Heurísticas para o Problema do Empacotamento Colorido*

Renan F.F. da Silva, Yulle G.F. Borges, Rafael C.S. Schouery
Instituto de Computação - Universidade Estadual de Campinas (Unicamp)
Av. Albert Einstein, 1251 - Cidade Universitária, Campinas - SP, 13083-852
{r223989@dac, glebbyo@ic, rafael@ic}.unicamp.br

RESUMO

O Problema do Empacotamento Colorido (CBPP) consiste em empacotar um conjunto de itens, cada um com um tamanho e uma cor, em recipientes de capacidade limitada, de modo a minimizar a quantidade de recipientes utilizados e a satisfazer a restrição que dois itens de mesma cor não podem ser empacotados em sequência em um mesmo recipiente. Em um estudo preliminar, propomos para o CBPP algumas heurísticas e uma abordagem meta-heurística baseada na *Variable Neighborhood Search* (VNS). Já neste artigo, trazemos alguns detalhes de implementação dos algoritmos apresentados na versão preliminar, e também propormos uma abordagem math-heurística que mescla programação linear e as meta-heurísticas VNS e *Greedy Randomized Adaptive Search* (GRASP). Os resultados computacionais indicam que a nossa math-heurística é superior à VNS, e que ambas as abordagens conseguem encontrar boas soluções mesmo para instâncias do problema com grande quantidade de itens.

PALAVRAS CHAVE. Corte e Empacotamento, Meta-heurística, Math-heurística

ABSTRACT

The Colorful Bin Packing Problem (CBPP) consists of packing a set of items, each one with a size and a color, in bins of limited capacity, minimizing the number of bins used and satisfying the constraint that two items of the same color cannot be packed side by side in the same bin. In a preliminary study, we proposed for the CBPP some heuristics and a meta-heuristic approach, based on the Variable Neighborhood Search (VNS). In this article, we bring some implementation details for the algorithms of the preliminary study, and we propose a math-heuristic approach, that mixes linear programming with the meta-heuristics VNS and Greedy Randomized Adaptive Search (GRASP). The results indicate that our math-heuristic is superior to VNS and that both approaches are capable of finding good solutions, even for instances with a large number of items.

KEYWORDS. Cutting and Packing, Meta-heuristic, Math-heuristic

*Estudo financiado pelos processos 2015/11937-9 e 2020/06511-0, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), e processos 144257/2019-0, 311039/2020-0 e 425340/2016-3, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

https://proceedings.science/p/157378?lang=pt-br

1. Introdução

No Problema do Empacotamento (BPP, do inglês: *Bin Packing Problem*), é dado um conjunto de itens, cada um com um tamanho, que precisam ser empacotados em recipientes de capacidade limitada, minimizando a quantidade de recipientes utilizados. Uma generalização do BPP é o Problema do Empacotamento Colorido (CBPP, do inglês: *Colorful Bin Packing Problem*), onde cada item, além de um tamanho, também possui uma cor, e temos que satisfazer a restrição de cores: dois itens de mesma cor não podem ser empacotados lado a lado em um mesmo recipiente.

O CBPP foi proposto independentemente por [Böhm et al., 2014] e [Dósa e Epstein, 2014], sendo estudado nesses artigos apenas do ponto de vista de algoritmos online e algoritmos de aproximação. Em particular, [Böhm et al., 2014] mostrou que as adaptações dos algoritmos clássicos do BPP chamados *First Fit*, *Best Fit* e *Worst Fit* não possuem razões de aproximação constante para o CBPP. Além disso, os autores provaram que qualquer algoritmo online precisa ter uma razão de competitividade assintótica de pelo menos 2.5 e mostram um algoritmo 3.5-competitivo (absoluto). Já [Borges e Schouery, 2020] propuseram um algoritmo exato para o CBPP, usando formulações pseudo-polinomiais, que resolvem instâncias geradas aleatoriamente com até 500 itens e 15 cores em segundos. No trabalho preliminar deste artigo [Silva et al., 2021], publicado como resumo estendido, propomos a adaptação de heurísticas do BPP para o CBPP acompanhado de algumas heurísticas novas para o problema, além de uma abordagem meta-heurística para o CBPP baseada na *Variable Neighborhood Search* (VNS).

O CBPP é um problema NP-difícil, visto que o mesmo é uma generalização do BPP [Karp, 1972]. Quando se trata de problemas NP-difíceis, além da abordagem de algoritmos exatos, outra abordagem interessante é a de algoritmos heurísticos, em especial meta-heurísticos, que costumam entregar soluções de alta qualidade em tempo computacionalmente razoável. Na literatura, muito autores já estudaram o BPP e suas variantes, usando tanto algoritmos exatos como algoritmos meta-heurísticos.

Os algoritmos exatos para o BPP se baseiam principalmente em duas categorias de formulação: a de [Gilmore e Gomory, 1961] e a *Arc Flow* de [de Carvalho, 1999]. Outros trabalhos que utilizam a formulação de Gilmore e Gomory são [Vance, 1998], [Belov e Scheithauer, 2006] e [Wei et al., 2020]. Já em relação à formulação *Arc Flow* temos os trabalhos de [Brandão e Pedroso, 2016], [Delorme e Iori, 2020] e [de Lima et al., 2022].

Por outro lado, em algoritmos heurísticos, temos o trabalho de [Fleszar e Hindi, 2002] com a meta-heurística *Variable Neighborhood Search* (VNS) aplicada ao BPP. Neste trabalho, os autores utilizam uma variação da heurística *Minimum Bin Slack*, proposta por [Gupta e Ho, 1999], como solução inicial da VNS. Essa combinação se provou bastante poderosa, conseguindo encontrar uma solução ótima para 1329 instâncias das 1370 analisadas.

Em seguida, [Loh et al., 2008] utilizaram a meta-heurística *Weight Annealing* para o BPP, conseguindo encontrar uma solução ótima em todas as instâncias estudadas por [Fleszar e Hindi, 2002]. Já [Castelli e Vanneschi, 2014] propuseram uma heurística para o BPP chamada BPHS, utilizada como solução inicial da VNS. Apesar dos bons resultados, essa combinação se mostrou inferior em qualidade das soluções quando comparada com os dois estudos anteriores.

Por fim, [Buljubašić e Vasquez, 2016] e [Kucukyilmaz e Kiziloz, 2018] propuseram, para o BPP, as meta-heurísticas *Consistent neighborhood search* (CNS) e *Island-parallel Grouping Genetic Algorithm* (IPGGA), respectivamente. Ambas as meta-heurísticas foram estudadas com o mesmo conjunto de instâncias, sendo que a CNS encontrou uma solução ótima para 1315 instâncias das 1318 estudadas, enquanto a IPGGA encontrou uma solução ótima para 1313 instâncias das 1318 estudadas.

https://proceedings.science/p/157378?lang=pt-br

1.1. Nossa Contribuição

Neste artigo, apresentamos alguns detalhes de implementação da heurística *Two-By-Two* e das vizinhanças da VNS que foram apresentadas no trabalho preliminar desse artigo [Silva et al., 2021], de forma a conseguirmos complexidades de tempo melhores que as implementações diretas. Além disso, também propomos uma abordagem math-heurística para o CBPP, que mescla programação linear e as meta-heurísticas VNS e *Greedy Randomized Adaptive Search* (GRASP), que, pelos nossos resultados computacionais, se mostrou superior à VNS.

1.2. Organização do Artigo

O artigo é organizado da seguinte forma. Na Seção 2, apresentamos uma definição formal do CBPP, definimos algumas notações úteis e provamos uma proposição que proporciona uma maneira eficiente de lidar com as restrições de cores. Na Seção 3, introduzimos um algoritmo auxiliar que é usado como sub-rotina em alguns de nossos algoritmos. A heurística *Two-by-Two*, a VNS e a math-heurística são apresentadas nas Seções 4, 5 e 6, respectivamente. Por fim, os resultados computacionais e a conclusão são feitas nas Seções 7 e 8.

2. Definição do Problema e Notações

A seguir definimos o CBPP e apresentamos algumas das notações utilizadas.

2.1. Problem do Empacotamento Colorido

No Problema do Empacotamento Colorido (CBPP), dados recipientes de capacidade L e um conjunto de itens $I = \{1, ..., n\}$, onde cada item i possui um tamanho $0 < s_i \leq L$ e uma cor $c_i \in \mathbb{N}$, temos por objetivo empacotar cada item $i \in I$ nos recipientes $\{B_1, B_2, ..., B_k\}$, com k mínimo de forma a satisfazer as restrições de capacidade e de cores. Isto é, para cada $1 \leq r \leq k$, temos que satisfazer a restrição de capacidade $\sum_{i \in B_r} s_i \leq L$, e satisfazer a restrição de cores: existe uma permutação π de B_r, tal que para todos os pares de itens (i, j) em B_r, se $c_i = c_j$, então $\pi(i) \neq \pi(j) + 1$, ou seja, dois itens de mesma cor não podem ser empacotados lado a lado em um mesmo recipiente.

Um subconjunto de itens B_r que satisfaz as restrições de capacidade e de cores é chamado de *padrão*, e uma solução para o CBPP é um conjunto de padrões tal que cada item pertence a exatamente um padrão. Visto isso, uma possível formulação para o CBPP é a formulação de [Gilmore e Gomory, 1961] (proposta inicialmente para o BPP):

$$
\begin{aligned}
(\mathrm{P}) \quad & minimize \quad && \sum_{p \in \mathcal{P}} \lambda^p && \\
& sujeito\ a \quad && \sum_{p \in \mathcal{P} \wedge i \in p} \lambda^p = 1, \quad \forall i \in I && (1) \\
& && \lambda^p \in \{0, 1\}, \qquad \forall p \in \mathcal{P} && (2)
\end{aligned}
$$

onde $\mathcal{P}$ é o conjunto de todos os padrões, e temos uma variável binária λ^p para cada $p \in \mathcal{P}$, que é igual a 1 se e somente se o padrão p é utilizado. A função objetivo é minimizar a quantidade de padrões utilizados.

2.2. Notações

Considerando uma solução viável fixa para o CBPP, utilizamos ao longo deste artigo as seguintes notações: o item i está empacotado no recipiente $b(i)$; $cap(i)$ é a capacidade utilizada do recipiente $b(i)$, isto é, a soma dos tamanhos dos itens de $b(i)$; $res(r)$ é a capacidade residual do recipiente B_r, isto é, $res(r) = L - \sum_{j \in B_r} s_j$; e $tight(r)$ é a cor, se existir, que se adicionada em

B_r leva a violação das restrições de cores. Se tal cor não existir, tomamos $tight(r) = \emptyset$. Como veremos, pode existir no máximo uma cor com tal propriedade.

Uma observação interessante é dada pelo Lema 1, que mostra uma condição necessária e suficiente para a existência da permutação π dos itens de B_r que satisfaz as restrições de cores, sem a necessidade de construí-la. Dado a simplicidade de tal condição, nossos algoritmos focam apenas em garantir a sua validade. Além disso, podemos construir uma permutação π de B_r usando o algoritmo resultante da prova por indução apresentada no Lema 1, que, com auxílio de uma árvore binária de busca balanceada, nos permite encontrar π com complexidade de tempo $\Theta(|B_r| \log |B_r|)$. Se fizermos isso para $1 \leq r \leq k$, encontramos as permutações de todos os recipientes de uma solução dada em $O(n \log n)$.

Lema 1: Seja $1 \leq r \leq k$ e tome $S = B_r$. Seja f a cor mais frequente de S, S_f o subconjunto de itens em S com a cor f e S_n o subconjunto de itens em S com as demais cores. Então, há uma permutação π de B_r que satisfaz as restrições de cores se e somente se $|S_f| \leq |S_n| + 1$. Note que quando temos a igualdade $|S_f| = |S_n| + 1$, a cor f é única e $tight(r) = f$.

Prova: Suponha por contradição que a permutação π existe com $|S_f| > |S_n| + 1$. Há $|S_n| - 1$ posições entre itens de S_n que um item de S_f pode ocupar, além de uma posição antes do primeiro e uma posição depois do último. Mas $|S_f| > |S_n| + 1$, então pelo princípio da casa dos pombos, existe ao menos dois itens de S_f lado a lado, levando a uma contradição.

Visto isso, mostraremos por indução em $|S|$ como construir a permutação π quando $|S_f| \leq |S_n| + 1$. Se $|S| \leq 2$, não existem dois itens de mesma cor, portanto qualquer permutação é válida. Logo, mostraremos para $|S| \geq 3$, assumindo que o resultado vale para $|S'| < |S|$ por hipótese de indução.

Tome $i \in S_f$, $j \in S_n$ e $S' = S \setminus \{i, j\}$. Além disso, seja g a cor mais frequente de S', S'_g o subconjunto de itens em S' com a cor g e S'_n o subconjunto de itens em S' com as demais cores. Se g é igual a f, claramente $|S'_g| \leq |S'_n| + 1$. Do contrário, note que $|S'_g| \leq |S_f|$ e $(S_f \setminus \{i\}) \subseteq S'_n$. Então, temos que $|S'_g| \leq |S_f| \leq |S'_n| + 1$, portanto a inequação $|S'_g| \leq |S'_n| + 1$ segue.

Dessa forma, podemos utilizar a hipótese de indução em S' e obter uma permutação π'. Por fim, podemos construir π adicionando i e j ao final de π', pois i e j tem cores diferentes, e ao menos um deles tem cor diferente do último item de π'. ■

3. Algoritmo Auxiliar

Em vários dos nossos algoritmos, temos um conjunto de itens I ordenados em ordem não-crescente de tamanho e um conjunto de recipientes $\mathcal{B}$ ordenados em ordem não-crescente de capacidade residual, e temos que encontrar, para cada $i \in I$, o recipiente $B_r \in \mathcal{B}$ mais cheio em que i pode ser empacotado. Em alguns casos, pode acontecer de i já estar empacotado em algum recipiente de $\mathcal{B}$, e nesta situação exigimos que $B_r \neq b(i)$. Se para um item $i \in I$ não existir um recipiente B_r que satisfaz essas restrições, então a resposta para item i é 0 (sendo este um índice inválido). Visto isso, mostraremos um algoritmo com complexidade de tempo $\Theta(|I| + |\mathcal{B}|)$ para resolver esse problema.

O objetivo deste algoritmo é encontrar o conjunto de pares $\{(i, to(i)) : 1 \leq i \leq |I|\}$, onde $to(i)$ é o índice do recipiente mais cheio em que o item i pode ser empacotado respeitando as restrições dadas. Uma relaxação disto consiste em encontrar um mapeamento de I para $\mathcal{B}$, tal que, para todo i, temos que i não cabe em $to(i) + 1$, ou seja, ignorando as restrições de cores e que i pode pertencer a $B_r \in \mathcal{B}$. Um algoritmo para resolver essa relaxação é dado no Algoritmo 1.

Agora mostraremos como corrigir as respostas inválidas da relaxação em $\Theta(|I| + |\mathcal{B}|)$. Note que, se a resposta para i é inválida, então precisamos decrementar $to(i)$ até encontrar o primeiro movimento válido, visto que a relaxação garante que i não cabe em nenhum recipiente de

índice maior que $to(i)$.

Temos duas possibilidades de respostas inválidas. A primeira delas é se $to(i)$ é igual ao índice do recipiente $b(i)$, e podemos corrigir simplesmente decrementando $to(i)$. A segunda possibilidade é se $tight(to(i)) = c_i$, e podemos corrigir encontrando o primeiro recipiente anterior $prev[to(i)]$ com uma cor *tight* diferente de $to(i)$. Para tornar esse processo eficiente, podemos pré-computar em $O(|\mathcal{B}|)$ um vetor $prev$ com a resposta para todos os recipientes usando a programação dinâmica apresentada no Algoritmo 1.

Note que precisamos consertar a mesma resposta (inválida) da relaxação no máximo três vezes, pois a primeira possibilidade pode ocorrer apenas uma vez e a segunda possibilidade não pode ocorrer duas vezes consecutivas. Logo, temos que o Algoritmo Auxiliar, mostrado no Algoritmo 1, tem complexidade de tempo linear $\Theta(|I| + |\mathcal{B}|)$.

Algorithm 1 Algoritmo Auxiliar

```
1:  function AlgoritmoAuxiliar(I, B)
2:      # Calcula a solução da relaxação
3:      to ← vetor de tamanho |I|
4:      last ← 0
5:      for i ← 1 to |I| do
6:          while last + 1 ≤ |B| and i cabe em B[last + 1] do
7:              last ← last + 1                  # Essa linha é executada no máximo |B| vezes.
8:          to(i) ← last
9:
10:     # Calcula o vetor prev
11:     prev ← vetor de tamanho |B| preenchido com 0
12:     for j ← 2 to |B| do
13:         if tight(j) ≠ ∅ and tight(j) = tight(j − 1) then
14:             prev[j] ← prev[j − 1]
15:         else
16:             prev[j] ← j − 1
17:
18:     # Corrigi a relaxação
19:     for i ← 1 to |I| do
20:         for k ← 1 to 2 do
21:             if to(i) = b(i) then
22:                 to(i) ← to(i) − 1
23:             if to(i) ≠ 0 and tight(to(i)) = c_i then
24:                 to(i) ← prev[to(i)]
25:
26:     return {(i, to(i)) : 1 ≤ i ≤ |I|}
```

4. Heurística de Construção Inicial

Em nosso trabalho preliminar [Silva et al., 2021], apresentamos adaptações das heurísticas de construção inicial *Best Fit Decreasing* (BFD) e *Minimum Bin Slack* (MBS) do BPP para o CBPP, além de propormos duas novas heurísticas de construção inicial chamadas *Hard BFD* e *Two-by-Two* também para o CBPP. Conforme os resultados do nosso trabalho preliminar, dentre as heurísticas

estudadas, a *Two-by-Two* é a que apresenta soluções de maior qualidade. Dito isso, apresentamos a mesma a seguir, desta vez com o pseudo-código de uma implementação melhor que a direta em $\Omega(n^3)$.

A heurística *Two-by-Two* consiste em um processo iterativo, onde em cada iteração criamos um recipiente B_r e fazemos movimentos, ou seja, escolhas de itens a serem empacotados, que minimizam uma função f. Primeiro, escolhemos um item para colocar em B_r. Em seguida, escolhemos em cada passo o melhor entre colocar um ou dois itens em B_r. Se não houver mais movimentos válidos, então começamos uma nova iteração.

A função f é definida da seguinte forma. Seja S o conjunto de itens não empacotados, g a cor mais frequente de S, S_g o subconjunto de itens em S com a cor g e S_n o subconjunto de itens em S com as demais cores. A função f simula a execução de um movimento e computa o valor $\left(\frac{res(r)}{L}\right)^2 + |S| \cdot \left(\frac{|S_g|}{|S|} - \frac{|S_g^0|}{|S^0|}\right)^2$, onde S^0 e S_g^0 são os valores de S e S_g no começo do algoritmo. No primeiro termo de f, temos o fator $res(r)$ que nos faz priorizarmos os movimentos que preencherem bem o recipiente B_r. Já o segundo termo de f é encarregado de impedir o surgimento de grandes discrepâncias de cores. Um pensamento intuitivo é que se escolhermos uma amostra aleatória de recipientes da solução ótima, então a proporção de cores da amostra deve ser próxima de $\frac{|S_g^0|}{|S^0|}$. Portanto, o segundo termo de f penaliza a diferença para essa proporção, sendo o fator multiplicativo $|S|$ necessário para que tal penalidade não seja irrelevante no começo do algoritmo.

Uma observação é que para encontrarmos os dois itens i e j que minimiza f, podemos fixar se o item $i \in S_n$ ou $i \in S_g$, e fazer o mesmo para o item j. Pois, desta forma o segundo termo de f se torna constante, e o par de itens (i, j) que minimizam f é aquele que melhor preencher o recipiente, logo podemos usar o Algoritmo Auxiliar para encontrá-los. O Algoritmo 2 é o pseudo-código da heurística *Two-by-Two*, com complexidade de tempo $\Theta(n^2)$.

Algorithm 2 Two-by-Two

$\mathcal{B} \leftarrow \emptyset$ ▷ Solução vazia
while $|I| > 0$ **do**
  $B_r \leftarrow$ um novo recipiente com o item i que minimiza f
  Remova i de I
  while $|I| > 0$ **do** # Aqui enchemos B_r até não haver mais movimentos válidos
    $bestMove \leftarrow$ melhor movimento envolvendo um item (se existir)
    for each escolha de $S_1, S_2 \in \{S_g, S_n\}$ **do** # Note que esse laço tem quatro iterações
      $\mathcal{B}' \leftarrow$ recipientes fictícios criados independentemente adicionando cada item $i \in S_1$ em B_r
      $moves \leftarrow$ AlgoritmoAuxiliar$(S_2, \mathcal{B}')$
      Avalie cada movimento encontrado acima, e atualize $bestMove$
    if $bestMove \neq \emptyset$ **then**
      Faça o melhor movimento, remova os itens usados de I e recalcule S, S_g e S_n
    else
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{B_r\}$
      break
return $\mathcal{B}$

5. Variable Neighborhood Search

A *Variable Neighborhood Search* (VNS) é uma meta-heurística proposta por [Mladenović e Hansen, 1997], composta por uma função objetivo f, um conjunto de vizinhanças $\{\mathcal{N}_1, ..., \mathcal{N}_k\}$ e

uma função *Shake*. A seguir apresentamos em maiores detalhes a VNS proposta na versão preliminar deste artigo [Silva et al., 2021].

5.1. Função Objetivo

A função objetivo f que utilizamos possui dois níveis: um objetivo primário que busca minimizar a quantidade de recipientes utilizados e um objetivo secundário que busca minimizar a ordem lexicográfica do vetor de capacidades residuais. O vetor de capacidades residuais é o vetor composto pelas capacidades residuais dos recipientes utilizados em ordem não-decrescente.

5.2. Vizinhanças

Seja x uma solução viável. A vizinhança $\mathcal{N}_k(x)$ é o conjunto de todas as soluções viáveis alcançáveis a partir de x ao se realizar um certo tipo de movimento. Um elemento desse conjunto é chamado *vizinho* de x.

A VNS realiza uma busca local em vizinhanças, como será explicado adiante. Um fator importante é a escolha da heurística utilizada na busca local. Em nossa VNS, utilizamos a heurística *Best Improvement*, que analisa todos os vizinhos e muda para o melhor vizinho (em relação à função objetivo). Além disso, veremos a seguir que o número de recipientes alterados em cada movimento de uma vizinhança é constante, portanto, a comparação de dois vizinhos pode ser feita por suas diferenças, o que leva tempo $O(1)$.

A nossa VNS tem vizinhanças com os movimentos: *Move Item*, *Swap Items*, *Swap And Move*, e *Move Two to One*, explicadas a seguir em maiores detalhes.

Move Item Neighborhood: é uma vizinhança que move um item de um recipiente para outro. Um possível algoritmo para essa vizinhança é adaptar, para o CBPP, o algoritmo do BPP usado por [Fleszar e Hindi, 2002], que tem complexidade de tempo $O(n^2)$. No entanto, como estamos buscando apenas o melhor movimento, é possível implementar essa vizinhança com complexidade de tempo $\Theta(n)$. Para tal, basta notar que o movimento buscado é justamente o melhor entre os movimentos devolvidos pelo Algoritmo Auxiliar.

Swap Items Neighborhood: é uma vizinhança que realiza a troca de dois itens de recipientes diferentes. A troca entre dois itens i e j, com $s_i \leq s_j$, entre os recipientes $b(i)$ e $b(j)$ é válida se $b(i) \neq b(j)$, $res(b(i)) + s_i - s_j \geq 0$ e os recipientes respeitarem as restrições de cores após a troca. Nesta vizinhança, ordenamos os itens em ordem não-decrescente de tamanho, com um critério de desempate onde, para $s_i = s_j$ e $i < j$, temos que a capacidade residual do recipiente do item i é menor ou igual que a capacidade residual do recipiente do item j, isto é, $res(b(i)) \leq res(b(j))$. Para cada item i, iteramos nos itens j na ordem reversa começando do índice k, onde k é o maior índice tal que $res(b(i)) + s_i - s_k \geq 0$, que pode ser encontrado com uma busca binária. O item k é o maior item que potencialmente pode ser trocado com i, e podemos provar que o melhor movimento para i fixo e $s_i \leq s_j$ é o primeiro j que encontramos tal que (i, j) é um movimento bom (i.e., que melhora a função objetivo). Esse algoritmo tem complexidade $O(n \log n + M)$, onde M é a quantidade de trocas válidas quando ignoramos as restrições de cores.

Move Two to One Neighborhood: é uma vizinhança que seleciona dois itens de recipientes diferentes para mover para um terceiro recipiente. A implementação direta dessa vizinhança tem complexidade de tempo $O(n^3)$, no entanto, é possível fazer melhor que isso. Para tal, movemos o item i para cada recipiente do conjunto $\mathcal{B} \setminus b(i)$, criando um conjunto $\mathcal{B}'$ de recipientes fictícios. Então, usamos o Algoritmo Auxiliar nos conjuntos $I \setminus \{i\}$ e $\mathcal{B}'$. Isso nos dá o melhor recipiente de

destino para os pares (i, j), onde $j \in I \setminus \{i\}$, com complexidade de tempo $\Theta(n)$. Se fizermos isso para cada $i \in I$, temos uma vizinhança com complexidade de tempo $\Theta(n^2)$.

Swap And Move Neighborhood: é uma vizinhança que selecionamos três itens, digamos i, j, k, de diferentes recipientes, trocamos de recipientes i e j, e movemos k para o recipiente $b(i)$. Se implementada de maneira completa, essa vizinhança tem complexidade de tempo $O(n^3)$, oferecendo um baixo custo-benefício. Portanto, implementamos uma versão mais restrita, em que, ao invés de testarmos todas as possibilidades de valores de k, testamos os movimentos que envolvem apenas o maior item que pode ser movido para $b(i)$. Usando a ideia do vetor $prev$, podemos implementar essa vizinhança com complexidade de $O(n^2 \cdot (\log n + |B_{max}|))$, onde $|B_{max}|$ é a quantidade máxima de itens em um recipiente.

5.3. Shake

A função *Shake* realiza movimentos aleatórios em uma solução, buscando por meio disso escapar de ótimos locais. Em particular, a nossa função Shake escolhe executar entre duas sub-rotinas com igual probabilidade. Na primeira sub-rotina realizamos 20 movimentos, entre mover um item ou trocar dois itens, com a restrição que cada recipiente pertença a no máximo um movimento. Já na segunda sub-rotina selecionamos aleatoriamente dois recipientes, e reempacotamos seus itens em uma ordem aleatória com a heurística *Best Fit*.

5.4. Algoritmo

O algoritmo da nossa VNS começa com $k = 1$ e recebe uma solução viável x construída pela nossa heurística de construção inicial *Two-by-Two*. Em cada passo, efetuamos uma busca local em x usando a vizinhança $\mathcal{N}_k$ até alcançarmos uma solução x' que é ótimo local. Se x' é melhor que x, então fazemos $x \leftarrow x'$ e $k \leftarrow 1$, do contrário, fazemos $k \leftarrow k+1$. Esse processo é executado até $k = |\mathcal{N}| + 1$. Neste ponto, a rotina *Shake* é executada para gerar uma pertubação em x, e então iniciamos uma nova iteração com $k = 1$.

Um critério de parada é necessário para esse processo não ser executado indefinidamente. Em nosso caso, temos dois critérios de parada. O primeiro deles é se a solução atingiu o limitante inferior L_1, onde $L_1 = \left\lceil \sum_{i \in I} \frac{s_i}{L} \right\rceil$, que é a quantidade mínima de recipientes que precisamos para empacotar os itens se pudéssemos fracioná-los. Já o segundo critério é um limite de tempo de execução t_{max}. Ao atingir algum desses critérios, o algoritmo devolve a melhor solução dentre a solução inicial e as soluções finais de cada busca local.

5.5. Ordenação das Vizinhanças

Veja que a VNS efetua uma busca local na vizinhança $\mathcal{N}_k$ somente se a solução atual for ótima local nas vizinhanças $\{\mathcal{N}_1, ..., \mathcal{N}_{k-1}\}$. Portanto, usualmente, na tentativa de reduzir o tempo computacional, as vizinhanças de baixo custo computacional precedem as de alto custo. Em nossa VNS, os melhores resultados foram alcançados utilizando as vizinhanças na exata ordem que apresentamos neste artigo.

6. Math-heurística

A nossa math-heurística é uma mesclagem de programação linear com as meta-heurísticas VNS e *Greedy Randomized Adaptative Search Procedure* (GRASP), sendo a GRASP uma meta-heurística semi-gulosa que foi inicialmente descrita por [Hart e Shogan, 1987]. Nossa math-heurística consiste em um processo de três estágios, executados intercaladamente até encontrarmos uma solução com valor igual ao limitante inferior L_1, ou até atingirmos um limite de tempo t_{max}. Durante esses estágios mantemos a solução incumbente F_{best}, isto é, a melhor solução conhecida, e também um conjunto de padrões $\mathcal{P}'$ formado pela união dos recipientes das soluções geradas. Na

primeira iteração, F_{best} advém da nossa heurística de construção inicial *Two-by-Two* e $\mathcal{P}' = F_{best}$. A seguir explicamos os três estágios.

No primeiro estágio executamos a VNS por t_1 segundos, partindo da solução incumbente F_{best}. Além disso, toda vez que atingirmos um ótimo local em alguma vizinhança, acrescentamos os padrões da solução atual em $\mathcal{P}'$. Com esse estágio tentamos encontrar padrões bons.

No segundo estágio resolvemos, com o solver comercial Gurobi, a relaxação linear restrita da formulação apresentada na Seção 2, isto é, consideramos o programa linear obtido a partir da remoção das restrições de integralidade de λ e considerando apenas as variáveis associadas ao subconjunto de padrões $\mathcal{P}' \subseteq \mathcal{P}$. Com esse estágio temos por objetivo filtrar os padrões de $\mathcal{P}'$, usando a intuição que os padrões utilizados, de forma parcial ou completa, pela solução ótima λ^* desse modelo restrito devem compor uma solução inteira de boa qualidade.

No terceiro estágio executamos a GRASP por t_2 segundos, e temos por objetivo arredondar a solução da relaxação, ponderando o caráter guloso com o caráter aleatório. A GRASP consiste em um processo de múltiplos reinícios de dois passos: um de construção de uma solução e outro de busca local.

Na construção de uma solução, recebemos a solução ótima λ^* da relaxação linear e um parâmetro $0 < \alpha < 1$. Seja S o conjunto de padrões p com $\lambda^*_p > 0$. Nesse passo, construímos incrementalmente uma solução F, em que $F = \emptyset$ inicialmente. Em uma iteração construímos uma lista de candidatos $L \subseteq S$, onde $|L| = \lceil \alpha |S| \rceil$ e L é composto pelos padrões p com os maiores valores de λ^*_p. Em seguida, escolhemos aleatoriamente um padrão $p \in L$ para ser adicionado em F, que será uma escolha válida se, e somente se, nenhum item $i \in p$ pertence a algum padrão $p' \in F$.

Ao final, pode ser que mesmo após analisarmos todos os padrões de S, ainda exista um subconjunto de itens $I' \subseteq I$ que não pertence a nenhum padrão de F. Neste caso, empacotamos I' em F usando uma versão aleatorizada da heurística *Two-by-Two*. Para inserir aleatoriedade na *Two-by-Two*, em vez de selecionarmos sempre o melhor movimento, selecionamos um movimento aleatório dentre a proporção α dos movimentos mais bem avaliados. Ao final, F será uma solução válida para o problema.

Em seguida, realizamos o passo de busca local em F, usando a parte de busca em vizinhanças da nossa VNS. Então, armazenamos os padrões de F em $\mathcal{P}'$ e atualizamos F_{best} se necessário. Por fim, se tivermos atingirmos o tempo limite t_3, vamos para o primeiro estágio, do contrário voltamos ao primeiro passo.

Como uma otimização, observamos ser vantajoso executar a VNS apenas na parte da solução criada pela *Two-by-Two* Aleatorizada, havendo duas razões para isso. A primeira razão é que os padrões de S usualmente preenchem bem os recipientes, logo é improvável que a busca local opte por alterá-los sem alguma rotina de pertubação, como a *Shake*. Já a segunda razão é que a busca local se torna muito mais rápida, dado que I' usualmente possui um tamanho muito menor quando comparado com I, portanto, tal otimização nos permite executar muito mais iterações da math-heurística.

7. Resultados Computacionais

Nessa seção apresentamos as instâncias utilizadas em nossos experimentos computacionais e fazemos uma comparação entre a VNS e a math-heurística.

Todos os resultados foram obtidos com computador com um processador Intel® Xeon® CPU E5-2630 v4 @ 2.20GHz, com 64 GB de RAM, com sistema operacional Ubuntu 18.04.6 LTS (64 bits) e utilizando a linguagem C++17 com o compilador GCC 7.5.0. Para resolver a relaxação linear utilizamos o solver comercial Gurobi na versão 9.1.2 executando em *single thread*.

https://proceedings.science/p/157378?lang=pt-br

7.1. Instâncias

As instâncias utilizadas nos experimentos são divididas em três classes. A primeira classe é uma adaptação das instâncias AI de [Delorme et al., 2016], com 202 e 403 itens, do BPP para o CBPP. Já as outras duas classes são instâncias geradas aleatoriamente, com número de itens $n \in \{102, 402, 2001, 10002\}$ e capacidade do recipiente $L \in \{10^3+1, 10^6+1\}$. A segunda classe é chamada *Random10to80* e tem itens com tamanhos escolhidos aleatoriamente no intervalo $[\frac{L}{10}, \frac{4L}{5}]$. Por fim, a última classe é chamada *Random25to50*, e consiste em $\frac{n}{3}$ triplas de itens com soma igual a L, onde cada item tem um tamanho escolhido aleatoriamente no intervalo $[\frac{L}{4}, \frac{L}{2}]$.

A coloração dos itens foi realizada de duas maneiras. O primeiro tipo é a Coloração Uniforme, onde cada instância possui uma quantidade de cores distintas $Q \in \{2, 5, 20\}$, e a cor de cada item é escolhido aleatoriamente no intervalo $[0, Q)$. Há versões de todas as instâncias com esse tipo de coloração, valendo destacar que tomamos o cuidado nas instâncias AI e *Random25to50* para que a coloração seja realizada de forma que ao menos uma solução ótima do BPP se mantenha viável para o CBPP. Já a segunda coloração é chamada Q2 *Half*, utilizada somente nas instâncias *Random25to50*. Na mesma, dado as $\frac{n}{3}$ triplas de itens, colorimos os dois itens mais pesados da tripla com cor 1, e o item restante com cor 0. A ideia dessa coloração é prejudicar as heurísticas iniciais, visto que facilmente podemos escolher triplas de itens que somam L mas não deveriam ir juntos, e para cada $k \geq 2$ itens de cor 0 que empacotarmos juntos em um recipiente, resulta em $k-1$ itens de cor 1 empacotados sozinhos.

Para as instâncias AI geramos uma instância para cada possível combinação de parâmetros, enquanto para as instâncias aleatórias geramos cinco instâncias para cada combinação. Por fim, vale destacar que, por construção, as instâncias AI e *Random25to50* possuem solução ótima igual a L_1, tanto no BPP como no CBPP, sendo as instâncias AI o conjunto da literatura do BPP mais difícil de se encontrar uma solução ótima.

7.2. Comparação entre a VNS e a Math-heurística

A VNS e a math-heurística (MH) foram executadas usando t_{max} igual a 60 segundos. Na MH utilizamos $\alpha = 0.3$, $t_1 = 5$ e $t_2 = 1$. Além disso, tanto na VNS como na MH utilizamos como solução inicial a solução dada pela *Two-by-Two*. Ademais, por questão de espaço, mostramos os resultados agrupando alguns tipos de coloração, e os dois tipos de capacidade do recipiente.

Na Tabela 1, as colunas "Diferença Média Inicial" e "Diferença Média Final" indicam as diferenças médias dos valores das soluções iniciais e finais para o limitante inferior L_1, e "Diferença Máxima Final" indica a diferença máxima de uma solução final para o limitante inferior L_1. Além disso, destacamos em negrito as menores diferenças médias de cada linha na tabela.

Como podemos ver, as nossas abordagens são muito boas, pois resolvem todas as instâncias *Random25to50* de 102 e 402 itens de forma ótima, e encontram soluções usando no máximo um recipiente extra em relação à solução ótima para 2001 itens. Além disso, as diferenças médias das soluções finais são menores que 3 na math-heurística, sendo maior justamente nas instâncias *Random10to80*, onde provavelmente L_1 é um limitante inferior ruim. Ademais, vale destacar as instâncias *Random25to50* de 10002 itens com Q2 *Half*, onde a diferença média das soluções inicias é de 81 recipientes, e nossas abordagens diminuiram essa diferença para no máximo 1.

Por fim, a nossa math-heurística é superior a VNS, conseguindo resolver de forma ótima 129 das 150 instâncias AI de 202 itens e 74 das 150 instâncias AI de 403 itens, além de conseguir diferenças médias finais menores nas *Random10to80*.

8. Conclusão e Trabalhos Futuros

As nossas abordagens se mostraram bastante efetivas, nos possibilitando encontrar soluções muito próximas ao valor de uma solução ótima mesmo para instâncias do problema com grande

					VNS			MH		
Classes	Número de Itens	Tipos de Coloração	Total	Diferença Média Inicial	Diferença Média Final	Diferença Máxima Final	Tempo (s)	Diferença Média Final	Diferença Máxima Final	Tempo (s)
AI	202	Q2	50	1,18	1,00	1	60,00	**0,22**	1	24,19
		Q5, Q20	100	1,00	1,00	1	60,00	**0,10**	1	15,28
	403	Q2	50	1,16	1,00	1	60,00	**0,54**	1	40,06
		Q5, Q20	100	1,00	1,00	1	60,00	**0,49**	1	38,75
Random 25to50	102	Q2 Half	10	1,20	**0,00**	0	0,33	**0,00**	0	0,24
		Q2, Q5, Q20	30	1,00	**0,00**	0	0,10	**0,00**	0	0,09
	402	Q2 Half	10	2,40	**0,00**	0	0,26	**0,00**	0	0,18
		Q2, Q5, Q20	30	1,10	**0,00**	0	0,18	**0,00**	0	0,25
	2001	Q2 Half	10	15,10	**0,50**	1	40,36	0,70	1	47,68
		Q2, Q5, Q20	30	2,40	0,53	1	34,89	**0,50**	1	34,81
	10002	Q2 Half	10	81,00	**0,90**	1	59,34	1,00	1	60,00
		Q2	10	6,30	1,90	4	55,77	**1,10**	4	54,43
		Q5, Q20	20	13,75	1,55	3	60,02	**1,20**	3	60,01
Random 10to80	102	Q2	10	3,50	2,00	5	48,48	**1,70**	4	42,78
		Q5, Q20	20	1,25	0,55	2	28,96	**0,50**	2	25,03
	402	Q2	10	5,40	4,30	9	54,00	**2,90**	9	48,51
		Q5, Q20	20	2,80	2,25	6	55,77	**1,55**	6	48,93
	2001	Q2	10	3,20	2,80	8	60,00	**0,70**	2	42,63
		Q5, Q20	20	5,40	3,70	5	60,00	**0,45**	1	31,01
	10002	Q2	10	3,70	2,20	4	60,00	**1,10**	2	44,89
		Q5, Q20	20	14,40	6,15	22	51,91	**0,90**	2	41,76

Tabela 1: Comparação da VNS com a math-heurística

quantidade de itens. Acreditamos que uma parte de tais resultados se devem ao fato da heurística *Two-by-Two* conseguir lidar bem com as restrições de cores, além de possuirmos vizinhanças eficientes tanto em qualidade como em complexidade de tempo. Já a outra parte, acreditamos se dever ao fato que formulação de Gilmore e Gomory possui uma relaxação forte para o *Bin Packing Problem* (BPP), sendo intuitivo que o mesmo deve valer para o CBPP. Em uma relaxação forte, esperamos que grande parte das variáveis utilizadas pela relaxação façam parte de soluções de alta qualidade, o que nos proporcionou um excelente método de filtragem de padrões para serem utilizados na construção inicial da GRASP.

Como trabalhos futuros, é interessante conseguirmos limitantes inferiores nas instâncias *Random10to80*, visto que o limitante inferior L_1 pode ser ruim e talvez estejamos próximos à solução ótima sem sabermos. Contudo, essa não deve ser uma tarefa fácil, em especial nas instâncias com $L = 10^6 + 1$, visto que geralmente as relaxações do BPP e suas variantes, costumam utilizar rotinas com complexidade de tempo $\Omega(n \cdot L)$.

Referências

Belov, G. e Scheithauer, G. (2006). A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171(1):85–106.

Borges, Y. e Schouery, R. (2020). Modelos pseudo polinomiais para o problema do empacotamento colorido. In *Anais do V Encontro de Teoria da Computação*, p. 37–40.

Brandão, F. e Pedroso, J. P. (2016). Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69:56–67.

Buljubašić, M. e Vasquez, M. (2016). Consistent neighborhood search for one-dimensional bin packing and two-dimensional vector packing. *Computers & Operations Research*, 76:12–21.

Böhm, M., Sgall, J., e Veselý, P. (2014). Online colored bin packing. In *International Workshop on Approximation and Online Algorithms*, p. 35–46.

Castelli, M. e Vanneschi, L. (2014). A hybrid harmony search algorithm with variable neighbourhood search for the bin-packing problem. *2014 Sixth World Congress on Nature and Biologically Inspired Computing (NaBIC 2014)*, p. 1–6.

de Carvalho, J. M. V. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86(0):629–659.

de Lima, V. L., Iori, M., e Miyazawa, F. K. (2022). Exact solution of network flow models with strong relaxations. *Mathematical Programming*, p. 1–34.

Delorme, M. e Iori, M. (2020). Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing*, 32(1):101–119.

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.

Dósa, G. e Epstein, L. (2014). Colorful bin packing. *Proceedings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory*, p. 170–181.

Fleszar, K. e Hindi, K. S. (2002). New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29(7):821–839.

Gilmore, P. C. e Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859.

Gupta, J. N. D. e Ho, J. C. (1999). A new heuristic algorithm for the one-dimensional bin-packing problem. *Production Planning & Control*, 10(6):598–603.

Hart, J. P. e Shogan, A. W. (1987). Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6(3):107–114.

Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, p. 85–103. Springer US.

Kucukyilmaz, T. e Kiziloz, H. E. (2018). Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem. *Computers & Industrial Engineering*, 125:157–170.

Loh, K.-H., Golden, B., e Wasil, E. (2008). Solving the one-dimensional bin packing problem with a weight annealing heuristic. *Computers & Operations Research*, 35(7):2283–2291.

Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Silva, R. F., Borges, Y., e Schouery, R. (2021). Heurísticas para o problema do empacotamento colorido. In *Anais do VI Encontro de Teoria da Computação*, p. 34–37. SBC.

Vance, P. H. (1998). Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimization and Applications*, 9(3):211–228.

Wei, L., Luo, Z., Baldacci, R., e Lim, A. (2020). A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*, 32(2):428–443.

https://proceedings.science/p/157378?lang=pt-br