A partir do limiar t e para cada item i, é obtido o conjunto L_{it}, de pontos à esquerda de t, e o conjunto R_{it}, de pontos à direita de t. Os elementos destes conjuntos são obtidos da seguinte maneira:

$$L_{it} = \left\{ x = \sum_{j \in S} \omega_j \colon 0 \leq x \leq \min\{t-1, W-\omega_i\}, S \in 2^{I \setminus \{i\}} \right\} \tag{31}$$

$$R_{it} = \left\{ W - \omega_i - x \colon x = \sum_{j \in S} \omega_j, 0 \leq x \leq W - \omega_i - t, S \in 2^{I \setminus \{i\}} \right\}. \tag{32}$$

Por exemplo, no problema da Figura 5 e usando $t = W/2$ para largura, observa-se o contêiner da seguinte maneira:

Figura 6: Ilustração do contêiner da Figura 5 de acordo com um limiar de $W/2$.



Fonte: Elaborada pelos autores.

Encontrando os dois conjuntos se obtém o conjunto de cada item para largura que é a união do conjunto da esquerda e da direita, ou seja:

$$M_{it}^{W} = L_{it} \cup R_{it}. \tag{33}$$

Enfim, para encontrar o conjunto geral da largura, basta fazer a união dos conjuntos de cada item:

$$M_t^W = \bigcup_{i \in I} M_{it}^W. \tag{34}$$

Para encontrar os conjuntos de pontos na direção da altura será utilizado o mesmo limiar e a mesma lógica. Assim, acima do limiar terá o conjunto U_{it} (equivalente ao da direita) e abaixo o conjunto D_{it} (equivalente ao da esquerda). Da mesma forma em que foi feita foi na largura, para a altura será feito a união dos conjuntos de cima e debaixo para encontrar o conjunto da altura de cada item.

$$M_{it}^H = D_{it} \cup U_{it}. \tag{35}$$

O conjunto geral da altura é obtido pela união ao longo dos itens.

$$M_t^H = \bigcup_{i \in I} M_{it}^H. \tag{36}$$

A possibilidade de implementar domínios específicos para cada item, descrita no final da subsecção do Padrão Boschetti, vale também para o Padrão Meet in the Middle.

4.2.4. Pré-Processamentos

A fim de melhorar ainda mais o 2D-OPP foram implementados dois pré-processamentos, baseados na quebra de simetria e no alongamento dos itens, abordados por Côté e Iori (2018). Estes pré-processamentos complementam o Padrão MiM. Para melhor didática será abordado apenas para a dimensão da largura, mas o mesmo pode ser feito para a altura.

Pré-Processamento 1. Neste primeiro pré-processamento o objetivo é explorar a simetria do padrão MiM, para fixar um item no maior lado do contêiner dividido pelo limiar. Em particular, selecionamos um item i' de largura mínima e quando

$$t \leq \left\lceil \frac{W - w_{i'}}{2} \right\rceil \tag{37}$$

removemos o mesmo do cálculo do conjunto de pontos da esquerda (L_{it}) de cada item $i \in I \setminus \{i'\}$ e fazemos $L_{i't} = \emptyset$. Caso

$$t \geq \left\lceil \frac{W - w_{i'}}{2} \right\rceil + 1 \tag{38}$$

então removemos i' do cálculo do conjunto de pontos da direita (R_{it}) de cada item $i \in I \setminus \{i'\}$ e fazemos $R_{i't} = \emptyset$.

Pré-Processamento 2. É definida uma variável auxiliar $\tilde{w}_{ip}$, para cada item $i \in I$ e coordenada $p \in M_{it}^W$. Inicialmente cada variável $\tilde{w}_{ip}$ recebe o valor w_i. Então temos duas proposições.

Proposição (a). Considere um item $k \in I$ e uma coordenada $p \in M_{kt}^W$ para um t qualquer. Se $p \in L_{kt}$, definimos

$$q = \min\{W, \min\{s \in M_{it}^W : i \in I \setminus \{k\}, s \geq p + \tilde{w}_{kp}\}\} \tag{39}$$

Assim, o valor da variável auxiliar será atualizado para $\tilde{w}_{kp} = q - p$. Caso contrário, ou seja, se $p \in R_{kt}$, então:

$$q = \max\{0, \max\{s + \tilde{w}_{is} : s \in M_{it}^W, i \in I \setminus \{k\}, s + \tilde{w}_{is} \leq p\}\} \tag{40}$$

E então deve-se realizar as seguintes atualizações $R_{kt} = R_{kt} \cup \{q\} \setminus \{p\}$ e $\tilde{w}_{kq} = p + w_k - q$.

Proposição (b). Considere um item $k \in I$ e dois pontos do Padrão MiM, $p, s \in M_{it}^{W}$ com $p < s$. Suponha que, usando a *Proposição (a)*, é possível alongar a largura do item quando o mesmo for empacotado em p e s, respectivamente, em $\tilde{w}_{kp}$ e $\tilde{w}_{ks}$. Se

$$s + \tilde{w}_{ks} \leq p + \tilde{w}_{kp} \tag{41}$$

então a coordenada p é removida do conjunto M_{kt}^{W}.

Assim, o *Pré-Processamento 2* consiste do uso da *Proposição (a)* para alongar as larguras de todos os itens a partir de cada coordenada, seguido do uso da *Proposição (b)* para remover coordenadas desnecessárias dos conjuntos.

5. Resultados Computacionais

Todos os modelos e algoritmos descritos foram implementados em C++ e compilados com g++ 11.3 em ambiente Linux em um processador de 3.60GHz e 16GB de memória RAM. O resolvedor de PLI foi o IBM Cplex 22.1 e o resolvedor de CP foi o IBM CP Optimizer 22.1, ambos do mesmo pacote.

Os testes[1] foram executados em 64 instâncias, sendo 48 apresentadas por de Queiroz e Miyazawa (2012) para o problema da mochila com conflitos (Tabela 1). Destacamos que neste problema cada item está associado a um valor e o objetivo é empacotar o subconjunto mais valioso em um único contêiner. Neste trabalho utilizamos estas instâncias para o 2BPPC, em que todos os itens precisam ser empacotados em algum contêiner, e ignoramos os valores dos itens. Até onde sabemos, nenhum outro trabalho resolveu estas instâncias para o 2BPPC. Outras 16 instâncias foram criadas modificando as originais, apenas aumentando o tamanho do contêiner original para 60 por 60. O propósito dessas instâncias foi comparar como performavam os algoritmos quando mais itens cabiam por contêiner. As 16 novas instâncias identificadas com os números 49 até 64, se basearam respectivamente nas instâncias (01, 18, 35, 04, 21, 38, 07, 24, 41, 10, 27, 44, 13, 30, 47, 16).

No artigo de de Queiroz e Miyazawa (2012) as 48 instâncias foram geradas baseadas no procedimento descrito por Yamada et al. (2002) e Martello e Vigo (1998) em que cada instância tem um de 4 tipos diferentes de itens, uma densidade de conflitos em $\{7\%, 10\%, 13\%, 16\%\}$, o tamanho do contêiner varia nos valores $W = H \in \{30, 40, 50\}$ e o número de itens é sempre 25. As dimensões dos itens são obtidas aleatoriamente em intervalos definidos pelo tipo, a saber:

- Tipo I: $w_i \in \left[\frac{2}{3}W, W\right]$; $h_i \in \left[1, \frac{1}{2}H\right]$;
- Tipo II: $w_i \in \left[1, \frac{1}{2}W\right]$; $h_i \in \left[\frac{2}{3}H, H\right]$;
- Tipo III: $w_i \in \left[\frac{1}{2}W, W\right]$; $h_i \in \left[\frac{1}{2}H, H\right]$;
- Tipo IV: $w_i \in \left[1, \frac{1}{2}W\right]$; $h_i \in \left[1, \frac{1}{2}H\right]$.

Ao todo foram implementadas 12 possíveis melhorias entre pré-processamentos da entrada e restrições para o Programa Linear Inteiro Básico (1)-(7) do 2BPPC. No problema do empacotamento podemos resolver sem o uso de padrões ou com os padrões Boschetti e MiM, que podem ser aplicados de 2 formas: cada item tendo seu domínio personalizado, ou a criação de um domínio geral igual para todos os itens; totalizando 5 variações possíveis. Como testar todas as combinações seria inviável, optamos por agrupar as melhorias em configurações incrementais, de forma a verificar o tempo em que cada uma consegue resolver as instâncias.

Os resultados dos testes são apresentados através de gráficos de perfis de desempenho (do inglês, *performance profiles*) (Dolan e Moré, 2002), que são usados para comparar o desempenho

[1] As instâncias, testes e código-fonte serão disponibilizados em `https://hokama.com.br/podes_2bppc/`

Tabela 1: As 48 instâncias originalmente para o problema da mochila com conflito.

Inst.	W=C	Tipo	D(%)	Inst.	W=C	Tipo	D(%)	Inst.	W=C	Tipo	D(%)
1	30	I	7	17	40	I	7	33	50	I	7
2	30	II	7	18	40	II	7	34	50	II	7
3	30	III	7	19	40	III	7	35	50	III	7
4	30	IV	7	20	40	IV	7	36	50	IV	7
5	30	I	10	21	40	I	10	37	50	I	10
6	30	II	10	22	40	II	10	38	50	II	10
7	30	III	10	23	40	III	10	39	50	III	10
8	30	IV	10	24	40	IV	10	40	50	IV	10
9	30	I	13	25	40	I	13	41	50	I	13
10	30	II	13	26	40	II	13	42	50	II	13
11	30	III	13	27	40	III	13	43	50	III	13
12	30	IV	13	28	40	IV	13	44	50	IV	13
13	30	I	16	29	40	I	16	45	50	I	16
14	30	II	16	30	40	II	16	46	50	II	16
15	30	III	16	31	40	III	16	47	50	III	16
16	30	IV	16	32	40	IV	16	48	50	IV	16

Fonte: Elaborada pelos autores.

de vários algoritmos sobre um conjunto de instâncias. Dada uma instância, temos o tempo de solução de cada algoritmo e o menor tempo entre eles. Assim, para cada algoritmo podemos calcular a razão entre seu tempo e o menor tempo alcançado, que chamamos de razão de desempenho. O eixo y do gráfico corresponde à porcentagem das instâncias, enquanto o eixo x corresponde à razão de desempenho. Assim, um ponto (x, y) indica que uma fração y das instâncias foi resolvida com razão de desempenho no máximo x. Em nossos gráficos cada curva representa uma configuração do nosso algoritmo, identificada pelas siglas das melhorias que a compõem. Em perfis de desempenho, consideramos que a curva com uma área maior abaixo dela fornece os melhores resultados gerais. Ou seja, quanto mais acima e a esquerda a curva, mais rápido é o algoritmo.

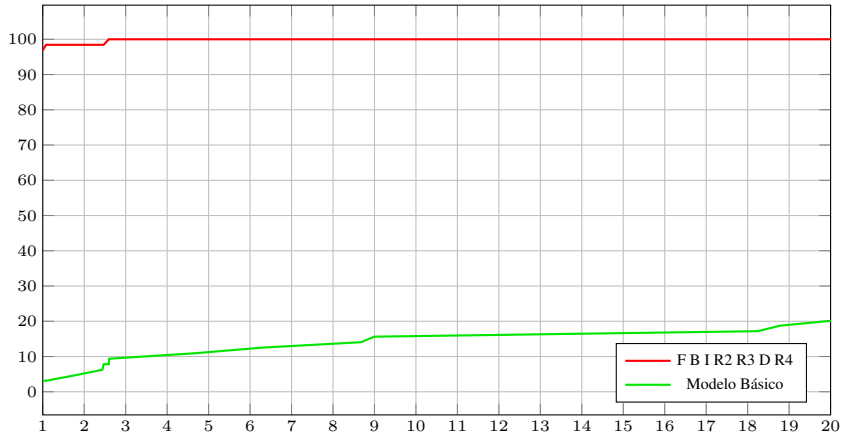
Chamaremos de Modelo Básico o modelo (1)-(7) em que os empacotamentos são resolvidos por CP sem o uso de padrões, e F B I R2 R3 D R4 significa que foram utilizadas as melhorias F, B, I, R2, R3, D e R4 apresentadas na Sessão 3. Além disso vale ressaltar que independente das melhorias e padrões utilizados, com tempo suficiente é possível atingir uma solução ótima. Entretanto, como o algoritmo de empacotamento pode ser chamado milhares de vezes durante a resolução de uma única instância, limitamos a 30 segundos o tempo de execução de cada empacotamento, sendo que caso não seja encontrado um empacotamento nesse tempo ele é considerado inviável.

A Figura 7 apresenta a comparação entre o modelo básico e a configuração F B I R2 R3 D R4. Podemos notar que o impacto dessas melhorias foi bastante significativo, pois a curva do modelo com melhorias começar perto de 100, significa que praticamente todas as instâncias foram resolvidas mais rapidamente por esta versão. Enquanto o modelo básico resolveu menos de 20% das instâncias mesmo com 20 vezes mais tempo.

A Figura 8 mostra a comparação entre a configuração F B I R2 R3 D R4 da figura anterior, acrescida das melhorias C e Rk separadamente. Observamos que com o uso de Rk melhoramos o desempenho, e com a ativação das cliques conseguimos melhorias ainda mais significativas, sendo que foi a mais rápida desse conjunto em 50% das instancias.

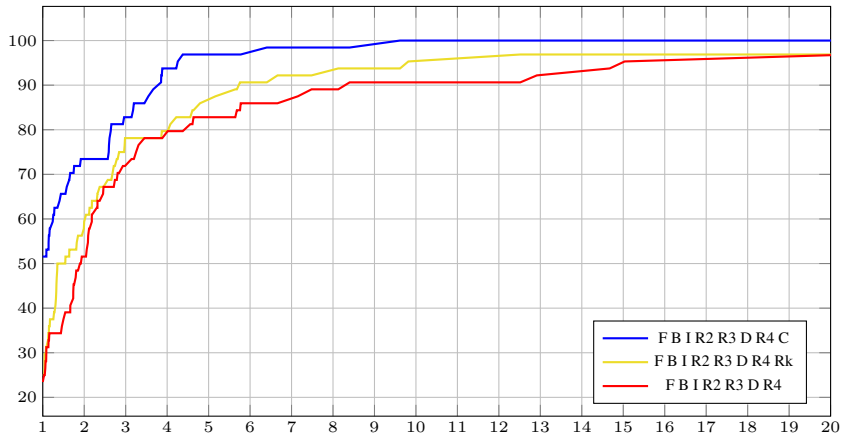
A Figura 9 apresenta a comparação entre a configuração F B I R2 R3 R4 D Rk, com a mesma adicionada dos limitantes inferiores e superiores L1 e L2. Observamos que os limitantes trouxeram uma melhora no desempenho do modelo. Ainda na Figura 9, também testamos essa

Figura 7: Comparação dos resultados entre o Modelo Básico e o modelo com as melhorias F, B, I, R2, R3, D e R4. Em ambos os casos o problema do empacotamento foi resolvido com o modelo de Programação por Restrições mais simples, apresentado na Seção 4.1.



Fonte: Elaborada pelos autores.

Figura 8: Comparação dos resultados entre a configuração F B I R2 R3 D R4, e desta acrescida da melhoria Rk, e depois da primeira acrescida da melhoria C. Em todos os casos o problema do empacotamento foi resolvido com o modelo de CP mais simples apresentado na Seção 4.1.



Fonte: Elaborada pelos autores.

última configuração acrescida das ordenações por área e depois da ordenação por conflitos. E observamos que ambas trouxeram ganhos para o algoritmo e que a ordenação por conflitos foi superior.

Por fim, a Figura 10 apresenta a comparação dos 5 diferentes padrões testados, aplicados quando estão ligadas as melhorias da configuração F B I R2 R3 D R4 Rk L1 L2 Oc, que obtiveram os melhores resultados anteriormente. Podemos observar que o padrão MiM individual obteve as melhores soluções para mais de 40% das instâncias e para mais de 90% das instâncias ele esteve no máximo a 20% do menor tempo.

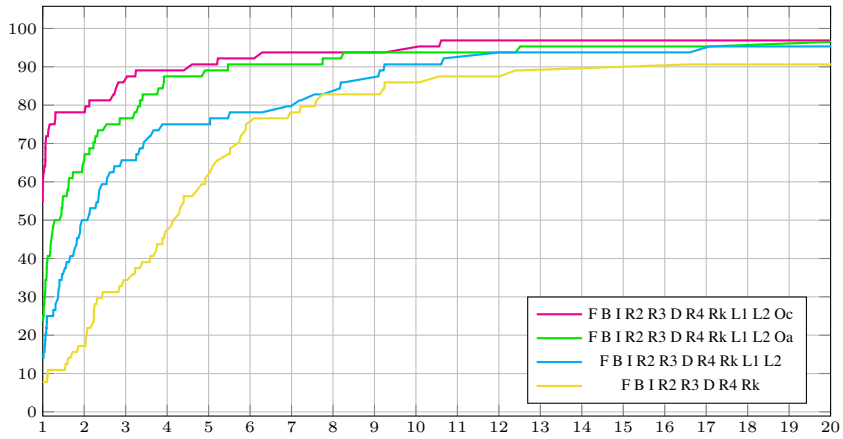
Por fim as Tabelas 2 e 3 mostram os tempos para a resolução de cada uma das 64 instâncias. Apresentamos nestas tabelas a comparação dos tempos “T(s)”, número de empacotamentos “#Emp” que se tentaram resolver e tempo total gasto apenas no modelo de empacotamento “Emp T(s)”. Mostramos os resultados obtidos pelo modelo básico e sem o uso de padrões, e pelo

Tabela 2: Comparação do Modelo Básico vs. o Modelo quando as melhorias F, B, I, R2, R3, D, R4, Rk, L1, L2 e Oc estão ligadas e o padrão MiM individual é utilizado.

	Modelo Básico			Melhor configuração		
Instância	T(s)	#Emp	Emp T(s)	T(s)	#Emp	Emp T(s)
1	3557.18	1478	6.52	0.15	43	0.12
2	13.50	2152	5.26	1.52	831	1.37
3	3599.61	528	0.35	0.16	20	0.01
4	3599.50	904	133.87	0.01	4	0.01
5	3592.05	2763	8.23	0.02	7	0.01
6	3595.90	1426	6.40	0.21	120	0.19
7	3598.34	412	0.30	0.05	18	0.01
8	0.14	41	0.10	0.03	6	0.02
9	236.94	3481	60.09	0.17	65	0.11
10	3597.07	548	0.98	0.06	27	0.04
11	3596.68	520	0.50	0.03	16	0.01
12	14.17	1099	11.88	0.22	84	0.20
13	2468.36	2655	9.24	0.02	7	0.01
14	3599.08	738	1.74	0.02	7	0.01
15	3599.15	439	0.38	0.06	50	0.03
16	3598.08	497	1.69	0.18	75	0.14
17	3473.31	6778	25.76	0.69	379	0.59
18	3599.64	1798	3.92	0.05	29	0.04
19	3599.39	435	0.46	0.05	34	0.02
20	3599.35	3505	356.92	1.42	218	1.38
21	3599.34	3076	139.80	0.02	7	0.01
22	3596.26	2023	7.76	2.41	1209	1.98
23	3315.81	473	0.37	0.03	17	0.01
24	3598.21	1251	7.28	0.46	231	0.44
25	17.21	701	14.82	6.04	41	6.02
26	3599.24	1172	2.93	0.04	17	0.03
27	3597.88	411	0.31	0.11	20	0.01
28	50.40	1518	41.68	0.30	146	0.27
29	3598.33	1278	2.88	0.27	57	0.09
30	3599.41	622	1.50	0.30	179	0.26
31	3599.45	510	0.47	0.02	14	0.01
32	3580.99	1198	4.43	1.08	192	1.05
33	3599.86	2973	10.76	0.04	24	0.03
34	1954.27	1415	4.87	0.08	42	0.06
35	3597.37	394	0.31	0.03	17	0.01
36	2.70	381	2.39	0.29	54	0.27
37	3596.66	1549	5.18	0.02	9	0.01
38	3599.72	629	1.40	0.02	8	0.01
39	3597.09	521	0.32	0.11	38	0.01
40	7.05	164	6.87	1.18	249	1.13
41	3599.41	2108	8.65	0.04	18	0.03
42	3599.71	1133	3.53	0.05	18	0.02
43	3468.51	520	0.32	0.06	19	0.01
44	2649.78	1040	3.66	0.03	12	0.02
45	3599.61	1053	2.78	0.02	9	0.01
46	3599.78	4916	29.84	0.03	12	0.02
47	3030.00	362	0.34	0.02	15	0.01
48	20.91	352	19.34	0.34	109	0.31

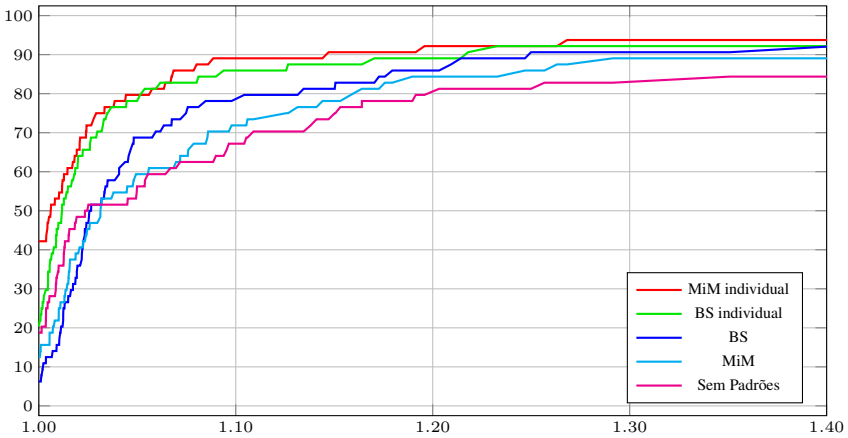
Fonte: Elaborada pelos autores.

Figura 9: Comparação dos resultados entre a configuração F B I R2 R3 D R4 Rk, e dela acrescida dos Limitantes Inferiores e Superiores, e ordenando os itens de acordo com sua área ou conflito.



Fonte: Elaborada pelos autores.

Figura 10: Comparação dos resultados entre o modelo com a melhor configuração anterior, variando o domínio das variáveis na Programação por Restrições, de acordo com os Padrões descritos na Seção 4.2.



Fonte: Elaborada pelos autores.

modelo quando as melhorias F, B, I, R2, R3, D, R4, Rk, L1, L2 e Oc estão ligadas e o padrão MiM individual é utilizado. Podemos notar ganhos expressivos em praticamente todas as instâncias. Vale notar que não apenas o tempo total sofreu redução como o número de empacotamentos tentados também foi bastante reduzido.

6. Conclusões

Neste artigo apresentamos o Problema do Empacotamento Bidimensional em Contêineres na Presença de Conflitos. Neste problema um conjunto de itens bidimensionais retangulares precisa ser empacotado no menor número de contêineres homogêneos de tamanho fixo. Além disso, alguns pares de itens possuem um conflito e não podem ser empacotados em um mesmo contêiner.

Apresentamos um modelo em Programação Linear Inteira para o problema, que apresenta um número exponencial de restrições, aquelas que eliminam conjuntos de itens que não conseguem ser colocados em um mesmo contêiner. Por esse motivo, não adicionamos essas restrições *a priori*,

Tabela 3: Comparação do Modelo Básico vs. o Modelo quando as melhorias F, B, I, R2, R3, D, R4, Rk, L1, L2 e Oc estão ligadas e o padrão MiM individual é utilizado.

	Modelo Básico			Melhor configuração		
Instância	T(s)	#Emp	Emp T(s)	T(s)	#Emp	Emp T(s)
49	0.66	42	0.09	0.79	3	0.79
50	467.02	1438	463.70	0.24	84	0.22
51	3596.60	674	0.74	0.02	14	0.01
52	0.18	35	0.14	0.06	2	0.06
53	3537.81	3906	587.55	777.82	782	777.55
54	2359.73	1163	3.28	0.16	100	0.15
55	360.08	76	360.03	362.46	105	362.43
56	1.03	42	0.14	0.02	3	0.01
57	3563.05	3314	18.63	0.02	7	0.01
58	30.76	58	30.12	0.02	3	0.01
59	54.72	2435	13.28	0.30	166	0.24
60	3599.87	1656	5.56	0.02	5	0.01
61	0.83	53	0.11	0.02	3	0.01
62	862.08	142	0.53	0.16	52	0.12
63	3599.08	1469	3.62	0.02	9	0.01
64	0.56	52	0.10	0.01	3	0.01

Fonte: Elaborada pelos autores.

apenas quando detectamos que uma delas foi violada durante o *branch-and-bound*. Notadamente descobrir se um conjunto de itens possui um empacotamento em um contêiner é um problema NP-Difícil.

Para resolver o problema do empacotamento utilizamos uma formulação em Programação por Restrições. Também adaptamos padrões que visam reduzir o domínio das variáveis desta formulação. Para a Formulação em PLI foram apresentadas diversas melhorias envolvendo limitantes, pré-processamentos, cortes e quebras de simetrias.

Os resultados obtidos mostram que para o conjunto de instâncias testadas as técnicas aplicadas foram muito significativas. Por exemplo, várias instâncias que não eram resolvidas em 3600 segundos passaram a ser resolvidas em 0.03 segundos ou menos. Somando os tempos para resolver as 64 instâncias o modelo básico levou quase 44 horas (o tempo máximo de cada instância é 3600 segundos) enquanto na melhor configuração esse tempo foi de 30 minutos, sendo que praticamente todo esse tempo foi gasto nas instâncias 53 e 55.

Concluímos então que as melhorias adaptadas e propostas foram muito efetivas. Nos próximos passos dessa pesquisa pretendemos aplicar mais algumas melhorias, como por exemplo o procedimento de *lifting* dos itens, e testar outras formas de quebra de simetria, como exigir que os bins sejam ordenados por área de ocupação. Além disso, também será interessante comparar os nossos resultados com outros da literatura. Embora para o nosso conhecimento não existam trabalhos que apresentem métodos exatos para o 2BPPC, podemos tentar comparar com métodos heurísticos, ou com problemas correlatos. Outras integrações também podem ser interessantes, por exemplo, com problemas de roteamento ou de dimensionamento de lotes.

Agradecimentos. Os autores agradecem o apoio do CNPq, FAPEMIG e Unifei pelo suporte financeiro.

Referências

Barták, R. Constraint programming: In pursuit of the holy grail. In: *Proceedings of the Week of Doctoral Students (WDS99)*. MatFyzPress Prague, 1999. volume 4. p. 555–564.

Boschetti, M. A., Mingozzi, A. e Hadjiconstantinou, E. New upper bounds for the two-dimensional orthogonal non-guillotine cutting stock problem. *IMA Journal of Management Mathematics*, v. 13, n. 2, p. 95–119, 2002.

Capua, R., Frota, Y., Vidal, T. e Ochi, L. S. Um algoritmo heurístico para o problema de bin packing com conflitos. In: *XLVII Simpósio Brasileiro de Pesquisa Operacional*. Porto de Galinhas, Pernambuco - PE. Sociedade Brasileira de Pesquisa Operacional (SOBRAPO), 2015. p. 4252–4261.

Carlier, J., Clautiaux, F. e Moukrim, A. New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. *Computers & Operations Research*, v. 34, n. 8, p. 2223–2250, 2007.

Christofides, N. e Whitlock, C. An algorithm for two-dimensional cutting problems. *Operations Research*, v. 25, n. 1, p. 30–44, 1977.

Clautiaux, F., Jouglet, A., Carlier, J. e Moukrim, A. A new constraint programming approach for the orthogonal packing problem. *Computers & Operations Research*, v. 35, n. 3, p. 944–959, 2008.

Côté, J.-F. e Iori, M. The meet-in-the-middle principle for cutting and packing problems. *INFORMS Journal on Computing*, v. 30, n. 4, p. 646–661, 2018.

de Queiroz, T. A. e Miyazawa, F. K. Problema da mochila 0-1 bidimensional com restrições de disjunção. In: *XVI Conferência Latino-Ibero-Americana de Pesquisa Operacional/XLIV Simpósio Brasileiro de Pesquisa Operacional*. Rio de Janeiro - RJ. Sociedade Brasileira de Pesquisa Operacional (SOBRAPO), 2012. p. 3315–3326.

de Queiroz, T. A., Hokama, P. H. D. B., Schouery, R. C. S. e Miyazawa, F. K. Two-dimensional disjunctively constrained knapsack problem: Heuristic and exact approaches. *Computers & Industrial Engineering*, v. 105, p. 313–328, 2017.

Dechter, R. *Constraint processing*. Morgan Kaufmann, 2003.

Dolan, E. D. e Moré, J. J. Benchmarking optimization software with performance profiles. *Mathematical Programming*, v. 91, n. 2, p. 201–213, 2002.

El Hayek, J., Moukrim, A. e Nègre, S. New resolution algorithm and pretreatments for the two-dimensional bin-packing problem. *Computers & Operations Research*, v. 35, n. 10, p. 3184–3201, 2008.

Epstein, L., Levin, A. e Van Stee, R. Two-dimensional packing with conflicts. *Acta Informatica*, v. 45, p. 155–175, 2008.

Gendreau, M., Laporte, G. e Semet, F. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research*, v. 31, n. 3, p. 347–358, 2004.

Herz, J. Recursive computational procedure for two-dimensional stock cutting. *IBM Journal of Research and Development*, v. 16, n. 5, p. 462–469, 1972.

Hokama, P., Miyazawa, F. K. e Schouery, R. C. A bounded space algorithm for online circle packing. *Information Processing Letters*, v. 116, n. 5, p. 337–342, 2016.

Khanafer, A., Clautiaux, F. e Talbi, E.-G. Tree-decomposition based heuristics for the two-dimensional bin packing problem with conflicts. *Computers & Operations Research*, v. 39, n. 1, p. 54–63, 2012.

Li, K., Liu, H., Wu, Y. e Xu, X. A two-dimensional bin-packing problem with conflict penalties. *International Journal of Production Research*, v. 52, n. 24, p. 7223–7238, 2014.

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

Muritiba, A. E. F., Iori, M., Malaguti, E. e Toth, P. Algorithms for the bin packing problem with conflicts. *Informs Journal on Computing*, v. 22, n. 3, p. 401–415, 2010.

Niskanen, S. *Cliquer - routines for clique searching*, 2002. Disponível em: `https://users.aalto.fi/~pat/cliquer.html`. Acesso em: 09/09/2022.

Rossi, F., Van Beek, P. e Walsh, T. *Handbook of constraint programming*. Elsevier, 2006.

Yamada, T., Kataoka, S. e Watanabe, K. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Information Processing Society of Japan Journal*, v. 43, n. 9, 2002.