Pesquisa Operacional para o Desenvolvimento
Vol. XX, n. YY, pp. ZZ-ZZ, ANO: XXXX

ISSN 1984-3534

METAHEURÍSTICA VNS PARA O PROBLEMA DE CARREGAMENTO DE PALETES DO DISTRIBUIDOR [1]

Gustavo Veigel Vieira[a] [*], **Alexandre Checoli Choueiri**[a]

[a] Departamento de Engenharia de Produção
UFPR, Curitiba-PR , Brasil

Recebido xx/xx/xxxx, aceito xx/xx/xxxx

RESUMO

Neste artigo o "Problema de Carregamento de Paletes do Distribuidor" *(Distributor's Pallet Loading Problem - DPLP)* de uma grande empresa do setor alimentício é abordado. Uma metaheuristica é utilizada para resolver o problema, considerando uma transformação do DPLP em 2 sub-problemas: o *Bin Packing*, onde camadas completas de caixas semelhantes formam os itens, e os paletes representam os *bins*, e um problema de carregamento em duas dimensões com as caixas restantes. Um algoritmo de Busca em Vizinhança Variável (VNS) foi desenvolvido para resolver o *Bin Packing*, e uma heurística gulosa para o carregamento 2D. Resultados computacionais indicam que o VNS proposto encontra soluções ótimas para grande parte do conjunto de instâncias do BPP do repositório OR-Library de forma rápida, e representações gráficas dos padrões de carregamento para a heurística 2D indicam um bom aproveitamento da área dos paletes. Também foram realizados testes em instâncias do estudo de caso, indicando que o método é eficaz para a utilização prática na empresa.

Palavras-chave: Meta-heurística, Carregamento 2D, VNS, Problema do Empacotamento, Picking.

ABSTRACT

This article addresses a real case of the Distributor's Pallet Loading Problem (DPLP) in a large company in the food sector. The approach adopted starts from the transformation of DPLP into 2 other problems: a Bin Packing, where complete layers of similar boxes form the items whereas the pallets represent the bins, and a two-dimension loading problem with the remaining boxes. A Variable Neighborhood Search (VNS) algorithm was developed to solve the Bin Packing and a greedy heuristic was designed to solve the 2D loading. Computational results indicate that VNS finds optimal solutions for a large part of the Bin Packing instances from the OR-Library repository in a low computational time (seconds), and graphical representations of loading patterns for the 2D heuristic indicate good use of the pallet area.

Keywords: Metaheuristic, 2D Packing, Variable Neighborhood Search, Bin Packing Problem, Picking.

[*] Autor para correspondência. E-mail: gusveigel@gmail.com

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

1. Introdução

A logística de mercadorias entre centros de distribuição e clientes é uma etapa crucial na logística empresarial. Dentre os custos dessa atividade, a maior parcela se deve ao transporte [16]. O transporte se desdobra nas atividades de roteirização e carregamento de materiais, sendo esta última uma prioridade para empresas que trabalham com altos volumes. Para que a logística de transportes seja eficiente, é necessário que ambas as decisões (carregamento e roteirização) sejam tomadas de forma simultânea [3].

Estudos indicam que uma otimização unificada entre roteirização e carregamento geram os melhores resultados [3], porém na prática, o problema é majoritariamente abordado em duas etapas distintas: o problema de roteirização de veículos *(Vehicle Routing Problem - VRP)* e o problema de carregamento de paletes *(Pallet Loading Problem - PLP)* ou carregamento de contêineres (*Container Loading problem - CLP*).

De forma geral, o processo é iniciado com a determinação do conjunto de rotas e caminhões que atenderão a demanda dos clientes, considerando restrições de janela de tempo *(Vehicle Routing Problem with Time Windows - VRPTW)*. Após a determinação das mesmas, as rotas são encaminhadas ao depósito onde os pedidos devem ser agrupados, empilhados em paletes e carregados nos veículos apropriados [35].

Nesse trabalho abordamos um estudo de caso de uma grande empresa do ramo alimentício, em que a otimização de rotas é previamente realizada desconsiderando o arranjo da carga (paletes) dentro dos veículos, o que acaba gerando infactibilidades rota-carregamento, acarretando retrabalho e atrasos nos pedidos. O processo funciona da seguinte forma: o centro de distribuição recebe diariamente uma lista de pedidos com os produtos congelados e resfriados a serem carregados em paletes, para que estes sejam alocados nos caminhões que realizarão as entregas. Produtos resfriados e congelados não podem ser carregados em um mesmo palete. O tipo de caminhão, sua carga e rota são decididos previamente através de um otimizador. Cabe aos operadores do centro de distribuição realizar a montagem dos paletes e seu carregamento nos caminhões. Para verificar se a escolha do caminhão é factível, o otimizador leva em consideração somente o peso total da carga versus a capacidade do caminhão.

Na operação de coleta dos produtos (*picking*) para montagem dos paletes, os colaboradores são orientados a realizar a menor rota possível de coleta nos centros de distribuição. Esse tipo de abordagem prioriza um *picking* mais rápido e dinâmico, entretanto, menos eficiente na otimização dos paletes. Como consequência, as caixas de um mesmo produto são carregadas sequencialmente durante a montagem dos paletes, formando camadas de um mesmo produto. Somente ao tentar alocar os paletes no veículo os operadores conseguem verificar a factibilidade da alocação. Em muitos casos os pedidos não cabem no veículo, o que implica no cancelamento da entrega e alocação da rota em um veículo de maior capacidade no dia seguinte.

Neste estudo foi desenvolvido um método para refinar a verificação de factibilidade carga-veículo, por meio de uma transformação do problema original de carregamento de paletes em um problema de *Bin-Packing (BPP)* (considerando camadas completas), e um problema de carregamento em duas dimensões com as caixas restantes. Nos cenários de resolução do problema de carregamento de paletes utilizando-se o BPP com camadas completas, é comum que a tratativa das caixas restantes não sejam atreladas à solução do BPP, entretanto, em nosso trabalho criamos uma abordagem que torna ambas etapas diretamente ligadas, utilizando *bins* do BPP como base para montagem das camadas 2D. Além disso, levamos em consideração a priorização do *picking* rápido e dinâmico do estudo de caso, onde progressivamente nos afastamos da solução inicial afim de tentar preservar o dinamismo do *picking*. Para o BPP utilizamos a metaheuristica VNS (*Variable Neighborhood Search*) e uma heurística gulosa para o carregamento 2D.

O trabalho está dividido em 5 Seções. Na seção 2 uma revisão de literatura é realizada com os trabalhos correlatos sobre o tema. A Seção 3 contém o desenvolvimento, algoritmos e técnicas de tratamento de dados utilizadas. Os resultados computacionais são apresentados na Seção 4, e finalmente a conclusão compõe a Seção 5.

2. Trabalhos correlatos

Nesta seção apresentamos os trabalhos correlatos, divididos em 2 partes: aqueles referentes ao carregamento de paletes e ao BPP.

2.1. Carregamento de paletes

De acordo com [18], o problema de carregamento de paletes pode ser dividido em dois grandes problemas; o "Problema de empacotamento de paletes do fabricante" *(Manufacturer's Pallet Loading Problem - MPLP)* e o "Problema de empacotamento de paletes do distribuidor" *(Distributor's Pallet Loading Problem - DPLP)*.

O MPLP consiste em alocar o maior número de objetos retangulares (caixas) de mesma dimensão em um objeto maior (palete). Caso as caixas possam ser alocadas sem restrição de orientação, o MPLP pode ser decomposto em dois subproblemas: carregamento 2D, onde deve-se arranjar bidimensionalmente o número máximo de retângulos (faces das caixas) dentro do retângulo do palete, e carregamento 1D, onde as estruturas das possíveis camadas do palete já são definidas [25]. Com isso, o problema se resume em arranjar a melhor combinação de camadas no palete, se reduzindo a um "problema da mochila" *(Knapsack Problem - KP)*, cuja capacidade máxima é a altura permitida do palete, e o peso dos itens são as alturas das camadas. Se todas as caixas devem ser carregadas em um número mínimo de paletes, o problema se torna um *Bin Packing Problem* (BPP).

No DPLP, um distribuidor preenche o pedido de vários clientes em um mesmo palete, de forma que as caixas podem possuir dimensões variadas. O objetivo do problema é maximizar o volume alocado em um palete padrão, de modo a minimizar o número total de paletes utilizados no pedido. Esse problema também pode ser abordado como um "Problema de carregamento de containers" *(Container Loading Problem - CLP)*, como destacado por [5].

Ao realizar um agrupamento de caixas de mesma altura, o DPLP pode ser reduzido a um carregamento em 2 dimensões [5], sendo que a cada solução uma camada de caixas de mesma altura é gerada. Então, a alocação de camadas aos paletes pode novamente ser reduzida a um BPP [18].

Uma revisão dos métodos e resultados para o PLP é encontrado em [32]. Uma abordagem para resolução do DPLP, semelhante à utilizada nesse trabalho, consta em [2]. Primeiramente, são montadas camadas completas de um mesmo produto para montar os paletes, considerando que cada produto tem uma configuração de camada pré estabelecida. Formadas as camadas, paletes com todas as camadas de um mesmo produto são montados, e em seguida, paletes com camadas de mais de um produto. As caixas que ficam de fora dessas camadas completas são montados em paletes de sobra. Restrições de orientação (camadas devem ser paralelas ao palete), suporte das camadas (camadas superiores devem ser iguais ou não se sobreporem sobre as camadas inferiores) e empilhamento das caixas (limite de peso máximo permitido em cima de um produto) são utilizadas por [2]. Em [6], um DPLP também é resolvido com a montagem de camadas para comporem os paletes, porém são consideradas restrições de altura máxima do palete e empilhabilidade por altura, na qual camadas superiores não devem ser mais altas que inferiores. Outro exemplo da separação do DPLP em subproblemas encontra-se em [6], em que também, primeiramente são formadas camadas e depois os paletes são montados. Em [34], o DPLP é abordado resolvendo-o com uma decomposição em diversos problemas 2D.

Além da possibilidade separação do PLP em um problema de carregamento 2D e BPP, [20] aborda um DPLP como um 3D-BPP, utilizando um algoritmo genético. No trabalho de [16], DPLP também é resolvido como um 3D-BPP, utilizando uma abordagem de geração de colunas baseada em camadas, documentada no trabalho de [10]. O algoritmo GRASP é utilizado por [26] para o carregamento de paletes, em seu trabalho de carregamento de paletes e carregamento de caminhões. Em [1], o algoritmo GRASP é usado na resolução de carregamento de contêineres, muito semelhante ao PLP.

2.2. Bin packing

Como citado na Seção 2.1, existe uma redução do MPLP para o BPP, quando o primeiro é composto de camadas de caixas com arranjos pré-definidos.

O BPP pode ser descrito da seguinte forma: seja m o número de itens a serem empacotados e n o número de recipientes (*bins*), o BPP envolve a atribuição dos itens, representados por i, aos *bins*, representados por j, de modo que a capacidade máxima C dos *bins* seja respeitada e a quantidade de *bins* utilizados seja minimizada.

O modelo matemático do BPP de [23] é apresentado de (1) a (6), considerando um número de diferente de itens e bins. No referido equacionamento, w_i representa o peso do item i, e y_j e x_{ij} são variáveis binárias de decisão.

$$\text{Min } z = \sum_{j=1}^{n} y_j \tag{1}$$

Sujeito a:

$$\sum_{i=1}^{m} x_{ij} w_i \leq C y_j \qquad j \in \mathbb{N} = \{1, 2, ..., n\} \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad j \in \mathbb{N} \tag{3}$$

$$y_j, x_{ij} \in \{0, 1\} \qquad j \in \mathbb{N},\ i \in \mathbb{M} \tag{4}$$

Onde:

$$y_j = \begin{Bmatrix} 1 & \text{se o bin } j \text{ é utilizado} \\ 0 & \text{caso contrário} \end{Bmatrix} \tag{5}$$

$$x_{ij} = \begin{Bmatrix} 1 & \text{o item } i \text{ é atribuído ao bin } j \\ 0 & \text{caso contrário} \end{Bmatrix} \tag{6}$$

A função objetivo (1) minimiza o número total de *bins* (paletes) necessários para empacotar todos os itens. A restrição (2) garante que as capacidades dos paletes sejam respeitadas, ao mesmo tempo em que força a variável y_j a ser igual a 1 quando pelo menos um item (camada) for atribuída ao palete j. A restrição (3) impõem que cada camada seja atribuída a exatamente um palete.

Embora a resolução do BPP por métodos exatos seja uma opção viável para algumas instancias, problemas de grandes dimensões ainda demandam métodos heurísticos para que uma resposta seja encontrada em tempo computacional aceitável [13].

As contribuições de [8] podem ser destacadas, onde são apresentadas heurísticas para problemas de empacotamento, como o "Melhor Encaixe Decrescente" *("Best Fit Decreasing - BFD")*. Na parte das meta-heurísticas [19] aborda o BPP com o "Recozimento Simulado" *("Simulated Annealing")*, [31] utiliza o algoritmo exato "BISON" em conjunto com a "Busca Tabu" *("Tabu Search")*, [9] utiliza a meta-heurística LNS (*Large Neighborhood Search*) e [11] se aprofunda no estudo de algoritmos genéticos para o BPP.

Uma comparação entre diversas metaheuristicas estado da arte para o BPP é tratada por [27], na medida em que [7] faz uma comparação entre modelos matemáticos e métodos exatos de resolução para o problema.

No que diz respeito ao uso da meta-heurística VNS, as referências [12] e [17] são citadas para *bins* de tamanhos diferentes, enquanto [30] é destacado como a principal referência para este trabalho. Este último utiliza o VNS para resolver o "Problema do Empacotamento com Categorias Compatíveis" *(Bin Packing Problem with Compatible Categories - BPCC)*, uma variação do BPP.

Mesmo com a redução do DPLP para o BPP, pode existir um excesso de caixas que não formaram camadas completas. No presente trabalho, trazemos uma nova tratativa para essas caixas alocando-as acima dos paletes formados com camadas completas, por meio de algoritmos de carregamento em duas dimensões.

2.3. 2D Bin packing

Problemas de carregamento em 2 dimensões foram introduzidos na literatura a partir de demandas complexas para um eficiente empacotamento de componentes retangulares em grandes folhas de material, arranjo de mercadorias em prateleiras e a organização de artigos e anúncios em páginas [21].

O trabalho de [15] foi o primeiro a propor uma modelagem para o problema, propondo uma abordagem de geração de colunas baseada na enumeração de todos os subconjuntos de itens (padrões) que podem ser empacotados em um único recipiente.

Em [4], um procedimento de busca em árvore para problemas de corte bidimensional não guilhotinado foi desenvolvido, com o objetivo de maximizar o lucro ao empacotar um subconjunto de itens em um único recipiente, abordando questões práticas de otimização em ambientes industriais. Os trabalhos de[22] e [21] trazem revisões sobre estudos envolvendo o "Bin Packing Bidimensional" (2DBPP).

Já o trabalho de [34] apresenta uma abordagem heurística para decompor problemas de empacotamento bidimensional, aproveitando a estrutura de camadas e pilhas. Sua técnica divide o problema em subproblemas mais simples, primeiro montado pilhas, que formam camadas, que depois são usadas em paletes.

Acerca do uso de espaços como abordagens para a resolução de problemas de carregamento 2D, [28] e [1] exploraram meta heurísticas de algoritmos genéticos e GRASP para resolver problemas de carregamento de vários recipientes em ambientes práticos. Suas contribuições oferecem soluções eficazes para problemas de empacotamento com restrições complexas.

Para o presente trabalho, as contribuições da separação de espaços do trabalho de [14] foram usadas como inspiração para a tratativa de itens soltos na construção de camadas 2D irregulares.

Observa-se que muitos trabalhos abordam o PLP de maneira segmentada, focando em subproblemas específicos como o BPP utilizando camadas e carregamento bidimensional. Considerando essas abordagens, uma das lacunas encontradas no PLP é a tratativa de itens soltos (não incluídos nas camadas completas), que são deixados de fora da solução ou jogados em paletes de sobras. Este problema é abordado no nosso trabalho através de uma metodologia que integra os itens soltos nos paletes já montados com camadas completas, por um algoritmo de carregamento 2D. Além da tratativa, introduzimos também o conceito de priorização de *picking*, onde prioriza-se a montagem dos paletes de forma que o *picking* seja o mais dinâmico possível.

3. Desenvolvimento

O presente trabalho foi separado em 3 fases de desenvolvimento: a transformação dos dados de *input* iniciais, o VNS para o BPP, e uma heurística gulosa para o tratamento das caixas soltas. As etapas são ilustradas na Figura 1.

Na primeira fase de desenvolvimento ("processamento dos dados", na Figura 1), tem-se como entrada a demanda dos produtos, e como saída uma lista de camadas completas para o BPP

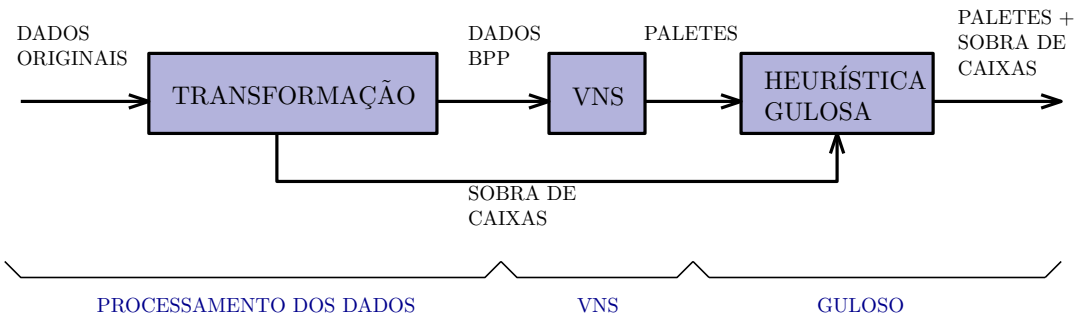

Figura 1: Fases de desenvolvimento do método

junto a uma lista de caixas que não formaram camadas completas que servirão de entrada para a heurística gulosa. É importante destacar que o padrão de arranjo de cada caixa no palete já é conhecido, de modo que a quantidade necessária de caixas de cada tipo para montar uma camada do palete já está determinada.

Ainda nessa fase, são calculados dois *lower bounds*. O primeiro *lower bound* verifica a factibilidade da relação carga-veículo, considerando a quantidade mínima de paletes necessários para atender à demanda. Se essa quantidade mínima exceder a capacidade de paletes do caminhão, a relação carga-veículo é considerada inviável, e será necessário escolher um caminhão de maior capacidade para transportar a demanda. O segundo *lower bound*, que também servirá como entrada para o BPP, representa a quantidade mínima de paletes necessários para acomodar as camadas completas geradas durante o processamento dos dados.

Para a segunda fase, com a lista de camadas completas geradas pela transformação de dados da primeira fase, o BPP é resolvido com o uso do VNS para montagem dos paletes (VNS na Figura 1). Os paletes montados e a lista de itens soltos que não foram acomodados nas camadas completas são os *inputs* da última etapa ("guloso" na Figura 1). Como as caixas de sobra possuem tamanhos diferentes, a heurística gulosa é responsável por montar camadas no espaço de duas dimensões (camadas 2D), considerando uma altura máxima da camada. Essas camadas podem então ser alocadas acima dos paletes já montados (somente uma camada por palete, já que as alturas não são uniformes).

Essa fase ("guloso") é dividida em duas etapas cíclicas: a escolha de um palete para a montagem de uma camada com os itens soltos, e a heurística 2D em si. Esse ciclo se repete até que todos os itens soltos tenham sido alocados, ou até o momento em que não hajam mais paletes disponíveis para montagem de uma camada 2D.

3.1. Transformação de dados

Os *inputs* iniciais que são passados para transformação de dados vem de uma lista de produtos a serem carregados, onde temos:

- I: conjunto de itens a serem carregados.
- i: item $i \in I$;
- s: SKU[1] (produto);
- q_s: quantidade de caixas a serem carregadas associadas ao produto (SKU) s;
- c_s: quantidade de caixas por camada de palete associadas ao SKU s;
- l_s: comprimento da caixa do SKU s;
- w_s: largura da caixa do SKU s;
- h_s: altura da caixa do SKU s;
- t_s: tipo de produto do SKU s;
- C: capacidade/altura máxima de um palete;

[1] Stock Keeping Unit

- L: comprimento de um palete;
- W: largura de um palete;

Em que:

$$t_s = \begin{cases} 1, & \text{se } s \text{ é resfriado} \\ 0, & \text{se } s \text{ é congelado.} \end{cases}$$

Com essa lista, é feito uma primeira verificação de factibilidade carga-veículo, e em seguida os *inputs* são pré-processados para o BPP.

Para a verificação de factibilidade, realiza-se o cálculo da quantidade miníma de paletes necessários para atender a demanda. Esse *lower bound*, definido como LB_1 e apresentado na equação 7, é feito pela divisão do volume total de todas as caixas da demanda pelo volume máximo de carga que um palete pode comportar, onde l_i, w_i e h_i são as dimensões das caixas de cada item e L, W e C são as dimensões de um palete.

$$LB_1 = \left\lceil \frac{\sum_i^n l_i w_i h_i}{LWC} \right\rceil \tag{7}$$

Caso o valor de LB_1 seja superior à capacidade máxima de paletes do caminhão designado para transportar a demanda, a relação carga-veículo é considerada infactível, e o programa se encerra devido à certeza de que a carga excede a capacidade do caminhão.

Feito a checagem, inicia-se a tratativa dos dados para o BPP. Inicialmente, os itens são ordenados para um *picking* dinâmico. A partir da lista de produtos ordenados, tenta-se montar o máximo de camadas completas possíveis. Existem duas possibilidades de camadas:

(i) camadas completas de um mesmo SKU;
(ii) camadas completas com mais de um SKU, que possuam as mesmas dimensões de caixa;

Primeiro prioriza-se a montagem de camadas homogêneas de um mesmo SKU. Passando por toda a lista de produtos, camadas j de um SKU são formadas com a divisão inteira de q_s por c_s. Cada camada terá uma altura $h_j = h_s$ do item que a formou.

Com os restos das divisões por inteiro da formação de camadas de SKUs únicos, são montadas as camadas com mais de um SKU. Para tal, o tipo de produto (t_s) deve ser o mesmo, e as dimensões l, w e h dos itens devem ser iguais, de forma que a padronização da camada seja a mesma. Assim como as camadas homogêneas, as camadas mistas terão uma altura $h_j = h_i$.

Formado as camadas, o *lower bound* LB_2 do BPP é calculado. Apresentado em 8, LB_2 estima a quantidade mínima de paletes necessários para acomodar as camadas completas geradas durante o processamento dos dados, através da divisão da soma das alturas das camadas pela altura máxima de um palete.

$$LB_2 = \left\lceil \frac{\sum_j^n h_j}{C} \right\rceil \tag{8}$$

No BPP, LB_2 será usado como critério de parada para o VNS. Ao atingir esse limite, a otimização não encontrá soluções factíveis que utilizem menos paletes do que o estimado pelo *lower bound*. Assim, a utilização desse critério de parada proporciona eficiência ao algoritmo, evitando iterações desnecessárias.

Ao final da fase de transformação dos dados, obtém-se uma lista de camadas completas para a formação dos paletes com BPP, um *lower bound* como critério de parada para o VNS, e uma lista de itens soltos, que serão usados na heurística. Após o processamento de dados, os produtos resfriados e congelados seguirão separadamente as fases apresentadas no desenvolvimento.

Entradas geradas para o BPP:

- J: conjunto de camadas a serem alocadas;
- c_j: camada $j \in J$;
- h_j: altura da camada j;
- LB_2: *lower bound* do BPP;

Entradas geradas para Heurística Gulosa:

- Q: conjunto de itens soltos a serem carregados.
- q: item $q \in Q$;
- l_q: comprimento da caixa do item $q \in Q$;
- w_q: largura da caixa do item $q \in Q$;
- h_q: altura da caixa do item $q \in Q$;

3.2. VNS para o BPP

Os métodos de busca local utilizados na otimização combinatória seguem uma abordagem em que uma solução inicial é aprimorada por meio de mudanças locais. A cada nova iteração, uma solução aprimorada, denotada por x, é obtida a partir da solução corrente, considerando as vizinhanças das buscas locais *LocalSearch1* e *LocalSearch2*. Esse processo é repetido até que não seja mais possível encontrar melhorias adicionais.

O VNS é uma técnica de otimização baseada na exploração sistemática de diferentes vizinhanças de uma solução com o objetivo de evitar uma estagnação em ótimos locais de baixa qualidade. Em contraste com muitos outros métodos de busca local, o VNS não segue uma trajetória fixa, mas, em vez disso, explora diferentes vizinhanças da solução corrente, pulando para novas soluções quando uma melhoria é identificada. Dessa forma, muitas características vantajosas da solução atual, como por exemplo, a otimização da maioria das variáveis, são mantidas e utilizadas para gerar soluções promissoras [24]. Ao final do VNS, os operadores de perturbação $N(x)$ são responsáveis por realizar alterações na solução atual, de maneira a adicionar um componente de exploração para o algoritmo. Com esses operadores a solução é jogada para outra região do *fitness landscape*, de forma a tentar evitar ótimos locais.

O pseudocódigo do VNS desenvolvido neste estudo é mostrado em Algoritmo 1, onde x_{best} é a melhor solução encontrada, x é a solução corrente, n é o número de paletes de x_{best}, K e α são os limites de iterações e iterações sem melhoria de x_{best}, e N_{iter} e N_α são seus respectivos contadores.

O procedimento é encerrado quando um dos três critérios de parada for atendido:

1. Após o número de iterações sem melhoria N_α superar (α)
2. Após N_{iter} superar K
3. Se o número de paletes n se igualar ao *lower bound* LB_2

Como solução inicial (linha 1), é utilizado o algoritmo do "Melhor Encaixe" *("Best Fit - BF")*, uma abordagem clássica para resolução do BPP. O BF consiste em tentar alocar cada item no *bin* disponível que tenha espaço restante mais próximo ao tamanho do item, desde que o item caiba no *bin*. Se o item não couber em nenhum *bin*, um novo é aberto.

Outra heurística clássica é o "Primeiro Encaixe" *("First Fit - FF")*, onde o item é alocado no primeiro *bin* com espaço disponível. Variantes mais eficientes incluem o "Primeiro Encaixe Decrescente" *("First Fit Decreasing - FFD")* e o "Melhor Encaixe Decrescente" *("Best Fit Decreasing - BFD")*, nas quais antes da alocação, os itens são ordenados de forma decrescente em relação ao tamanho, priorizando alocar primeiro os itens maiores, que tendem a ser mais desafiadores de serem acomodados posteriormente.

As heurísticas FF e BFD são usadas como componentes de reconstrução nos operadores de *shaking* do VNS, que serão detalhadas mais adiante.

Nas próximas subseções as principais funções do algoritmo proposto serão definidas.

Algoritmo 1: VNS para o BPP

Entrada: Conjunto de camadas J, alturas h_j, capacidade do palete C, *lower bound* LB_2, limite de iterações K e limite de iterações sem melhoria α

Saída: Paletes montados x_{best}

1 $x_{best} \leftarrow BF(J, h_j, C)$;
2 $x \leftarrow x_{best}$;
3 $N_{iter} \leftarrow 0$;
4 $N_\alpha \leftarrow 0$;
5 **repita**
6 $x \leftarrow LocalSearch1(x)$;
7 $x \leftarrow LocalSearch2(x)$;
8 $x \leftarrow LocalSearch1(x)$;
9 **se** $g(x) > g(x_{best})$ **então**
10 $x_{best} \leftarrow x$;
11 $n \leftarrow NumPaletes(x_{best})$;
12 $N_\alpha \leftarrow 0$;
13 **senão**
14 $N_\alpha \leftarrow N_\alpha + 1$;
15 $h \leftarrow SortShaking$;
16 $x \leftarrow Shaking(x, h)$;
17 $N_{iter} \leftarrow N_{iter} + 1$;
18 **até** $N_{iter} > K$ ***ou*** $N_\alpha > \alpha$ ***ou*** $n = LB_2$;

3.2.1. Função de aptidão

Um fator crucial para qualquer procedimento de busca de vizinhança é a escolha da função objetivo. Utilizar a função objetivo direta do problema, ou seja, minimizar o número de *bins* 1, pode ser redundante, visto que camadas podem ser organizadas de maneiras diferentes, porém ainda correspondendo ao mesmo número de *bins* [12]. Dado que as melhores soluções frequentemente envolvem *bins* mais cheios [29], foi adotada uma função de ocupação $g(x)$, denominada função de aptidão, com o propósito de avaliar e atribuir uma pontuação a uma dada solução x [30].

Na função de aptidão proposta (9), N_j representa o conjunto de itens i pertencentes ao *bin* j (ou seja, aqueles para os quais $x_{ij} = 1$), e n_{bins} é o número de *bins* usadas em uma solução específica x.

$$g(x) = \sum_{j}^{n_{bins}} \left(\sum_{i \epsilon N_j} x_{ij} w_i \right)^2 \tag{9}$$

A função de aptidão funciona de tal forma que, dado duas soluções de uma mesma instância, x e x', ambas com n_{bins}, a solução que tiver o maior desvio padrão em relação à capacidade usada dos *bins* terá uma nota maior na função. Ou seja, soluções que apresentarem *bins* quase cheios e outros quase vazios, terão pontuação maior do que soluções com *bins* igualmente preenchidos. Comparando a pontuação de duas soluções através de $g(x)$ (linha 9), garantimos a escolha da solução mais apta.

3.2.2. Busca Local

Em cada iteração, dois procedimentos de busca local diferentes são realizados (linhas 6, 7 e 8). O primeiro procedimento, chamado de L_1 (*LocalSearch1*) e apresentado no Algoritmo 2, e tem

por objetivo mover uma camada de um *bin* para outro, de forma a melhorar a função de aptidão.

Algoritmo 2: *LocalSearch1*

Entrada: Conjunto de camadas J, alturas h_j, capacidade do palete C, Paletes montados x
Saída: Paletes montados x

```
1  P ← CojuntoPaletes(x);
2  para cada palete a ∈ P faça
3      para cada palete b ∈ P | a ≠ b faça
4          para cada camada j ∈ a faça
5              se h_j ⩽ (C − Σ_{i∈a} h_i) então
6                  x_r ← Realoca(a, b, j, x);
7                  se g(x_r) > g(x) então
8                      x ← x_r;
```

Sejam a e b os índices dos dois *bins* (paletes) considerados para a operação, onde a é o *bin* que terá uma camada removida e b o *bin* que receberá essa camada. Para cada camada pertencente a *bin* a, verifica-se se esta pode ser realocada em b. A realocação ocorre apenas se as restrições de capacidade do compartimento de destino não forem extrapoladas (linha 5) e se houver melhoria na função de aptidão 9 (linha 7). O procedimento continua até que todos os *bins* tenham sido considerados.

A segunda busca local L_2 (*LocalSearch2*), apresentada em Algoritmo 3, é baseada na troca de camadas entre *bins* diferentes, a partir da solução corrente fornecida por L_1.

Algoritmo 3: *LocalSearch2*

Entrada: Conjunto de camadas J, alturas h_j, capacidade do palete C, Paletes montados x
Saída: Paletes montados x

```
1  P ← CojuntoPaletes(x);
2  para cada palete a ∈ P | a ∉ BinsUsados faça
3      para cada palete b ∈ P | b ∉ BinsUsados | a ≠ b faça
4          para cada camada j_a ∈ a faça
5              para cada camada j_b ∈ b faça
6                  se C ⩽ (Σ_{i∈a} h_i − h_{j_a} + h_{j_b}) e C ⩽ (Σ_{i∈b} h_i − h_{j_b} + h_{j_a}) então
7                      x_r ← TrocaCamada(a, b, j_a, j_b, x);
8                      se g(x_r) > g(x) então
9                          x ← x_r;
10                         BinsUsados ← a;
11                         BinsUsados ← b;
```

Sejam a e b os índices para referenciar os *bins* avaliados para a troca de camada. Para cada camada em a, verifica-se a possibilidade de troca com todas as camadas de b. Caso a troca de camadas não extrapole a capacidade dos *bins* (linha 6) e haja melhoria da função de aptidão (9) (linha 8), a troca é realizada. Os *bins* que tiverem suas camadas trocadas, não são considerados nas iterações seguintes (linhas 10 e 11), e L_2 se encerra quando todos os pares de a e b possíveis tenham sido avaliados.

Tanto para L_1 e L_2, a ordem em que os índices a e b são selecionados é pela própria

formação dos *bins* no BF. Note que a redução do número de *bins* só é possível com L_1, portanto, as buscas locais são feitas em sequência, começando por L_1, seguindo para L_2, e finalizando novamente com L_1.

3.2.3. Operadores de perturbação (*shaking*)

No intuito de obter novas soluções para serem exploradas pelas buscas locais, são utilizados operadores de perturbação *("Shaking Operators")*, que alteram a solução atual, destruindo e reconstruindo uma parte da mesma. A função *Shaking (x,h)* seleciona um dos operadores de perturbação N_h e o aplica à solução atual x (linha 16). Tendo como base [30], foram implementados 4 operadores de *shaking*.

(i) N_1: aleatoriamente quebram-se $x\%$ dos *bins*, e a heurística BFD é utilizada para construção de novos *bins* com as camadas soltas dos *bins* quebrados;
(ii) N_2: aleatoriamente quebram-se $y\%$ ($y > x$) dos *bins*, e a heurística BFD é utilizada para construção de novos *bins* com as camadas soltas dos *bins* quebrados;
(iii) N_3: aleatoriamente quebram-se $x\%$ dos *bins*, as camadas soltas são atribuídas de forma aleatória, e novos *bins* são construídos com a heurística FF;
(iv) N_4: dois novos *bins* são abertos, e aleatoriamente escolhem-se camadas de outros *bins* para preenche-los, até que uma camada não caiba em um dos dos novos *bins*;

Ao fim de cada iteração, um dos 4 operadores é escolhido aleatoriamente (linha 15) para realizar a perturbação em x.

3.3. Heurística Gulosa

Tendo os paletes montados através do BPP (x), a capacidade dos *bins* (C), a lista de itens soltos que não foram acomodados nas camadas completas (Q), e as dimensões desses itens (l_q, w_q, h_q), temos os *inputs* necessários para heurística gulosa. O Algoritmo 4 abaixo mostra como os itens soltos são alocados em cima dos paletes.

Algoritmo 4: Heurística Gulosa

Entrada: x, C, Q, l_q, w_q, h_q
Saída: Paletes modificados x_c, padrões x_{2D}

```
1  x_c ← x;
2  x_2D ← ∅;
3  repita
4      (x_c, bin, q_0) ← EscolheBin(x_c, C, Q, h_q);
5      se q_0 ≠ 0 então
6          (Q, x_2D) ← Heuristica2D(bin, q_0, Q, L, W, l_q, w_q);
7          camada2D ← Verdadeiro;
8      senão
9          camada2D ← Falso;
10 até Q = ∅ ou camada2D = Falso;
```

Como já citado acima, essa fase é divida em duas etapas que rodam de forma cíclica. Primeiro, ocorre a seleção de um *bin* e um item inicial q_0 para a montagem de uma camada 2D em cima do palete com dimensões L e W (linha 4). Em seguida ocorre o processo de montagem da camada 2D (linha 6). Ambas as etapas são descritas abaixo, sendo que o ciclo se repete até que não existam mais itens soltos ou paletes disponíveis para a montagem de mais camadas 2D.

3.3.1. Escolhe Bin

Para que uma camada 2D seja montada em cima de um palete, é necessário que q_0, o maior item da camada, seja menor ou igual à altura de sobra do palete j, conforme a condição 10.

$$\mathrm{C} - \sum_{i=1}^{n} x_{ij} h_i \geq h_{q_0} \qquad j \,\epsilon\, \mathbb{N} = \{1, 2, ..., n\} \tag{10}$$

Dentre todos os itens ainda não alocados de Q, q_0 deve ser o item cuja altura mais se aproxima da altura de sobra do palete, e os demais itens que irão compor a camada 2D devem ser necessariamente menores ou iguais à h_{q_0}. Para facilitar a alocação desses itens, o conjunto de itens não alocados Q é ordenado pelas alturas h_q de forma decrescente, priorizando a alocação de itens mais altos primeiro. Tendo escolhido um palete e q_0, a camada 2D é montada com os itens subsequentes do conjunto ordenado Q.

Existem duas operações que realizam a escolha do par item-palete, apresentadas no Algoritmo 5: *EscolheBinExistente* (linha 2) e *MovimentaCamada* (linha 6), sendo que *MovimentaCamada* será executado apenas se *EscolheBinExistente* não conseguir definir q_0 (linha 3).

Algoritmo 5: Escolhe Bin

Entrada: x_c, C, Q, h_q
Saída: Nova configuração x_c, bin, q_0

```
1 Q ← OrdenaItems(Q);
2 (q_0, bin, x_c) ← EscolheBinExistente(x_c, C, Q, h_q);
3 se q_0 = 0 então
4     (q_0, bin, x_c) ← MovimentaCamada(x_c, C, Q, h_q);
```

Para entender o funcionamento dessas duas operações, a seguinte notação e classificação dos *bins* foi criada:

- *Bin Disponível* (D): são *bins* que camadas 2D podem ser alocadas em cima;
- *Bin Usado* (U): são *bins* que já foram alocadas camadas 2D e não podem ser mais utilizados;
- *Bin Prematuro* (P): são *bins* criados pelo procedimento *MovimentaCamada*, que ainda não estão aptos a receberem uma camada 2D;

EscolheBinExistente A primeira opção de combinação de q_0 e palete é obtida por *EscolheBinExistente* (linha 2). Caso haja mais de uma combinação possível, é escolhido a combinação de *bin* e item que minimize a sobra de espaço.

Existem 3 restrições que devem ser respeitadas para que a combinação seja válida:

(i) O *bin* não pode ser um *Bin U*;
(ii) O *bin* não pode ser um *Bin P*;
(iii) O item escolhido q_0 só poderá ser o menor item do conjunto Q se ele for o último item a ser alocado;

Tendo em vista que é permitido apenas uma camada 2D por *Bin D*, a restrição (iii) serve para evitar que sejam montadas camadas 2D com q_0 sendo o menor item de Q, visto que essa camada poderia não ter itens suficientes para monta-la de forma eficiente. Caso nenhum item seja escolhido pelo *EscolheBinExistente*, a combinação é escolhida através do *MovimentaCamada*.

Algoritmo 6: *MovimentaCamada*

Entrada: x_c, C, Q, h_q
Saída: x_c, bin, q_0

```
1  q_0 ← MaiorItem(Q);
2  (BinD, C_r) ← EscolheCamadaRemovida(x_c, h_{q_0});
3  se BinP = ∅ então
4      (x_c, BinP) ← AbreBinP(x_c, C_r);
5      (bin, x_c) ← Bin2D(x_c, BinD);
6  senão se h_{C_r} > h_{BinP} então
7      se h_{q_0} ⩽ h_{BinP} então
8          (bin, x_c) ← Bin2D(x_c, BinP);
9      senão
10         x_c ← NovoBinD(x_c, BinP);
11         BinP = ∅;
12         (x_c, BinP) ← AbreBinP(x_c, C_r);
13         (bin, x_c) ← Bin2D(x_c, BinD);
14 senão
15     BinP ← AdicionaC_r(C_r, BinP);
16     (bin, x_c) ← Bin2D(x_c, BinD);
```

MovimentaCamada A operação é apresentada pelo Algoritmo 6. Nessa operação q_0 obrigatoriamente será o item de maior altura em Q (linha 1).

Primeiramente, procura-se uma camada de um *Bin D* que, se retirada, permite que q_0 seja alocado (linha 2). Para a definição de qual *Bin D* terá a camada removida (C_r), escolhe-se o *bin* cuja camada deslocada gere a menor diferença entre a altura h_{q_0} e a altura de sobra do *bin*.

Selecionado C_r, verifica-se a existência de um *Bin P* (linha 3). Um *Bin P* é um *bin* que foi criado a partir de uma camada C_r e que ainda não possui uma altura suficiente para ser utilizado para alocação de uma camada 2D. Caso *Bin P* não exista, um *Bin P* é criado a partir de C_r e o *Bin D* que teve C_r removido é escolhido para alocação da camada 2D (linhas 4 e 5). Com a montagem da camada 2D, *Bin D* passa a ser um *Bin U* como representado na Figura 2.

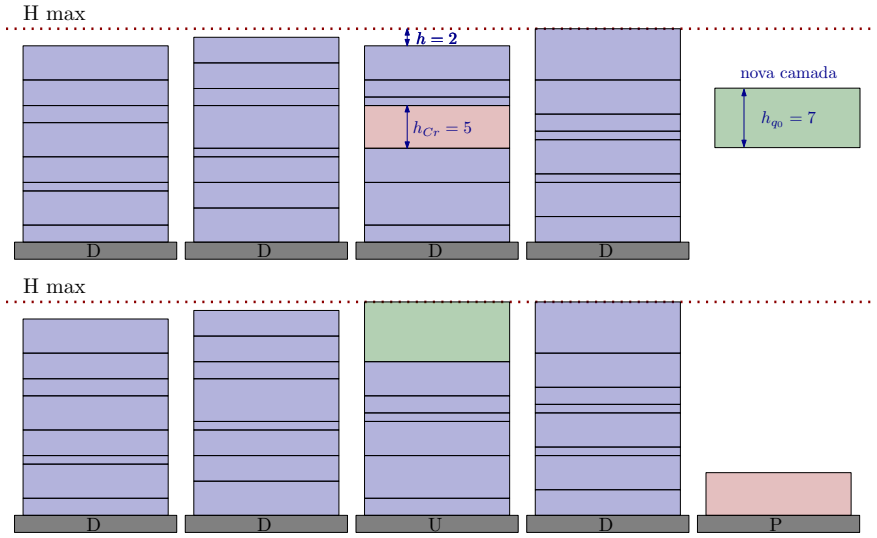

Figura 2: Cria Bin Prematuro

Caso *Bin P* já exista e a camada C_r não couber no mesmo (linha 6), significa que esse *Bin P* já atingiu a maturação e possui uma altura compatível a de um *Bin D*. Nesse caso, se a altura h_{q_0} for menor ou igual a altura de sobra de *Bin P*, a camada 2D é montada em *Bin P* (linha 8 e 9), que passa a ser considerado como um *Bin U*.

No exemplo da Figura 3, a camada 2D tem $h_{q_0} = 7$, o *Bin D* tem espaço de sobra 4, e a camada C_r tem altura $h = 4$, totalizando uma altura disponível 8, enquanto o *Bin P* tem altura disponível 7. Alocar a camada 2D no diretamente no *Bin P*, nesse cenário, é mais vantajoso, visto que altura residual seria 0 $(7 - 7)$, em comparação à altura residual no *Bin D*, que seria 1 $(8 - 7)$.

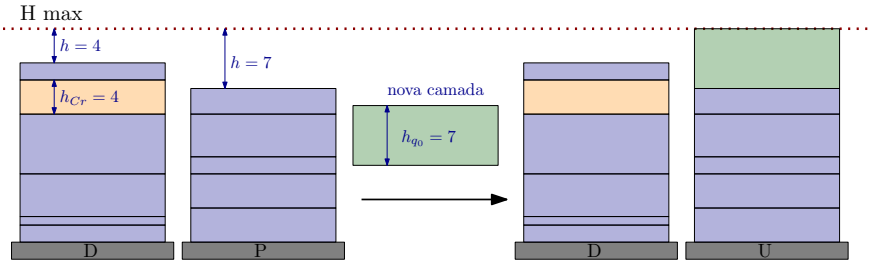

Figura 3: Uso direto do Bin Prematuro

Caso a camada C_r não caiba no *Bin P* e a altura h_{q_0} for maior que a altura de sobra de *Bin P*, esse *Bin P*, que já está maturado, passa a ser um *Bin D* (linhas 10 e 11), um novo *Bin P* é aberto com C_r (linha 12), e a camada 2D é montada em *Bin D* que teve a camada removida (Figura 4).

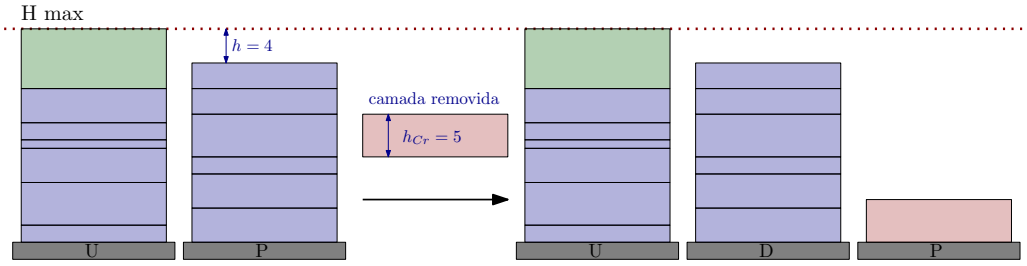

Figura 4: Maturação Bin Prematuro

Caso nenhuma das situações apresentadas aconteça, C_r é deslocada para *Bin P* (linha 15), e *Bin D* é usado para a montagem 2D (linha 16) (Figura 5).

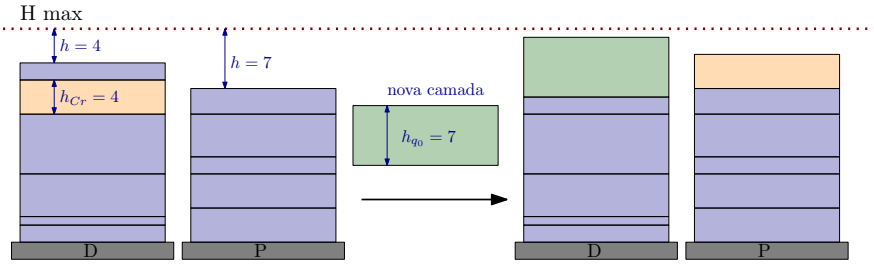

Figura 5: Camada deslocada para Bin Prematuro

3.3.2. Heurística 2D

O procedimento guloso para heurística 2D é apresentado em Algoritmo 7. Após a execução de *EscolheBin2D* (Algoritmo 5), tem-se o *bin* no qual é montada a camada 2D junto ao item inicial q_0 do conjunto Q. A partir desse item, os próximos itens de Q serão alocados na camada até o

critério de parada. Nesse procedimento, existem dois pontos chaves: o conceito de espaços e a acoplagem de itens iguais.

Algoritmo 7: Heurística2D

Entrada: $bin, q_0, Q, L, W, l_q, w_q$
Saída: Conjunto x_{2D}, Q

```
1  e ← (L, W);
2  E ← e;
3  i ← 0;
4  repita
5      E ← OrdenaEspacos(E);
6      para cada e ∈ E faça
7          se q_i cabe em e então
8              (coord_i, l_i, w_i, q_{i+1}, Q) ← Orienta&Acopla(q_i, Q, x_2D, l_q, w_q);
9              E ← NovosEspacos(coord, l_i, w_i);
10             Coube = True;
11             i ← i + 1;
12             saia do loop;
13         senão
14             Coube = False;
15 até Coube = False;
```

Um espaço ($e \in E$) é uma região disponível para acomodar um item. O espaço inicial na montagem da camada 2D tem as dimensões de comprimento L e largura W do palete (linha 1). O algoritmo se resume a alocar itens em espaços.

Sejam L_e e W_e as dimensões de e referentes ao comprimento horizontal (x) e comprimento vertical (y) de um eixo. Sempre que um item (ou conjunto de itens iguais) q_i é alocado em um espaço, esse espaço deixa de existir e dois novos são criados: um acima do item alocado, com dimensões (L_e, $W_e - w_i$), com origem em ($x_e, y_e + w_i$), e um espaço a direita do item alocado, com dimensão ($L_e - l_i, W_e$) e origem em ($x_e + l_i, y_e$). Essa operação é representada pela função *NovosEspaços* (linha 9). Se um dos novos espaços é vazio (uma de suas dimensões é zero) o espaço não é criado.

A cada iteração i, verifica-se se q_i cabe em e com ou sem rotação. Se q_i não couber em nenhum e, o algoritmo se encerra (linha 14). Caso caiba, tenta-se acoplar o máximo de itens iguais a q_i, formando um conjunto de itens com dimensões l_i e w_i. Por exemplo, supondo que q_i tenha 4 itens iguais, e e tenha espaço suficiente para acoplarmos apenas dois desses itens. A acoplação é feita, dois itens q_i são retirados de Q, e q_{i+1} recebe q_i para que na próxima iteração o restante dos itens sejam alocados no próximo conjunto E. Os acoplamentos são sempre feitos na horizontal à direita.

Além da opção de acoplamento, há também a alternativa de rotacionar os itens. Nessa opção, realiza-se uma simulação de acoplamentos e acomodações nos espaços para verificar se a rotação é vantajosa ou não. Para escolha de alocação dos itens com ou sem rotação, os critérios abaixo são considerados:

(i) Escolhe-se a combinação de rotação e acoplamento que gere um espaço vazio;
(ii) Escolhe-se a combinação de rotação e acoplamento que tenha mais itens acoplados;
(iii) Escolhe-se a combinação de rotação e acoplamento que gere o maior espaço;

Em caso de empate em todos os critérios, o conjunto sem rotação é escolhido. A função de orientação e acoplamento é representada por *Orienta&Acopla* (linha 8), e na Figura 6 são

apresentados três exemplos de criação de espaços, rotação e acoplamento que respeitam os critérios mencionados.

Quando é definido qual arranjo será utilizado, os novos espaços são criados, q_i é registrado, o espaço usado é removido, e os itens usados são removidos de Q. Com os novos espaços gerados, o código continua a partir do próximo item da lista, até que não seja possível alocar mais nenhum item.

O algoritmo se encerra quando q_i (rotacionado ou não) não couber em nenhum espaço (linha 14). Então, mais um loop da Heurística Gulosa (Algoritmo 4) se inicia: mais um *bin* é escolhido para alocar uma camada 2D e uma nova camada é montada, até que todos os itens sejam alocados ou não hajam mais *bins* para o *EscolheBin* (Algoritmo 5).

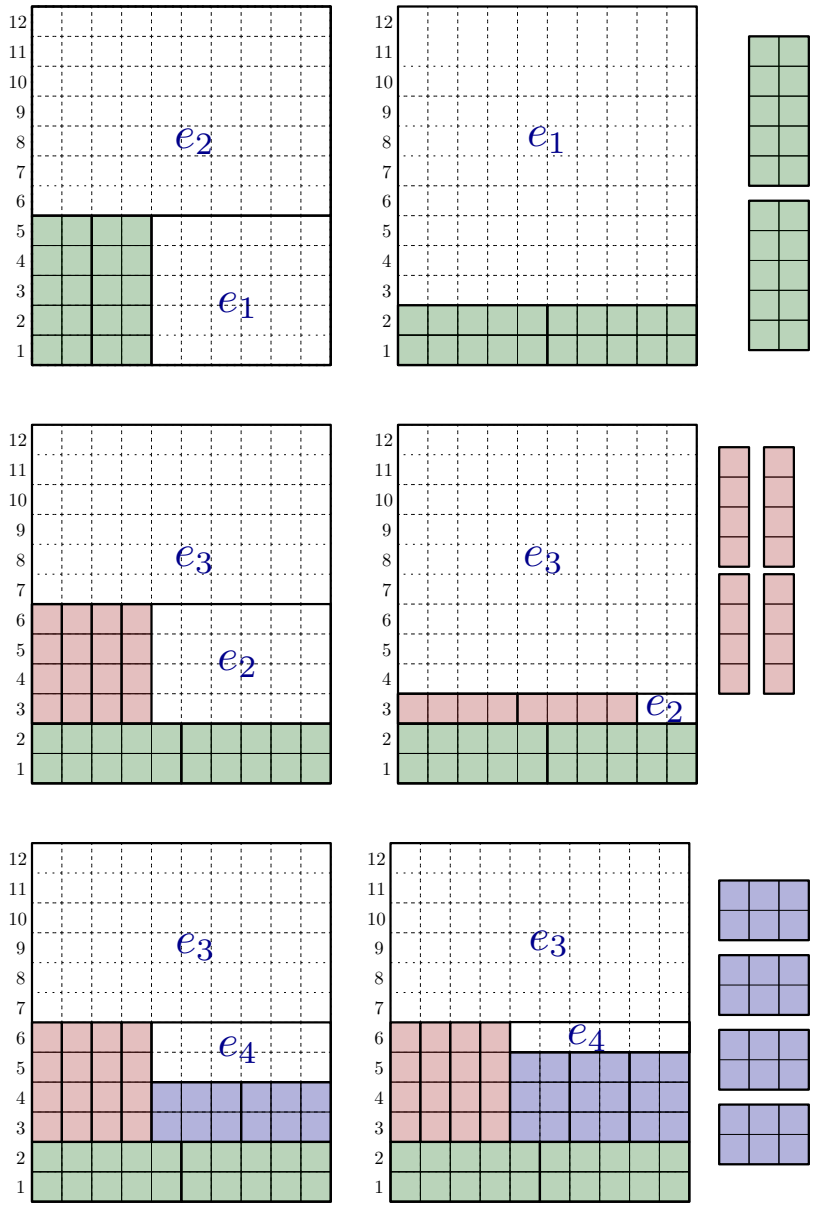

Figura 6: Exemplo Heurística 2D

4. Resultados

Nesta seção são apresentados os resultados computacionais para os algoritmos propostos. Na subseção 4.1 os resultados do algoritmo VNS aplicado a um conjunto de instâncias da literatura são apresentados. Já na subseção 4.2, uma instância teste criada simulando uma demanda da

empresa é utilizada para validar a solução completa, VNS e heurística gulosa 2D. Os algoritmos foram implementados utilizando VBA e os experimentos computacionais são realizados em um PC de 64 bits equipado com 8 GB de RAM e um processador Intel Core i7-3537U operando entre 2,0 e 2,6 GHz. Os parâmetros utilizados no VNS foram:

1. Iterações sem melhoria: 50
2. Porcentagem de quebra x: 0.2
3. Porcentagem de quebra y: 0.5

4.1. VNS para o BPP

A Tabela 1 mostra os resultados computacionais para 2 conjuntos de instâncias do BPP do repositório OR-Library. Os testes foram realizados em dois conjuntos de instâncias, **u120**_ e **u500**_, cada uma destas com uma variação de 20 problemas com 120 e 500 itens, respectivamente. Os resultados na tabela mostram o nome da instância (Instancia), o menor número de *bins* encontrados até o momento (BKS - *Best known solution*), o número encontrado pelo algoritmo (VNS), o número de iterações executadas (n_iter), o tempo em segundos (Tempo) e o *gap* percentual entre a melhor solução e a solução encontrada. O *gap* é dado pela fórmula 11 ([33]).

$$|f(s) - f(s^*)|/f(s^*) \tag{11}$$

Em que s é a solução encontrada pelo VNS e s^* a melhor solução encontrada até o momento (BKS).

Instancia	BKS	VNS	n_iter	Tempo	gap
u120_00	48	48	103	1.09	0.00*
u120_01	49	49	87	0.95	0.00*
u120_02	46	46	74	0.79	0.00*
u120_03	49	49	74	0.92	0.00*
u120_04	50	50	68	0.76	0.00*
u120_05	48	48	137	1.55	0.00*
u120_06	48	48	76	0.84	0.00*
u120_07	49	49	95	1.17	0.00*
u120_08	51	51	59	0.76	0.00*
u120_09	46	46	79	1.00	0.00*
u120_10	52	52	65	0.82	0.00*
u120_11	49	49	128	1.94	0.00*
u120_12	48	48	79	1.26	0.00*
u120_13	49	49	83	1.02	0.00*
u120_14	50	50	70	0.92	0.00*
u120_15	48	48	151	1.84	0.00*
u120_16	52	52	54	0.70	0.00*
u120_17	52	52	56	0.73	0.00*
u120_18	49	49	101	1.30	0.00*
u120_19	50	50	61	0.81	0.00*

Instancia	BKS	VNS	n_iter	Tempo	gap
u500_00	198	199	82	15.60	0.01
u500_01	201	202	92	17.40	0.00
u500_02	202	203	80	16.28	0.00
u500_03	204	205	61	13.94	0.00
u500_04	206	207	98	20.75	0.00
u500_05	206	206	67	14.22	0.00*
u500_06	207	208	74	16.15	0.00
u500_07	204	205	98	21.32	0.00
u500_08	196	197	73	15.35	0.01
u500_09	202	202	96	19.64	0.00*
u500_10	200	200	68	12.97	0.00*
u500_11	200	201	138	26.15	0.01
u500_12	199	201	106	20.67	0.01
u500_13	196	197	83	16.02	0.01
u500_14	204	205	71	13.77	0.00
u500_15	201	202	158	30.61	0.00
u500_16	202	203	84	16.22	0.00
u500_17	198	200	80	15.45	0.01
u500_18	202	202	86	17.90	0.00*
u500_19	196	197	56	12.41	0.01

Tabela 1: Resultados para as instâncias OR-Library Falkenauer

Os valores da coluna *gap* marcados com um asterisco indicam as instâncias em que o BKS foi encontrado. Analisando a Tabela, percebe-se que para o conjunto u120_ o algoritmo proposto encontrou o BKS em todas as instâncias, já para o conjunto u500_ foram encontrados em 4 das 20, sendo que naquelas em que o BKS não foi encontrado, o gap não passou de 1%.

4.2. Heurística 2D gulosa

A instância desenvolvida para simular o algoritmo inteiro apresentado nesse estudo foi projetada para simular uma demanda real, onde os dados fornecidos são apresentados na subseção 3.1.

A Figura 7 mostra os arranjos de camadas 2D obtidos ao final da fase “Guloso” (Figura 1), considerando paletes com dimensões 1000mm por 1200m. Além de não ser uma solução custosa computacionalmente, a heurística apresentou padrões com alto aproveitamento da área.

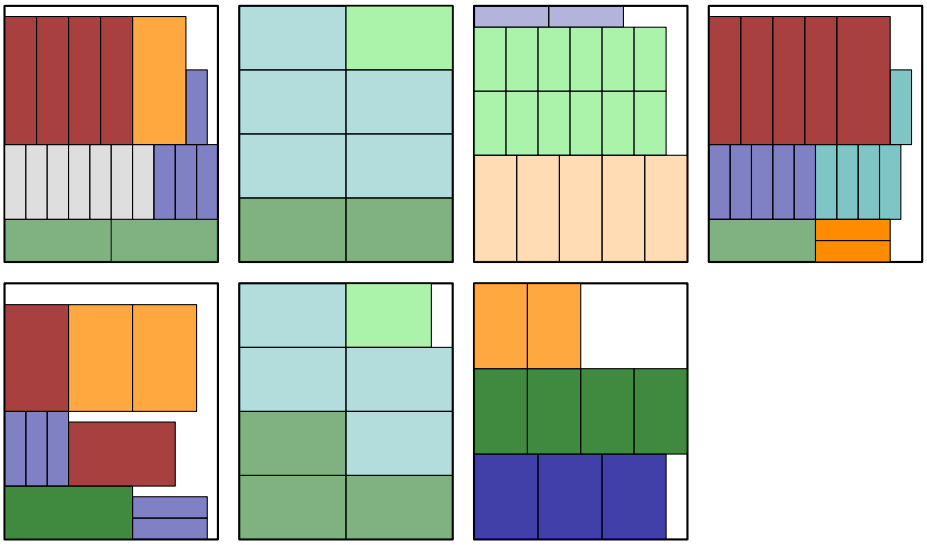

Figura 7: Resultados heurística 2D

Conclusões

Neste artigo, o “Problema de Empacotamento de Paletes do Distribuidor” (DPLP) de uma grande empresa do setor alimentício foi abordado. Ao transformar o DPLP em um “Problema de Empacotamento de Caixas” (BPP) e implementar o algoritmo de Busca em Vizinhança Variável (VNS), foram obtidos resultados rápidos e eficazes para as instâncias testadas.

A heurística gulosa para o Problema de Empacotamento Bidimensional (BPP 2D) se provou eficiente para o tratamento dos itens não incluídos na composição das camadas do BPP, entretanto, é importante ressaltar a limitação dessa combinação do uso do BPP com a heurística gulosa. Como já visto na subseção 3.3, um dos critérios de parada da heurística gulosa é não haverem mais paletes disponíveis para a alocação de uma camada 2D. Instâncias onde hajam muitos itens soltos e poucos paletes iniciais formados pelo BPP, podem acarretar no algoritmo se encerrar sem que todos os itens sejam alocados nos paletes. O tratamento desses itens nesse tipo de cenário é um ponto de melhoria para desenvolvimento e aplicações futuras.

Os resultados deste estudo não apenas oferecem uma solução prática para os desafios específicos enfrentados pela empresa de distribuição de alimentos, mas também contribuem para a compreensão mais ampla de metodologias eficientes de carregamento de paletes. A integração de VNS e estratégias heurísticas mostrou a adaptabilidade dessas técnicas na abordagem de problemas logísticos do mundo real. Para pesquisas futuras, o aprofundamento da introdução do *picking* varável é uma oportunidade a ser explorada.

Agradecimentos. Os autores agradecem o apoio da Sociedade Brasileira de Pesquisa Operacional (SOBRAPO).

Referências

[1] Maria Teresa Alonso, Ramón Alvarez-Valdés, and Francisco Parreño. A grasp algorithm for multi container loading problems with practical constraints. *4or*, 18:49–72, 2020.

[2] Maria Teresa Alonso, Ramon Alvarez-Valdes, Francisco Parreño, Jose Manuel Tamarit, et al.

Algorithms for pallet building and truck loading in an interdepot transportation problem. *Mathematical Problems in Engineering*, 2016, 2016.

[3] MT Alonso, Antonio Martinez-Sykora, R Alvarez-Valdes, and F Parreño. The pallet-loading vehicle routing problem with stability constraints. *European Journal of Operational Research*, 302(3):860–873, 2022.

[4] John E Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64, 1985.

[5] EE Bischoff and MSW Ratcliff. Loading multiple pallets. *Journal of the Operational Research Society*, 46:1322–1336, 1995.

[6] Mauro Dell'Amico and Matteo Magnani. Solving a real-life distributor's pallet loading problem. *Mathematical and Computational Applications*, 26(3):53, 2021.

[7] Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20, 2016.

[8] Samuel Eilon and Nicos Christofides. The loading problem. *Management Science*, 17(5):259–268, 1971.

[9] Ali Ekici. A large neighborhood search algorithm and lower bounds for the variable-sized bin packing problem with conflicts. *European Journal of Operational Research*, 2023.

[10] Samir Elhedhli, Fatma Gzara, and Burak Yildiz. Three-dimensional bin packing and mixed-case palletization. *INFORMS Journal on Optimization*, 1(4):323–352, 2019.

[11] Emanuel Falkenauer, Alain Delchambre, et al. A genetic algorithm for bin packing and line balancing. In *ICRA*, pages 1186–1192. Citeseer, 1992.

[12] Krzysztof Fleszar and Khalil S Hindi. New heuristics for one-dimensional bin-packing. *Computers & operations research*, 29(7):821–839, 2002.

[13] Michel Gendreau and Jean-Yves Potvin. Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140:189–213, 2005.

[14] John A George and David F Robinson. A heuristic for packing boxes into a container. *Computers & Operations Research*, 7(3):147–156, 1980.

[15] PC Gilmore and RE Gomory. Multistage cutting problem of two and more dimensions. *New York.: Thomas J. Watson Research Center, Yorktown Heigh*, 1964.

[16] Fatma Gzara, Samir Elhedhli, and Burak C Yildiz. The pallet loading problem: Three-dimensional bin packing with practical constraints. *European Journal of Operational Research*, 287(3):1062–1074, 2020.

[17] Vera Hemmelmayr, Verena Schmid, and Christian Blum. Variable neighbourhood search for the variable sized bin packing problem. *Computers & Operations Research*, 39(5):1097–1108, 2012.

[18] Thom J Hodgson. A combined approach to the pallet loading problem. *IIE Transactions*, 14(3):175–182, 1982.

[19] Thomas Kämpke. Simulated annealing: Use of a new tool in bin packing. *Annals of Operations Research*, 16:327–332, 1988.

[20] Kyungdaw Kang, Ilkyeong Moon, and Hongfeng Wang. A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Applied Mathematics and Computation*, 219(3):1287–1299, 2012.

[21] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European journal of operational research*, 141(2):241–252, 2002.

[22] Andrea Lodi, Silvano Martello, and Daniele Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1-3):379–396, 2002.

[23] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[24] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[25] Silvia Regina Morales and Reinaldo Morabito. Uma heurística simples e eficaz para resolver o problema do carregamento de paletes do produtor. *Gestão & Produção*, 4:52–76, 1997.

[26] Ana Moura and Andreas Bortfeldt. A two-stage packing problem procedure. *International Transactions in Operational Research*, 24(1-2):43–58, 2017.

[27] Chanaleä Munien, Shiv Mahabeer, Esther Dzitiro, Sharad Singh, Siluleko Zungu, and Absalom El-Shamir Ezugwu. Metaheuristic approaches for one-dimensional bin packing problem: A comparative performance study. *IEEE Access*, 8:227438–227465, 2020.

[28] Jonas Olsson, Torbjörn Larsson, and Nils-Hassan Quttineh. Automating the planning of container loading for atlas copco: Coping with real-life stacking and stability constraints. *European Journal of Operational Research*, 280(3):1018–1034, 2020.

[29] Marcela Quiroz-Castellanos, Laura Cruz-Reyes, Jose Torres-Jimenez, Claudia Gómez, Héctor J Fraire Huacuja, and Adriana CF Alvim. A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Computers & Operations Research*, 55:52–64, 2015.

[30] Luiz FO Moura Santos, Renan Sallai Iwayama, Luísa Brandão Cavalcanti, Leandro Maciel Turi, Fabio Emanuel de Souza Morais, Gabriel Mormilho, and Claudio B Cunha. A variable neighborhood search algorithm for the bin packing problem with compatible categories. *Expert Systems with Applications*, 124:209–225, 2019.

[31] Armin Scholl, Robert Klein, and Christian Jürgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627–645, 1997.

[32] Elsa Silva, José F Oliveira, and Gerhard Wäscher. The pallet loading problem: a review of solution methods and computational experiments. *International Transactions in Operational Research*, 23(1-2):147–172, 2016.

[33] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[34] Túlio AM Toffolo, Eline Esprit, Tony Wauters, and Greet Vanden Berghe. A two-dimensional heuristic decomposition approach to a three-dimensional multiple container loading problem. *European Journal of Operational Research*, 257(2):526–538, 2017.

[35] Emmanouil E Zachariadis, Christos D Tarantilis, and Chris T Kiranoudis. The pallet-packing vehicle routing problem. *Transportation Science*, 46(3):341–358, 2012.