Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Instituto de Matemática e Estatística

Anderson Zudio de Moraes

Meta-Heurísticas para o Problema de Empacotamento 2D e 3D

Rio de Janeiro
2018

Anderson Zudio de Moraes

Meta-Heurísticas para o Problema de Empacotamento 2D e 3D

Dissertação apresentada como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Orientador: Prof. Dr. Paulo Eustáquio Duarte Pinto
Coorientador: Prof. Dr. Igor Machado Coelho

Rio de Janeiro
2018

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC-A

M827 Moraes, Anderson Zudio de.
Meta-heurísticas para o problema de empacotamento 2D e 3D/ Anderson Zudio de Moraes. 2018.
124 f. : il.

Orientador: Paulo Eustáquio Duarte Pinto.
Coorientador: Igor Machado Coelho
Dissertação (Mestrado em Ciências Computacionais) - Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística.

1. Algoritmos computacionais - Teses. 2. Computação - Matemática. 3. Embalagens - Teses. I. Pinto, Paulo Eustáquio Duarte. II. Coelho, Igor Machado. III. Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística. IV. Título.

CDU 004.421

Patricia Bello Meijinhos - CRB7/5217 - Responsável pela elaboração da ficha catalográfica

_____ _____

Assinatura Data

Anderson Zudio de Moraes

Meta-Heurísticas para o Problema de Empacotamento 2D e 3D

Dissertação apresentada como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Aprovada em 19 de Fevereiro de 2018
Banca Examinadora:

Prof. Dr. Paulo Eustáquio Duarte Pinto (Orientador)
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Igor Machado Coelho (Coorientador)
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Fabiano de Souza Oliveira
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Jayme Luiz Szwarcfiter
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Luiz Satoru Ochi
Universidade Federal Fluminense

Rio de Janeiro
2018

RESUMO

ZUDIO, Anderson. *Meta-Heurísticas para o problema de Empacotamento 2D e 3D.* 2018. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro.

O problema de empacotamento consiste em empacotar, ortogonalmente e sem sobreposição, um conjunto de itens na menor quantidade de caixas possível. As versões bi e tridimensional do problema generalizam o caso bem conhecido unidimensional, um dos primeiros problemas da classe NP-Difícil. Esta dissertação estuda a versão clássica dos problemas de empacotamento bidimensional e tridimensional, considerando os casos com itens de orientação fixa e com rotação. O problema tem várias aplicações industriais, e o caso de orientação fixa se relaciona a outros problemas complexos como os de corte, repartição e agendamento. Várias heurísticas são propostas e combinadas com meta-heurísticas para resolver instâncias de grande porte do problema com soluções de boa qualidade. Extensivos testes computacionais são realizados com 820 instâncias padrões utilizadas em vários trabalhos na literatura. Os resultados computacionais obtidos pela melhor abordagem deste trabalho, denominada BRKGA-VCD, demonstram que o método proposto obtém soluções de qualidade superior ou equivalente aos algoritmos estado da arte encontrados na literatura, melhorando o melhor resultado conhecido para este conjunto de instâncias.

Palavras-chave: Meta-heurística. Empacotamento bidimensional. Empacotamento tridimensional. contêiner de tamanho único.

ABSTRACT

ZUDIO, Anderson. *Metaheuristics for the 2D and 3D bin packing problem.* 2018. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro.

The bin packing problem consists of orthogonally packing a set of boxes into the minimum number of bins without overlap. The two and three-dimensional bin packing problem generalizes the well known unidimensional bin packing problem, which was characterized as NP-Hard in the beginning of NP-C Theory. This dissertation studies the classic versions of two and three-dimensional case of bin packing with fixed and non-fixed orientated boxes. The bin packing problem has many industrial applications and relates to other complex problems like cutting, repartitioning and scheduling. This work proposes heuristics combined with metaheuristics to devise good quality solutions for large-scale instances. This work presents extensive computational tests with 820 standard literature instances. The computational results devised by the best algorithm proposed, called BRKGA-VCD, demonstrate that this method produces equivalent or better solution compared to state-of-art algorithms found in the literature, improving the overall best known solution quality for this set of instances.

Keywords: Metaheuristics. Two-dimensional Bin Packing. Three-dimensional Bin Packing. Single bin-size.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

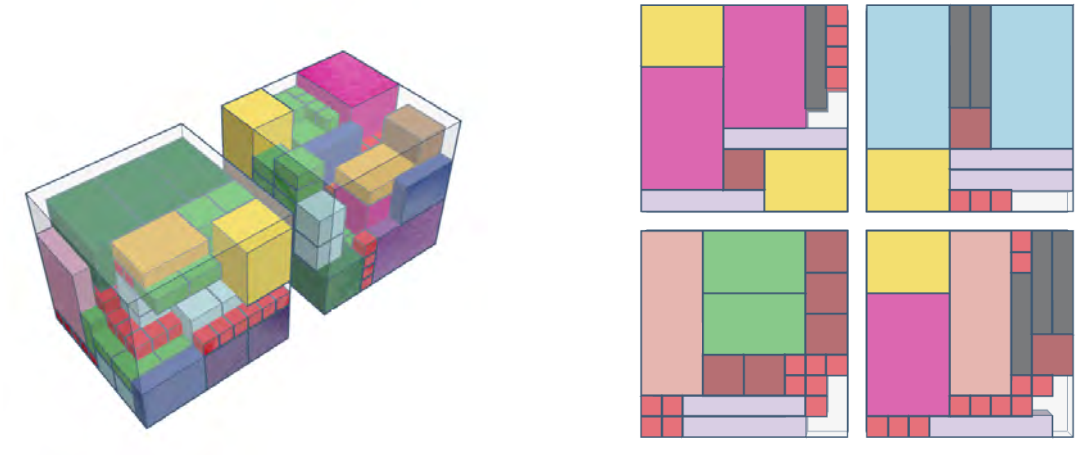
LISTA DE ALGORITMOS

SUMÁRIO

INTRODUÇÃO

O problema de empacotamento clássico tridimensional (3BP) consiste em empacotar ortogonalmente um conjunto de n itens em forma de paralelepípedos caracterizados por sua altura h_i, largura w_i e profundidade d_i, $i \in \{1, 2, ..., n\}$, no menor número possível de caixas homogêneas de altura $H \geq h_i$, largura $W \geq w_i$ e profundidade $D \geq d_i$. A quantidade de caixas é ilimitada e os itens não podem ser alocados com sobreposição. Deste modo, nenhuma outra restrição é imposta nesta versão do problema. O caso em que os itens têm orientação fixa são estudados separadamente do caso em que há rotação. De acordo com a topologia proposta por (WÄSCHER; HAUSSNER; SCHUMANN, 2007), o 3BP é classificado como *Single Stock-Size Cutting Stock Problem* (SSSCSP) ou *3D-Single Bin-Size Bin Packing Problem* (3D-SBSBPP). Quando consideramos que todo item i tem sua profundidade $d_i = D$, o 3BP passa a generalizar o problema de empacotamento clássico bidimensional (2BP ou 2D-SBSBPP). Neste caso, os itens são caracterizados por sua largura w_i e altura h_i. Assim, o 2BP consiste em empacotá-los no menor número possível de retângulos de dimensão $W \times H$. Analogamente, o 3BP e o 2BP generalizam o problema de empacotamento clássico bem conhecido unidimensional (1BP ou 1D-SBSBPP). Logo, esses problemas são da classe NP-difícil, pois o problema que generalizam foi um dos primeiros caracterizados como tal (KARP, 1972; GAREY; JOHNSON, 1979). A Figura 1 mostra exemplos de empacotamentos do 3BP e 2BP.

Figura 1 – Exemplos de empacotamento tridimensional e bidimensional.



(a) Solução de uma instância do 3BP.

(b) Solução de uma instância do 2BP.

Apesar do avanço da computação e da difusão de ambientes heterogêneos, os métodos exatos para os problemas de empacotamento, tais como (MARTELLO; PISINGER; VIGO, 2000) e (FEKETE; SCHEPERS; VEEN, 2007), são inviáveis de serem executados com instâncias grandes. O primeiro trabalho citado utiliza um método *Branch & Bound* para resolver o caso tridimensional que é difícil de ser paralelizado. Portanto,

uma das alternativas práticas para a solução destes problemas é a utilização de heurísticas e meta-heurísticas. Meta-heurísticas são aplicadas com sucesso em vários problemas de otimização combinatória (GLOVER; KOCHENBERGER, 2006). Nos problemas de empacotamento, heurísticas construtivas gulosas são capazes de obter soluções de boa qualidade para instâncias com um grande número de itens. As meta-heurísticas utilizam estratégias independentes do problema para escapar de ótimos locais. Assim, os algoritmos gulosos podem ser combinados com meta-heurísticas para construir heurísticas que são capazes de obter soluções de boa qualidade em um curto tempo de execução (PARREÑO et al., 2010a; GONÇALVES; RESENDE, 2013).

Neste contexto, esta dissertação descreve abordagens para a resolução do 3BP e do 2BP construídas através de meta-heurísticas combinadas com uma heurística gulosa que realiza o empacotamento dos itens. Os métodos propostos são detalhados através de diversos componentes, que, por sua vez, consideram que os itens podem ter orientação fixa ou não. Desta maneira, cada algoritmo proposto dispõe de diversas configurações que sintetizam diferentes versões. Em conjunto com o pseudocódigo de cada algoritmo introduzido neste trabalho, também foi feita uma análise de complexidade, com o objetivo de descrever o seu comportamento prático.

As possíveis versões de cada algoritmo proposto foram comparadas em um teste computacional preliminar. As estratégias selecionadas para cada abordagem foram baseadas nos melhores resultados obtidos para instâncias do 3BP sem rotação. Os parâmetros calibráveis de entrada da melhor versão de cada algoritmo foram ajustados empiricamente através de uma ferramenta computacional. Posteriormente, as abordagens propostas foram comparadas entre si em um teste computacional composto por 820 instâncias com itens de orientação fixa. Tais instâncias são a base de teste para diversos trabalhos da literatura. O método proposto que obteve os melhores resultados, denominado BRKGA-VCD, foi comparado com outros algoritmos no estado da arte encontrados na literatura em termos de qualidade da solução obtida para este conjunto de instâncias, considerando que os itens têm orientação fixa ou admitindo a possibilidade de rotação. Os dados mostram que o BRKGA-VCD provê soluções de alta qualidade, melhorando o melhor resultado reportado na literatura para este conjunto de instâncias.

Várias aplicações industriais têm interesse particular no 3BP. Dentre elas, podemos citar aplicações de carregamento de caminhões e aviões, armazenamento de mercadorias, empacotamento de encomendas, reconfiguração dinâmica de hardware e carregamento de paletes. A versão clássica, que consiste em empacotar itens com orientação fixa, é aplicada em problemas que exigem estabilidade. Por exemplo, quando um editor precisa preencher as páginas de um jornal, ele tem que considerar como rearranjar e posicionar os retângulos de textos e propagandas, ou seja, empacotamento de itens em que uma de suas faces deve permanecer obrigatoriamente para cima. Apesar de ser uma simplificação do mundo real, as heurísticas que encontram boas soluções para os problemas de empacotamento na versão clássica são aplicadas com sucesso em versões com restrições adicionais que modelam situações reais (PARREÑO et al., 2010b; KANG; MOON; WANG, 2012; LI; ZHAO; ZHANG, 2014; SARAIVA; NEPOMUCENO; PINHEIRO, 2015; BOYAR et al., 2016; MOURA; BORTFELDT, 2017). Além disso, o problema aparece como subparte de outros problemas mais complexos como os de agendamento, corte e particionamento.

Poucos trabalhos na literatura são encontrados para a versão tridimensional do problema, dentre tais só uma minoria permite a rotação dos itens. Grande parte dos métodos propostos são fornecidos em pseudocódigos sem qualquer implementação, o que impossibilita uma comparação real entre eles. Além disso, os trabalhos encontrados na literatura

não fornecem as respostas obtidas para cada instância separadamente. Esses trabalhos somente reportam respostas cumulativas de várias instâncias agrupadas. A base de teste padrão para o 3BP foi proposta por (MARTELLO; PISINGER; VIGO, 2000) através de um gerador que utiliza uma função de números pseudoaleatórios implementado pelos próprios autores. O gerador possui uma semente fixa, a fim de fornecer as mesmas instâncias independente do ambiente computacional que foi utilizado para gerá-las. Porém, estas instâncias não são disponibilizadas como um pacote previamente gerado, o que pode invalidar a análise comparativa feita por vários trabalhos científicos caso, acidentalmente, o usuário do gerador não configure-o corretamente, acarretando a possibilidade de gerar instâncias com itens diferentes entre os ambientes computacionais. Os autores do trabalho recente (HIFI; NEGRE; WU, 2014) alegam que outros autores reportam resultados melhores que os obtidos pelo algoritmo exato para alguns grupos de instâncias do 3BP sem rotação.

Alguns dos métodos propostos que resolvem o caso tridimensional têm aplicação direta no caso bidimensional. Porém, podemos observar que as instâncias da base de teste para o caso bidimensional utilizam uma quantidade menor de itens que as de teste tridimensional. Assim, os resultados para os métodos bidimensionais são obtidos em uma fração mínima de tempo comparada com o teste tridimensional, o que torna difícil o teste empírico na base bidimensional de métodos que já atingem boas soluções em pouco tempo de execução na base tridimensional. Além disso, podemos observar que os métodos com os melhores resultados reportados na literatura apresentam pouca variação no resultado final, tanto no 3BP quanto no 2BP, o que indica que grande parte dos resultados obtidos são mínimos locais com alta probabilidade de ser a solução ótima.

Desta maneira, este trabalho apresenta três novos algoritmos para o 3BP e o 2BP baseado nas meta-heurísticas GRASP, Busca Tabu e BRKGA. Os métodos foram testados e calibrados preliminarmente com uma parte da base de teste padrão da literatura composta por 320 e 500 instâncias para os casos tri e bidimensional, respectivamente. Os resultados obtidos pelo BRKGA proposto, referentes ao 3BP e 2BP com itens de orientação fixa ou permitindo sua rotação, para este conjunto de instâncias é melhor ou equivalente aos resultados reportados nos trabalhos comparados da literatura. Tais resultados são os melhores obtidos neste trabalho. Eles estão disponibilizados em conjunto com a própria instância[1], de maneira que qualquer leitor interessado possa construir graficamente, com facilidade, a configuração das caixas com seus respectivos itens.

O restante do trabalho é organizado da seguinte forma:

- Capítulo 1: descreve a fundamentação teórica da dissertação com conceitos de teoria da computação e meta-heurísticas. Este capítulo detalha e ilustra cada meta-heurística utilizada em conjunto com os seus componentes através de descrições que independem do problema abordado.

- Capítulo 2: apresenta a revisão bibliográfica dos problemas de empacotamento bidimensional e tridimensional, destacando importantes trabalhos da literatura que tratam de aspectos matemáticos ou abordagens para os problemas de empacotamento. É esboçada a prova uma prova de que o 3BP e o 2BP são da classe NP-Difícil. São descritos métodos exatos e abordagens alternativas para solução aproximada dos problemas.

[1] As instâncias e os melhores resultados podem ser obtidos no seguinte sítio eletrônico: <https://gitlab.com/AndersonZM/bin-packing-instance-sol>

- Capítulo 3: descreve e apresenta os algoritmos propostos para o 3BP e o 2BP. Este capítulo detalha cada método em conjunto com seus componente através de pseudocódigos. A descrição utiliza a fundamentação teórica do Capítulo 1. Além disso, o capítulo também fornece uma análise de complexidade para cada componente de cada algoritmo, a fim de descrever o seu comportamento prático e o seu processo de implementação.

- Capítulo 4: mostra os resultados computacionais deste trabalho. As heurísticas GRASP, Busca Tabu e BRKGA propostas são métodos que podem utilizar diferentes estratégias para implementar cada fase ou componente. O capítulo exibe um teste computacional preliminar que estabelece as melhores estratégias para cada heurística. As melhores versões do GRASP e da Busca Tabu foram calibradas empiricamente através de uma ferramenta computacional. Posteriormente, as abordagens foram comparadas entre si com 820 instâncias do 3BP e do 2BP sem rotação. Como o BRKGA proposto obteve os melhores resultados, este capítulo apresenta uma comparação desta abordagem com outros algoritmos encontrados na literatura para o 3BP e o 2BP com itens de orientação fixa ou que podem ser rotacionados. Além disso, o capítulo também descreve o ambiente computacional utilizado e as instâncias.

- Finalmente, apresenta-se as conclusões para a dissertação em conjunto com as propostas de trabalhos futuros.

1 CONCEITOS TEÓRICOS

Este capítulo introduz os conceitos teóricos utilizados neste documento. São descritas as heurísticas e meta-heurísticas de forma geral com ênfase nos mecanismos que cada uma aplica para sintetizar soluções, apresentando seus conceitos e pseudocódigos independentes do problema. Este capítulo também introduz os termos e componentes utilizados por esses métodos. Além disso, o capítulo descreve alguns conceitos fundamentais de teoria da complexidade computacional, que são utilizados para analisar teoricamente a performance dos métodos propostos.

1.1 Complexidade Computacional

A teoria da complexidade computacional é o estudo que classifica um problema computacional de acordo com sua dificuldade inerente. A análise da complexidade de um algoritmo está ligada diretamente com sua eficiência, de maneira que nos mostre o quanto de um determinado recurso de máquina o algoritmo requer quando executado, em função do tamanho da entrada processada. Geralmente, a complexidade é a medição do tempo de processamento ou consumo de memória. A complexidade de tempo se refere à quantidade de instruções primitivas da máquina que um algoritmo requer para sua execução completa. A complexidade de espaço ou memória é a análise da quantidade de células de memória que são utilizadas simultaneamente durante a execução do algoritmo. Os conceitos desta seção tomam como referência os livros (SZWARCFITER; MARKENZON, 2010; SZWARCFITER, 2018).

1.1.1 Notação O–grande

Existem diversas notações e maneiras de classificar a complexidade de um determinado recurso. Uma das classificações fundamentais é a de **pior caso**. O objetivo desta classificação é medir a utilização máxima do recurso em questão. Para medir a complexidade de um algoritmo é necessário associar o parâmetro relativo ao tamanho da entrada, denotado por n, com o recurso computacional da máquina requerido. Este processo é feito através de uma função $f(n)$, porém, na maior parte dos casos, é difícil ou até mesmo impossível obter esta função. Assim, utilizamos a notação $T(f(n))$ para denotar o termo de maior crescimento assintótico da função $f(n)$. Sabemos que $T(f(n))$ se aproxima do valor $f(n)$ com o crescimento da entrada. Desta maneira, temos que:

$$\lim_{n\to\infty} \frac{f(n)}{T(f(n))} = 1, \text{ onde } T(f(n)) \text{ é o termo de maior grau de } f(n).$$

Assim, temos uma família de notações que são amplamente utilizadas para evidenciar a ordem de grandeza deste termo de maior grau da função $f(n)$.

Definição 1.1.1. *Sejam as funções reais positivas f e g. Dizemos que $f(n) = \mathcal{O}(g(n))$ se, para alguma constante positiva C e um valor inicial $n_0, \forall n \geq n_0$ temos $f(n) \leq Cg(n)$.*

Na prática as funções escolhidas para $g(n)$ possuem apenas um termo que representa a ordem da função $f(n)$. Por exemplo, para quaisquer inteiros positivos A, B e C, se $f(n) = An^3 + Bn^2 + Cn$, então $f(n) = \mathcal{O}(n^k)$ com $k \geq 3$. Esta notação é amplamente difundida e utilizada na literatura.

Convencionou-se considerar como *tratáveis* os problemas para os quais a complexidade de pior caso pode ser limitada superiormente como um polinômio em n e *intratáveis* em caso contrário.

1.1.2 Notação Theta

Uma das notações utilizadas nesta dissertação para analisar a complexidade de pior caso dos métodos propostos é o **big-O** (O-grande). A outra é a notação Θ. Esta notação é utilizada para classificar a complexidade através de uma função que define seu limite inferior e superior, restringindo a utilização do recurso em questão. Assim, temos a seguinte definição:

Definição 1.1.2. *Sejam as funções reais positivas f e g. Dizemos que $f(n) = \Theta(g(n))$ se, para duas constantes positivas C_1, C_2 e um valor inicial $n_0, \forall n \geq n_0$, temos $f(n) \geq C_1 g(n)$ e $f(n) \leq C_2 g(n)$.*

Por exemplo, para qualquer inteiro A, se $f(n) = An^4$, então $f(n) = \Theta(n^4)$.

1.1.3 Problema de Decisão

O problema P é um problema de decisão quando o conjunto solução S para qualquer instância é composto somente pelos elementos *Sim* e *Não*. Por exemplo:

- Sejam o número A e um conjunto C de números. Existe algum número $B \in C$ maior que A?
- Sejam G um mapa de rotas entre cidades, $A \in G$ uma cidade e B um número. É possível visitar todas as cidades de G com uma rota de comprimento menor que B começando pela cidade A?
- Sejam os conjuntos finitos I de paralelepípedos e C de cubos homogêneos. É possível empacotar ortogonalmente todos os paralelepípedos de I nos cubos do conjunto C sem sobreposição?

1.1.4 Problema de Otimização

Sejam um problema P, o conjunto $S = \{s_1, s_2, \cdots, s_n\}$ com as n soluções viáveis e a função objetivo $f : S \to \mathbb{R}$ que associa cada solução $s_i \in S$, $i = 1, 2, \cdots, n$, com um número real. O problema P é um problema de otimização de minimização se ele consiste em determinar uma solução $s^\star \in S \mid f(s^\star) \leq f(s), \forall s \in S$, denominada solução ótima. Analogamente, P é um problema de otimização de maximização se ele procura determinar $s^\star \in S \mid f(s^\star) \geq f(s), \forall s \in S$. Ou seja, P é um problema de otimização se ele busca uma solução viável s' que maximiza ou minimiza o valor de uma função objetivo f associada ao problema.

Considere o conjunto $X = \{x_1, x_2, \cdots, x_m\}$ das m variáveis que representam as soluções viáveis em S. Os problemas de otimização podem ser divididos em dois tipos:

- Otimização contínua: são os problemas onde pelo menos uma variável $x \in X$ é contínua, ou seja, pode assumir um número infinito não enumerável de valores.

- Otimização combinatória: são os problemas onde toda variável $x \in X$ é discreta, isto é, pode assumir um número finito ou infinito contável de valores.

Desta maneira, problemas de otimização combinatória têm um número enumerável de soluções viáveis. Todo problema de otimização tem um problema de decisão associado. Por exemplo, os problemas de decisão da seção anterior estão associados aos seguintes problemas de otimização:

- Seja um conjunto de números S, determine o maior número $A \in S$.

- Sejam G um mapa de rotas entre cidades e $A \in G$ uma cidade, encontre a comprimento da menor rota que visita todas as cidades de G começando em A.

- Sejam os conjuntos finitos I de paralelepípedos e C de cubos homogêneos, determine o menor número possível de cubos em C que empacotam ortogonalmente todos os paralelepípedos de I sem sobreposição.

1.1.5 Classes de Problemas

Os problemas computacionais são classificados de acordo com sua dificuldade inerente. Portanto, diversas classes são definidas para identificar o quão eficiente um determinado algoritmo pode ser para dado problema de acordo com um recurso computacional. Algumas das classes fundamentais são definidas abaixo:

- Classe P (**Polynomial Time**): consiste dos problemas de decisão para os quais existe pelo menos um algoritmo determinístico com complexidade polinomial que o resolve. Ou seja, são os problemas de decisão em que há uma máquina de *Turing* determinística que resolve o problema em um número de passos polinomial em relação ao tamanho da entrada.

- Classe NP (**Nondeterministic Polynomial Time**): é a classe fundamental que consiste dos problemas de decisão em que a resposta *Sim* pode ser verificada de forma eficiente. Isto é, para qualquer instância que responde *Sim*, a resposta pode ser provada por um certificado que pode ser verificado por uma máquina de *Turing* determinística em um número de passos polinomial em função do seu tamanho.

- Classe NP-Difícil: é a classe que consiste dos problemas P tal que todo problema $P' \in NP$ pode ser reduzido em um número de passos polinomial para P. Ou seja, podemos transformar, sempre que existir, uma solução de P' em uma solução de P através de uma máquina de *Turing* determinística com um número de passos polinomial em relação ao seu tamanho. Desta maneira, resolver P' é equivalente a resolver P.

- Classe NP-Completo: é a classe que consiste dos problemas de decisão que pertencem às classes NP e NP-Difícil.

1.2 Heurísticas

Heurística é um algoritmo que resolve um determinado problema sem garantias sobre a qualidade da solução encontrada ou sobre seu desempenho. Heurística vem do termo *Eureka*, relativo à capacidade de aproveitar a percepção humana sobre um problema para resolvê-lo de forma aproximada. Geralmente, heurísticas sacrificam qualidade, completude, acurácia ou precisão por tempo de execução. Esse tipo de abordagem é uma boa alternativa quando o algoritmo exato que resolve o problema exige um alto tempo de processamento ou não é conhecido. Geralmente, o algoritmo exato consegue resolver instâncias pequenas do problema. Porém, o seu tempo de execução fica impraticável quando os parâmetros de entrada aumentam, devido à complexidade do algoritmo. Assim, heurísticas são aplicadas, a fim de obter soluções boas o suficiente na prática, mas que não são necessariamente ótimas.

1.2.1 Algoritmos Gulosos e Gulosos Adaptativos

Heurísticas que constroem soluções viáveis desde o princípio são chamadas de **heurísticas construtivas**. Algoritmos **gulosos e gulosos adaptativos** são exemplos de heurísticas construtivas. Um algoritmo guloso é um método que, a cada iteração, constrói gradativamente a solução através de candidatos ranqueados por seu potencial de qualidade, onde um candidato é uma parte ou pedaço que forma parcialmente uma solução (RESENDE; RIBEIRO, 2016).

Durante a construção, o algoritmo guloso avalia todos os possíveis candidatos e os ordena de acordo com seu potencial ganho de qualidade. Desta maneira, a cada iteração do algoritmo o melhor candidato é selecionado. Entretanto, caso o candidato seja inviável, ele é descartado e o melhor candidato viável é selecionado. O processo finaliza quando uma solução é totalmente construída. O Algoritmo 1 é um esquema geral de um algoritmo guloso para um problema de otimização, onde g é a função que avalia o potencial de qualidade de um candidato.

Algoritmo 1 Construtivo Guloso

função Guloso()
1: $s \leftarrow \emptyset$ ▷ Solução construtiva começa do princípio
2: Lista $\leftarrow$ Vazio
3: **para** cada possível candidato c **faça**
4: Lista $\leftarrow$ Lista $\cup$ $(c, g(c))$ ▷ Lista recebe candidato c e seu potencial
5: Ordena Lista pelo potencial de qualidade
6: **enquanto** s não está completo **faça**
7: $c' \leftarrow$ melhor candidato da Lista ▷ Candidatos inviáveis são descartados
8: $s \leftarrow s \cup \{c'\}$ ▷ Constrói s parcialmente com c'
9: **retorna** s ▷ Retorna a solução construída

Em alguns problemas de otimização, a seleção de um candidato altera o potencial de qualidade dos outros candidatos. Algoritmos gulosos adaptativos são a alternativa para essa situação. Um algoritmo guloso adaptativo realiza o mesmo procedimento de um algoritmo guloso, mas cada vez que um candidato é selecionado, os outros candidatos são reavaliados e reordenados. Desta maneira, as iterações futuras se adaptam, de acordo com a qualidade atual do candidato, perante aos escolhidos anteriormente. O Algoritmo 2

descreve o pseudocódigo de um guloso adaptativo, onde g é a função que avalia o potencial de qualidade de um candidato.

Algoritmo 2 Construtivo Guloso Adaptativo

função Guloso()
1: $s \leftarrow \emptyset$ ▷ Solução construtiva começa desde o princípio
2: **enquanto** s não está completo **faça**
3: Lista $\leftarrow \emptyset$
4: **para** cada possível candidato c **faça**
5: Lista $\leftarrow$ Lista $\cup (c, g_s(c))$ ▷ Lista recebe candidato c e seu potencial
6: Ordena Lista pelo potencial de qualidade
7: $c' \leftarrow$ melhor candidato da Lista ▷ Candidatos inviáveis são descartados
8: $s \leftarrow s \cup \{c'\}$ ▷ Constrói s parcialmente com c
9: **retorna** s ▷ Retorna a solução construída

Nos problemas de empacotamento, a heurística construtiva proposta nesta dissertação para efetuar o empacotamento dos itens é um algoritmo guloso. Dada uma configuração de caixas, o próximo passo da heurística escolhe um empacotamento que gera a melhor configuração de caixas no atual momento da execução, de acordo com um critério para encher as caixas, independente da qualidade das configurações que possam ser geradas nos passos futuros. Assim, uma única solução é construída desde o princípio através do posicionamento de um item em uma caixa em cada passo.

1.2.2 Busca Local

Sejam um problema de otimização combinatória P, f sua função objetivo associada e S o conjunto de soluções viáveis. Uma **busca local** é uma heurística iterativa que tenta encontrar uma solução melhor a cada iteração através de uma estratégia que modifica parcialmente a solução da iteração atual. A busca local começa por uma solução inicial s^0 na iteração 0, e, a cada iteração $k \in \{1, 2, \ldots, n\}$ subsequente, uma nova solução $s^k \in S$ é determinada de modo que:

- $f(s^k) > f(s^{k-1})$ caso P seja um problema de maximização.
- $f(s^k) < f(s^{k-1})$ caso P seja um problema de minimização.

A busca local finaliza na iteração em que não é possível atender o critério acima. Isto é, a busca local recebe como parâmetro de entrada uma solução inicial s^0 e tenta encontrar soluções melhores, tal que a solução da iteração atual s^k seja melhor que a solução da iteração anterior s^{k-1} até quando for possível obter melhora. A maneira pela qual as soluções são modificadas e os dois métodos mais comuns para guiar o processo de busca da heurística são descritos nas próximas seções.

1.2.2.1 Vizinhança

Seja uma solução $s \in S$, onde S é o conjunto de soluções do problema de otimização combinatória P. Uma **vizinhança** $V(s)$ é um conjunto de soluções obtido através de uma relação que modifica s parcialmente. Esta relação, chamada de estrutura de vizinhança $V : S \rightarrow S$, pode ser implementada através de um conjunto de n tuplas, geralmente

de números inteiros, e um algoritmo ou regra que utiliza a tupla como parâmetro de entrada para modificar a solução s. Cada elemento deste conjunto, combinado com o algoritmo que modifica s, é denominado **movimento**. Desta maneira, quando aplicamos o movimento M_k em s, obtemos uma nova solução s^k, $k \in \{1, 2, \cdots, n\}$. Portanto, a vizinhança $V(s) = \{s^1, s^2, \cdots, s^n\}$ sobre a estrutura de vizinhança V com n movimentos é obtida aplicando cada movimento M_k na solução s. As soluções $s^1, s^2, \cdots, s^n \in V(s)$ são chamadas de **vizinhos** de s.

Em uma busca local, o processo que modifica a solução da iteração corrente utiliza uma estrutura de vizinhança composta de movimentos que geram uma vizinhança. Desta maneira, a busca local utiliza a função objetivo associada ao problema P para avaliar cada vizinho da iteração atual, selecionando algum vizinho que melhore a iteração corrente, caso exista. Observe que uma solução pode ter vizinhos que são soluções inviáveis, isto é, não atendem alguma restrição de P. Estas soluções devem ser descartadas por alguma estratégia, geralmente uma grande penalidade associada à função objetivo, ou consertadas através de um método que recebe uma solução inviável e gera uma solução viável. Nesta dissertação, o 3BP e o 2BP não precisam de tais técnicas, pois sempre são trabalhados com soluções viáveis. No restante deste documento este detalhe não será mais discutido.

1.2.2.2 First Improvement vs. Best Improvement

Best Improvement e **First Improvement** são estratégias de exploração de vizinhança aplicadas em busca local que ditam como o processo de seleção por soluções melhores deve agir no momento em que uma solução é melhorada. O *First Improvement* recebe como parâmetro de entrada a solução da iteração corrente da busca local para gerar os seus vizinhos, de acordo com a estrutura de vizinhança V dada como segundo parâmetro de entrada. Após aplicar o movimento no parâmetro de entrada, o processo verifica se o vizinho gerado é melhor através da função objetivo associada ao problema. Caso haja melhora, o *First Improvement* retorna imediatamente a solução vizinha gerada. O Algoritmo 3 descreve o procedimento realizado pelo *First Improvement* para um problema de minimização.

Algoritmo 3 First Improvement

função FirstImprovement(**Solução:** s; **Estrutura de vizinhança:** V)
1: **para** cada $s_k \in V(s)$ **faça** ▷ Para cada vizinho de s
2: **se** $f(s_k) < f(s)$ **então** ▷ Se s_k for melhor que s
3: **retorna** s_k ▷ Seleciona o primeiro vizinho que melhora s
4: **retorna** s ▷ Se um vizinho melhor s não existir, retorna o próprio.

O *Best Improvement* funciona de maneira similar ao *First Improvement*, recebendo também como parâmetro de entrada a solução da iteração corrente da busca local e uma estrutura de vizinhança. A diferença é que o *Best Improvement* aplica todos os movimentos na solução. Assim, o processo verifica e mantém o melhor vizinho gerado da solução atual. Portanto, o *Best Improvement* tem o potencial de avaliar um número maior de soluções. O Algoritmo 4 detalha este procedimento para um problema de minimização.

No decorrer do processo, não há nenhuma garantia de que o *Best Improvement* encontre soluções melhores que o *First Improvement*, ou até mesmo que ambos vão encontrar soluções diferentes. Os dois métodos dependem do problema, da estrutura de vizinhança e das características do conjunto solução da instância sendo resolvida. A vantagem do

Algoritmo 4 Best Improvement

função BestImprovement(**Solução:** s; **Estrutura de vizinhança:** V)

1: $s^* \leftarrow s$ ▷ Melhor solução s^* recebe solução s
2: **para** cada $s_k \in V(s)$ **faça** ▷ Para cada vizinho de s
3: **se** $f(s_k) < f(s^*)$ **então** ▷ Se s_k for melhor que a melhor solução encontrada
4: $s^* \leftarrow s^k$ ▷ Mantém a melhor solução em s^*
5: **retorna** s^* ▷ Retorna a melhor solução encontrada

Best Improvement é o seu potencial de encontrar soluções de qualidade superior ao *First Improvement*, enquanto o *First Improvement* tem o potencial de retornar sua solução final em menor tempo de execução. Quando uma nova heurística que utiliza busca local é proposta para um problema de otimização combinatória, é comum efetuar um teste computacional empírico preliminar para avaliar qual das duas estratégias funciona melhor para dado problema. Finalmente, o Algoritmo 5 apresenta a busca local, onde o procedimento *Improvement* pode ser o *First Improvement* ou o *Best Improvement*.

Algoritmo 5 Busca Local

função BuscaLocal(**Solução inicial:** s_0; **Estrutura de vizinhança:** V)

1: $s^* \leftarrow s_0$ ▷ Melhor solução s^* recebe solução inicial
2: **faça**
3: $s^* \leftarrow$ IMPROVEMENT(s^*, V) ▷ *Best Improvement* ou *First Improvement*
4: **enquanto** Houver melhora em s^*
5: **retorna** s^* ▷ Retorna a melhor solução encontrada

Existem outras estratégias que podem ser aplicadas no lugar do *Best Improvement* ou *First Improvement*, mas não foram utilizadas neste trabalho. Um exemplo é o *Multi Improvement*, que é baseado na exploração de todos vizinhos independentes ou movimentos independentes de uma dada vizinhança. O conceito de independência é fundamentada em movimentos que podem ser aplicados simultaneamente na mesma solução. O *Multi Improvement* já foi utilizado com sucesso em (RIOS et al., 2018), onde os autores resolvem o problema de miníma latência com uma implementação em GPU (*Graphics Processing Unit*). Os experimentos computacionais do trabalho citado mostram que o *Multi Improvement* é capaz de obter soluções melhores ou equivalentes comparado com o *Best Improvement* em uma implementação em GPU para o problema de mínima latência.

1.2.2.3 Ótimo Local vs. Ótimo Global

Sejam um problema de otimização combinatória P, sua respectiva função objetivo f e o conjunto de soluções viáveis S. Uma solução $s^* \in S$ é chamada de ótimo global quando:

- $\nexists s' \in S \quad | \quad f(s') > f(s^*)$ caso o problema P seja de maximização
- $\nexists s' \in S \quad | \quad f(s') < f(s^*)$ caso o problema P seja de minimização.

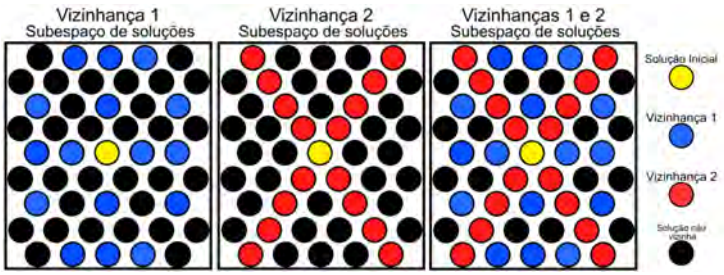
Em outras palavras, s^* é um **ótimo global** quando ela é a melhor solução viável de P. Considere a busca local com *Best Improvement* H para P que utiliza uma estrutura de vizinhança V para gerar uma vizinhança. Após aplicar H com uma solução inicial

$s^0 \in S$, a solução $s'' \in S$ é obtida com um número arbitrário de iterações de H. Portanto, qualquer vizinho $v \in V(s'')$ é pior ou igual a s''. Assim, s'' é denominado de **ótimo local** conforme a vizinhança $V(s'')$ da estrutura de vizinhança V. Um ótimo local s'' é uma solução que não pode ser melhorada por alguma estrutura de vizinhança. É importante ressaltar que um ótimo local em relação a dada vizinhança pode não ser ótimo local de uma vizinhança diferente. Observe que o ótimo local s'' pode ser ou não um ótimo global de P. Assim, um ótimo global s^* é uma solução que é ótimo local em relação a todas as possíveis vizinhanças $V(s) \subseteq S$ para qualquer solução $s \in S$.

1.2.3 Variable Neighborhood Descent

Muitos problemas de otimização combinatória apresentam representações de soluções com diversas estruturas de vizinhanças que, apesar de serem capazes de atingir os mesmos ótimos locais, verificam diferentes soluções dado o mesmo ponto de partida. Além disso, muitas vezes uma única estrutura de vizinhança não é capaz de percorrer o espaço de soluções para encontrar determinados ótimos globais. Desta maneira, essas estruturas combinadas são capazes de abranger um maior números de soluções. Esta é a ideia da heurística **Variable Neighborhood Descent** (VND). O método combina diferentes estruturas de vizinhança através de uma estratégia que utiliza mais de uma busca local (CAPOROSSI; HANSEN; MLADENOVIĆ, 2016). A Figura 2 apresenta graficamente esta ideia. O gráfico mostra um subespaço de soluções disposto em um plano, destacando duas vizinhanças de um mesmo ponto de partida com as cores azul e vermelho. Cada estrutura de vizinhança busca em direções diferentes, onde soluções distintas são avaliadas. A figura mostra que, ao combinarmos as duas vizinhanças, uma quantidade maior de soluções são avalidas.

Figura 2 – Representação gráfica de um subespaço de soluções com duas vizinhanças distintas de uma mesma solução inicial.



A estratégia aplicada pelo VND é percorrer o espaço de soluções em diferentes direções até que a solução encontrada seja ótimo local em relação a todas as estruturas de vizinhança combinadas. Este processo amplifica e diversifica a busca por ótimos locais. Enquanto uma execução da busca local é capaz de atingir somente um ótimo local, uma execução do VND é capaz de atingir vários ótimos locais. Dessa forma, o VND tem uma probabilidade menor de ficar preso em um determinado local do espaço de soluções e a probabilidade de encontrar um ótimo global aumenta. Porém, o tempo de execução aumenta devido ao maior número de soluções avaliadas.

Dependendo do ponto de partida, uma busca local pode ser incapaz de abranger todo o espaço de soluções do problema, devido à direção que ela busca. Por exemplo, se visualizarmos o espaço de soluções como um tabuleiro de xadrez, tal que cada casa é uma solução, uma busca local que implementa uma estrutura de vizinhança que percorre o

espaço da mesma maneira que um bisco percorre o tabuleiro não consegue atingir metade das soluções. Dessa forma, estamos lidando com um algoritmo que pode ser incapaz de determinar o ótimo global ou atingir certas soluções de alta qualidade. Uma vantagem do VND é possibilitar a combinação de diferentes estruturas de vizinhança para que este caso não aconteça. Mesmo que possamos identificar uma estrutura de vizinhança que consegue abranger todo o espaço de soluções, diversos movimentos podem ser necessários para atingir certas soluções. Quando combinamos diferentes estruturas de vizinhança, o número de movimentos para atingir vizinhos distantes decresce.

O VND recebe uma solução inicial s_0 e n estruturas de vizinhança como parâmetros de entrada. Começando pela estrutura de vizinhança V_1, o procedimento realiza uma busca local com cada V_k sequencialmente, $k \in \{1, 2, \ldots, n\}$. Se houver melhora na solução atual, então o processo iterativo é imediatamente reiniciado em V_1, tal que as buscas são realizadas com a nova solução melhorada. O processo finaliza sua execução quando a solução atual s não pode ser melhorada, isto é, s é ótimo local em relação a cada vizinhança $V_k(s)$.

A performance do VND está diretamente ligada à ordem das estruturas de vizinhança utilizadas. Se cada vizinhança $V_k(s)$ é composta por soluções distintas, então qualquer ordem entre elas é justificável. Entretanto, é uma boa prática utilizar as estruturas de vizinhança de acordo com o seu número de movimentos, pois as estruturas de vizinhança com menos movimentos têm o potencial de achar seu ótimo local em menor tempo de execução. Além disso, se $V_i(s) \subset V_j(s)$, então um ótimo local de V_j também é ótimo local de V_i. Portanto, não faz sentido executar V_i depois de V_j. Como a qualidade das soluções aumenta com o número de iterações, a busca nessas duas vizinhanças é justificada se a busca local em $V_i(s)$ é suficientemente rápida para filtrar soluções com má qualidade de serem executadas na vizinhança $V_j(s)$. Desta maneira, um ganho de performance substancial pode ser obtido, devido à rápida melhora que V_i pode obter nas soluções quando comparada a V_j. O Algoritmo 6 detalha o VND para um problema de otimização combinatória, onde a função *BuscaLocal* é o Algoritmo 5.

Algoritmo 6 Variable Neighborhood Descent

função VND(**Solução inicial:** s_0, **Vetor de estruturas de vizinhança:** V)

```
1: s* ← s0                                          ▷ Melhor solução.
2: k ← 1                                            ▷ O processo começa em V1
3: enquanto k ≤ |V| faça
4:     s* ← BuscaLocal(s*, Vk)                      ▷ Busca um ótimo local conforme Vk(s*)
5:     se Houver melhora em s* então
6:         k ← 1
7:     senão
8:         k ← k + 1
9: retorna s*                                       ▷ Retorna a melhor solução
```

1.2.4 Representação da Solução

Um componente importante em heurísticas para problemas de otimização combinatória é a **Representação da Solução**. Este componente deve ser definido pelo programador, com o objetivo de representar computacionalmente a solução de um determinado problema através de variáveis discretas. As estruturas mais comuns utilizadas são vetores

de números inteiros, vetores de *bits* e permutações. Após definir a representação da solução, uma estrutura de vizinhança pode ser definida para sintetização de uma heurística.

Por exemplo, considere o problema clássico da mochila que é NP-Difícil: dada uma mochila de capacidade B e um conjunto $I = \{a_1, a_2, \cdots, a_n\}$ de n itens, cada um caracterizado por seu peso $w_i > 0$ e seu valor $v_i > 0$, $1 \leq i \leq n$, determinar o subconjunto de itens tal que a soma de valores seja máxima e de modo que a soma dos pesos não ultrasse a capacidade B. Uma possível representação da solução do problema da mochila é um vetor de n *bits* que define quais itens estão sendo carregados: se o i-ésimo *bit* for 1, então o i-ésimo item está na mochila, caso contrário ele não está. Desta maneira, uma estrutura de vizinhança pode ser a simples troca de dois bits da representação, onde os movimentos dessa estrutura podem ser representados pelo conjunto de tuplas: $\{(i, j) \mid 1 \leq i < j \leq n\}$.

Nos problemas de empacotamento, diversas representações de soluções são utilizadas. Uma delas utiliza soluções codificadas. Representações de soluções codificadas precisam de um algoritmo determinístico que converte a solução codificada em uma solução que utiliza sua respectiva representação não codificada. Uma das vantagens de utilizar soluções codificadas é poder realizar operações e métodos que não geram soluções inviáveis. Por exemplo, uma representação da solução adequada para um problema pode ter várias soluções inviáveis como é o caso do problema da mochila. A estrutura de vizinhança especificada no exemplo do parágrafo anterior pode gerar soluções inviáveis facilmente. Algumas heurísticas precisam utilizar soluções codificadas para aplicarem suas estratégias de modo eficiente. Este é o caso de um dos métodos do Capítulo 4 desta dissertação que utiliza a meta-heurística descrita na Seção 1.3.4.

1.3 **Meta-heurísticas**

Meta-heurísticas são algoritmos que sintetizam heurísticas para um problema de otimização, independente dos detalhes do mesmo. Meta-heurísticas são compostas por estratégias e mecanismos que são capazes de encontrar soluções sem ficar restritos ou presos em ótimos locais de baixa qualidade, independente das características do problema aplicado. Inúmeros trabalhos na literatura utilizam esses métodos para resolver problemas de otimização combinatória que são da classe NP-Difícil. Devido à simplicidade de alguns mecanismos desses métodos, diversas vezes meta-heurísticas são combinadas de forma a sintetizar métodos híbridos envolvendo várias estratégias diferentes para buscar soluções para determinado problema. Assim, esses algoritmos são ferramentas robustas para serem aplicadas na resolução prática de problemas de otimização combinatória, sendo uma alternativa quando o respectivo algoritmo exato não é conhecido ou exige um alto tempo de execução. (GLOVER; KOCHENBERGER, 2006)

As meta-heurísticas realizam iterações sucessivas de modo a manter a melhor solução encontrada durante a execução. Alguns desses métodos introduzem parâmetros que precisam ser calibrados através de testes empíricos. Essas variáveis podem assumir valores fixos durante todo o processo de execução da heurística ou podem ser modificadas dinamicamente durante a execução, de acordo com algum critério. Tais parâmetros geralmente impactam na performance e na eficácia do algoritmo, seja determinando a probabilidade de escolha num componente ou limitando o tamanho máximo de uma determinada estrutura. O processo iterativo desses algoritmos utiliza um **Critério de Parada** para finalizar a execução. Geralmente este critério de parada é determinado tempo de execução, número de iterações máxima, alvo de qualidade conhecido para a instância do problema ou número de iterações sem melhoras na melhor solução obtida. Porém, outros critérios de parada

mais complicados podem ser utilizados. Para os problemas de empacotamento, os trabalhos citados no Capítulo 2 desta dissertação utilizam o tempo de execução ou número máximo de iterações como critério de parada em seus testes empíricos. Os algoritmos apresentados nesta seção sempre generalizam o critério de parada.

1.3.1 Greedy Randomized Adaptive Search Procedure

O **Greedy Randomized Adaptive Search Procedure** (GRASP) é uma metaheurística composta de duas fases no seu processo iterativo. Cada iteração do método é independente das anteriores. Uma nova solução é construída na primeira fase e melhorada na segunda. Uma referência atual para o GRASP e suas aplicações é o livro (RESENDE; RIBEIRO, 2016), no qual esta seção se baseia.

1.3.1.1 Fase de construção

A primeira fase do GRASP, chamada de fase de construção, recebe como parâmetro de entrada um número real $0 \leq \alpha \leq 1$ para realizar um método construtivo que obtém uma solução viável do problema. A ideia desta fase é utilizar um **algoritmo semi-guloso** ou **guloso randomizado** para construir uma solução desde o princípio. Então, o algoritmo puramente guloso adaptativo para o problema consideraria somente o melhor candidato no determinado momento de escolha para construir a solução. Deste modo, a fase de construção do GRASP utiliza o parâmetro α para introduzir aleatoriedade ao guloso adaptativo. Assim, o algoritmo semi-guloso deve sempre considerar uma lista com os $100\alpha\%$ melhores candidatos cada vez que precisar escolher um na construção da solução. A lista dos melhores candidatos é denominada RCL (*Restricted Candidate List*) e seu conceito é apresentado graficamente na Figura 3. A figura apresenta um vetor de candidatos ordenados pelo potencial ganho de qualidade na solução, onde o processo construtivo deve escolher aleatoriamente qualquer candidato na RCL. O α é um parâmetro que deve ser devidamente calibrado. Caso seu valor seja 0 a fase de construção se torna o algoritmo guloso adaptativo, pois somente o melhor candidato vai estar presente na RCL. Caso o valor de α seja 1, o processo se torna completamente aleatório, pois a RCL será composta por todos os candidatos. O Algoritmo 7 mostra o pseudocódigo desta fase com o semi-guloso, onde a função g avalia o potencial de ganho de qualidade dos candidatos para a solução que está sendo construída.

Algoritmo 7 Semi-Guloso ou Guloso Randomizado

Requerimento: Parâmetro α no intervalo $[0, 1]$
função Semi-Guloso(**Número real:** α)
1: $s \leftarrow \emptyset$ ▷ Solução construtiva começa desdo princípio.
2: **enquanto** s não está completo **faça**
3: Lista $\leftarrow$ Vazio ▷ Lista de candidatos
4: **para** cada possível candidato c **faça** ▷ Cada passo seleciona um c para s
5: Lista $\leftarrow$ Lista $\cup$ $(c, g(c))$ ▷ Adiciona c pela ordem de seu potencial $g(c)$
6: RCL $\leftarrow$ $100\alpha\%$ melhores candidatos da Lista
7: $c' \leftarrow$ Candidato aleatório da RCL
8: $s \leftarrow s \cup \{c'\}$ ▷ Adiciona o candidato em s
9: **retorna** s ▷ Retorna a solução construída

Figura 3 – Representação gráfica do vetor de candidatos e do RCL delimitado pelo α.

1.3.1.2 Fase de Melhora

Na segunda fase, chamada de fase de melhora, a solução obtida na fase de construção é aprimorada através de uma busca local que pode utilizar Best Improvement ou First Improvement. A ideia principal do GRASP é construir soluções de maneira gulosa mantendo aleatoriedade, tal que as soluções geradas tenham uma boa qualidade média para a busca local finalizar em poucas iterações, e sejam diversificadas para encontrar diversos ótimos locais do espaço de soluções. Portanto, o GRASP recomeça através de uma nova solução a cada iteração buscando novos ótimos locais obtidos nesta fase. Assim, as iterações do GRASP são **independentes**, onde o processo deve manter a melhor solução encontrada. O Algoritmo 8 apresenta o pseudocódigo do GRASP para um problema de minimização, onde f é a função objetivo associada ao problema de otimização combinatória que está sendo resolvido, a função *Semi-Guloso* é o processo semi-guloso do Algoritmo 7 e a *BuscaLocal* é o Algoritmo 5 com Best Improvement ou First Improvement.

Algoritmo 8 Greedy Randomized Adaptive Search Procedure

Requerimento: Parâmetro α no intervalo $[0, 1]$
função GRASP(**Número Real:** α)
1: $s^* \leftarrow \emptyset$ ▷ Melhor solução começa vazia com $f(s^*) = \infty$
2: **enquanto** Critério de parada não satisfeito **faça**
3: $s \leftarrow$ Semi-Guloso(α) ▷ Constrói uma solução gulosa aleatória.
4: $s \leftarrow$ BuscaLocal(s) ▷ Best ou First
5: **se** $f(s) < f(s^*)$ **então** $s^* \leftarrow s$ ▷ Mantém a melhor solução
6: **retorna** s^* ▷ Retorna a melhor solução encontrada

1.3.2 Busca Tabu

A **Busca Tabu** (BT) é uma meta-heurística determinística. O seu processo iterativo começa em uma solução inicial, buscando novas soluções a cada iteração, sem estratégias de recomeço e randomização. O método foi inicialmente proposto em (GLOVER, 1986). O conteúdo desta seção usa como referência o trabalho (GLOVER; LAGUNA, 2013).

A ideia principal da busca tabu é utilizar mecanismos que exploram o uso de uma **memória adaptativa** para controlar a busca no espaço de soluções de um problema de otimização combinatória, de modo que o processo iterativo é guiado em diferentes direções para encontrar novos ótimos locais. Portanto, uma estrutura chamada **Tabu** é introduzida para restringir a direção de busca. A busca tabu recebe como parâmetro de entrada uma solução inicial s_0 viável, uma estrutura de vizinhança V e o tamanho máximo T_{max} da lista tabu. A solução inicial pode ser gerada aleatoriamente ou construída através de uma heurística apropriada para o problema. O tamanho T_{max} é um parâmetro que deve ser calibrado. A cada iteração do método o melhor vizinho da solução corrente s

é selecionado, independente de ser melhor que o próprio s. Porém, algumas soluções na vizinhança $V(s)$ são proibidas de serem escolhidas pela lista tabu, formalizando uma nova vizinhança $V^*(s) \subset V(s)$ restringida. O tabu pode ser composto de movimentos ou soluções. Entretanto, guardar soluções inteiras na lista tabu pode exigir um alto consumo de memória e uma perda de performance considerável no acesso.

Após escolher o melhor vizinho s', verifica-se a lista tabu em busca do movimento que gerou s' ou o próprio s' (dependendo da implementação). Se s' é tabu, então s' deve ser desconsiderado e um novo melhor vizinho que satisfaça as restrições do tabu deve ser selecionado. No final da iteração, o melhor vizinho escolhido será a próxima solução corrente e a melhor solução encontrada em todas as iterações da meta-heurística é mantida. Neste momento, a lista tabu deve ser atualizada para restringir a solução encontrada nas iterações futuras através da memorização do movimento ou da própria solução. Se a lista estiver cheia de acordo com T_{max}, uma restrição deve ser removida. Esta remoção pode utilizar um critério como escolher o elemento mais antigo, o de maior qualidade ou uma estratégia mais complexa. Finalmente, critérios de **aspiração** podem ser sintetizados. Estes critérios são regras que criam exceções na restrição do tabu. Assim, uma solução banida pode ser escolhida de qualquer maneira. Algumas estratégias para este mecanismo são classificadas como:

- *Global* – se a solução tabu melhorar a melhor solução já encontrada.
- *Regional* – se a solução tabu apresentar uma vizinhança promissora, explorar um espaço de soluções novo ou sua qualidade for consideravelmente superior que os outros vizinhos.
- *Padrão* – selecionar o melhor vizinho se todas as soluções na vizinhança são tabu.

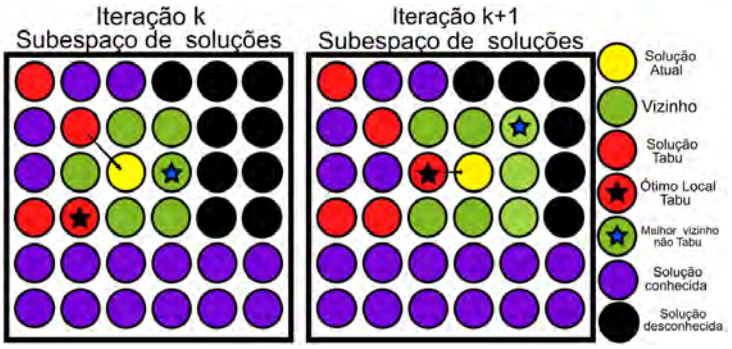
Quando utilizamos uma estratégia pouco restritiva para a memória dinâmica, há uma alta possibilidade de várias delas já serem conhecidas por iterações passadas, assim, reexplorando espaços de soluções previamente encontrados. Porém, muitas restrições podem proibir direções promissoras de serem exploradas ou gerar problemas de performance com muitos acessos ao tabu. Desta maneira, a performance e a eficácia desta meta-heurística está fortemente ligada com as regras sintetizadas para o tabu.

A Figura 4 mostra duas iterações da meta-heurística através de um subespaço de soluções disposto em um plano. A vizinhança é composta pelas oito soluções imediatamente em volta da solução corrente, e a seta representa a direção tomada pela iteração anterior. A figura mostra que, na iteração k, o ótimo local é uma solução tabu. Então, o processo escolhe o melhor vizinho não tabu como a nova solução. Observe que, se o ótimo local não fosse tabu, o método iria verificar uma vizinhança de soluções conhecida, onde iria continuar em direções já exploradas anteriormente. Desta maneira, a iteração $k+1$ começou uma rota em uma direção não explorada do espaço em busca de novos ótimos locais por causa da lista de restrição. Este exemplo mostra que a meta-heurística é capaz de escapar de caminhos que ficam presos em um único subespaço de soluções, caso a vizinhança e a lista tabu sejam devidamente sintetizadas. O Algoritmo 9 apresenta a busca tabu para um problema de otimização de minimização, onde *Atualiza* é o procedimento que realiza o gerenciamento da lista tabu.

1.3.3 Algoritmo Evolucionário

Algoritmo Evolucionário é uma classe abrangente de algoritmos inspirados pelo princípio de seleção natural, tal que as etapas e componentes do algoritmo são nomeados

Figura 4 – Exemplo gráfico da execução de um BT com duas iterações arbitrárias representadas em um subespaço de soluções.



Algoritmo 9 Busca Tabu

função BT(**Solução inicial:** s_0; **Estrutura de vizinhança:** V; **Tamanho máximo do Tabu:** T_{max})

1: $s^*, s \leftarrow s_0$ ▷ Melhor solução s^* e solução corrente s recebem solução inicial
2: $T \leftarrow \emptyset$ ▷ Lista tabu começa vazia
3: **enquanto** critério de parada não satisfeito **faça**
4: $s' \leftarrow \emptyset$ ▷ Tal que $f(s') = \infty$
5: **para** cada $s'' \in V(s)$ **faça** ▷ Para cada vizinho de s
6: **se** $f(s'') < f(s')$ **então** $s' \leftarrow s''$ ▷ Mantém o melhor vizinho.
7: **se** $s' \in T$ **então** ▷ Se s' é uma solução tabu ou o movimento é tabu
8: **se** s' não atende o critério de aspiração **então**
9: $s' \leftarrow \emptyset$ ▷ Descarta s' e ache o próximo melhor
10: **se** s' é melhor que s^* **então** $s^* \leftarrow s'$ ▷ Nova melhor solução.
11: $s \leftarrow s'$ ▷ Solução corrente recebe o melhor vizinho não tabu
12: ATUALIZA(T, T_{max}, s') ▷ Atualiza o tabu de acordo com T_{max} e s'
13: **retorna** s^* ▷ Retorna a melhor solução encontrada

de acordo com termos biológicos. Um algoritmo evolucionário utiliza diversos mecanismos para evoluir um conjunto de soluções através de um processo dividido em fases que realizam um procedimento análogo à evolução biológica. Existem vários algoritmos evolucionários que são aplicados com sucesso em inúmeras aplicações de inteligência artificial e em diversos problemas de otimização combinatória. No contexto de meta-heurísticas, o **Algoritmo Genético** original engloba as estratégias geralmente utilizadas por algoritmos evolucionários. Esta seção descreve de forma resumida os conceitos e estratégias que compõem o algoritmo genético através de descrições em alto nível, seguindo como referência o Capítulo 5 do livro (GLOVER; KOCHENBERGER, 2006).

1.3.3.1 Algoritmo Genético

O **Algoritmo Genético** (AG) é uma meta-heurística composta por vários parâmetros calibráveis, tal que cada fase pode ser implementada com vários operadores distintos. Portanto, é um método difícil de ser calibrado com diversos aspectos práticos a serem considerados para sua implementação. A ideia do algoritmo é simular artificialmente um processo evolutivo com um grupo de soluções, mantendo as melhores a cada iteração para

aplicar uma estratégia que as combina. Desta maneira, as piores soluções são descartadas e um novo conjunto de soluções é formado. Posteriormente, algumas soluções neste novo conjunto são modificadas aleatoriamente utilizando outro operador, com o objetivo de diversificar a busca no espaço de soluções. Este processo é análogo à evolução natural de animais e plantas: somente os mais aptos reproduzem e sobrevivem nas gerações posteriores. O termo *algoritmo genético* foi utilizado pela primeira vez em (HOLLAND, 1975).

Considere um problema de otimização combinatória com um espaço de soluções finito S, tal que a representação de cada solução $s \in S$ é sintetizada por um conjunto X de variáveis discretas e $f : S \to \mathbb{R}$ é a função objetivo associada. Em algoritmos evolucionários, cada solução s é um *indivíduo*, um conjunto de soluções é denominado de *população*, o conjunto final de soluções de cada iteração recebe o nome de *geração* e a respectiva qualidade $f(s)$ da solução é chamada de *aptidão* ou *valor adaptativo*. A representação por componentes do mundo real de cada indivíduo é chamada de *fenótipo*. A representação da solução utilizada no método que codifica o indivíduo é denominada de *genótipo*. Assim, cada indivíduo possui um vetor que o representa chamado de *cromossomo*, que é composto pelas váriaveis discretas $x \in X$. Cada elemento x do cromossomo é denominado *gene*, e os valores que cada variável pode assumir fazem parte de um alfabeto finito Σ. Os elementos de Σ recebem o nome de *alelo*.

Desta maneira, o algoritmo genético é um processo iterativo que realiza a seleção, reprodução, mutação e evolução de uma população com P_{max} indivíduos ranqueados por sua aptidão e codificados através de um genótipo. O processo se estende por várias gerações até que o critério de parada seja satisfeito. Cada geração é composta pela população estabelecida na iteração atual, que é formada por indivíduos cada vez mais aptos a sobreviver. Cada um desses indivíduos é descrito por um cromossomo, que por sua vez é composto de genes que podem assumir qualquer alelo do alfabeto. As fases aplicadas no algoritmo genético são:

- **População Inicial** – Esta fase é aplicada antes do processo iterativo, e deve receber como parâmetro de entrada o número P_{max} de indivíduos que forma uma população. Assim, a saída deve ser uma população com os primeiros indivíduos, que podem ser gerados por um processo completamente aleatório ou através de um processo construtivo que é capaz de sintetizar diferentes soluções. Note que a performance do algoritmo genético está fortemente ligada com a escolha apropriada do parâmetro P_{max}: se uma população é pequena, o processo pode não buscar de forma eficiente no espaço de soluções, porém uma população grande pode gerar problemas de performance pelo custo de memória para armazenar todos os indivíduos e pelo tempo de computação elevado para terminar uma geração.

- **Avaliação** – Recebe uma população como parâmetro de entrada para associar cada indivíduo com sua aptidão através da função objetivo associada ao problema de otimização. Além disso, esta fase deve manter os indivíduos ordenados do melhor para o pior.

- **Seleção** – Recebe uma população e um parâmetro $0 < \alpha < 1$ para sintetizar uma nova população com um número menor de indivíduos. A nova população deve conter os $100\alpha\%$ melhores indivíduos da população original de entrada. Desta forma, somente os melhores indivíduos prosseguem para a fase de reprodução e para a próxima geração. Dependendo do genótipo, uma certa quantidade de indivíduos

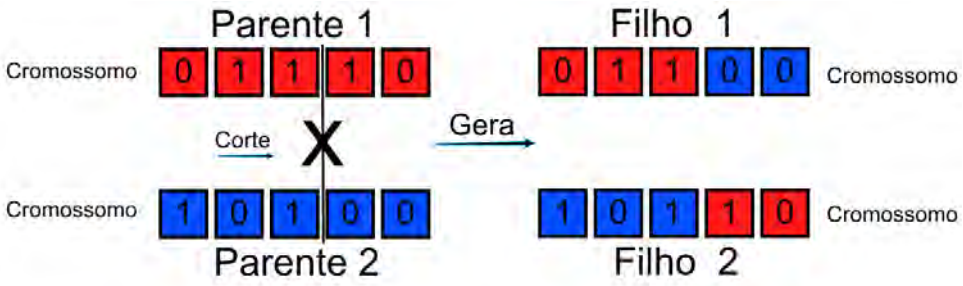
com os mesmos cromossomos pode existir na população. Portanto, a seleção pode desconsiderar indivíduos iguais.

- **Reprodução:** – Recebe como parâmetro de entrada uma população e o número máximo de indivíduos, a fim de preencher a população com novos indivíduos. Os indivíduos devem ser gerados através de um operador $\oplus$. Geralmente, este operador é binário e o seu resultado é um indivíduo: $I_1 \oplus I_2 = I_3$. O indivíduo I_1 é I_2 são denominados de *parentes* ou *pais*, e qualquer solução sintetizada por este operador é chamado de *filho*. Os *parentes* são escolhidos aleatoriamente ou através de uma estratégia.

  Por exemplo, um operador binário conhecido é o *one-point* que pode gerar um ou dois novos indivíduos. O *one-point* consiste em escolher aleatoriamente um ponto de corte i no cromossomo dos parentes. Um dos indivíduos pode ser formado copiando os i primeiros genes do primeiro parente, e o restante de seus genes são copiados do segundo parente. O segundo indivíduo é construído analogamente com a intercalação dos parentes. A Figura 5 apresenta este esquema graficamente através de um genótipo composto por um vetor de *bits*. Uma estratégia bem conhecida para a seleção dos parentes é o processo da *roleta*. Este processo seleciona dois parentes aleatoriamente, tal que a probabilidade é proporcional à qualidade do indivíduo: os melhores indivíduos têm uma maior probabilidade de serem escolhidos para reproduzir. A Figura 6 exemplifica o processo da roleta com seis indivíduos de diferentes aptidões.

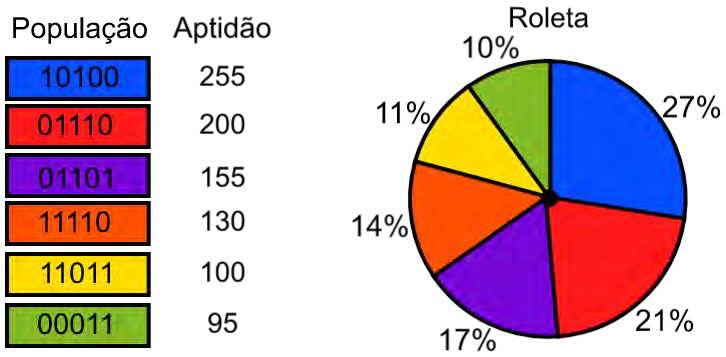
- **Mutação** – A fase de mutação recebe como parâmetro de entrada uma população e um parâmetro $0 \leq \omega < 1$. O objetivo desta fase é diversificar a busca no espaço de soluções introduzindo pequenas modificações em $100\omega\%$ indivíduos da população. O parâmetro ω deve ser um número pequeno para afetar somente alguns poucos indivíduos. A mutação é uma operador $\uplus$ unário que modifica um indivíduo aleatoriamente. Este mecanismo pode ser aplicado de forma simples, por exemplo, modificando um ou mais genes do indivíduo. Um indivíduo modificado por esse operador é chamado de *mutante*.

Figura 5 – Exemplo gráfico do operador de reprodução *one-point*.



Vale a pena enfatizar que os mecanismos descritos nas fases acima podem ser implementados de diversas maneiras, onde a performance e o impacto de cada estratégia depende do problema. Para uma determinada heurística, as fases de reprodução e mutação podem não existir ou o processo de reprodução pode utilizar uma quantidade arbitrária de parentes para gerar vários filhos. Desta maneira, cada fase pode utilizar mais de um parâmetro de entrada calibrável. O Algoritmo 10 mostra um *framework* para o algoritmo

Figura 6 – Exemplo gráfico da estratégia da roleta.



genético através das fases descritas acima, onde os parâmetros de entrada podem variar de acordo com a estratégia aplicada em cada etapa.

Algoritmo 10 Algoritmo Genético

função AG(**Quantidade de indivíduos:** P_{max}; **Taxa de seleção:** α; **Operador de reprodução:** $\oplus$; **Operador de mutação** $\uplus$; **Taxa de mutação:** ω)

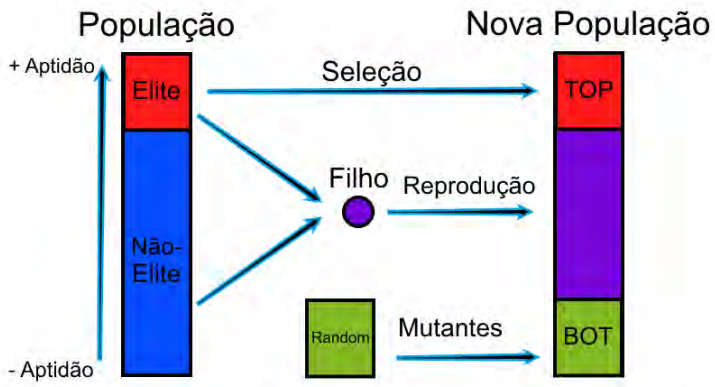
1: $P \leftarrow$ PopulaçãoInicial(P_{max}) ▷ População inicial com P_{max} indivíduos aleatórios
2: Avalia(P) ▷ Mantém os indivíduos ranqueados por aptidão
3: **enquanto** critério de parada não satisfeito **faça**
4: $P \leftarrow$ Seleção(P, α) ▷ Seleciona os $100\alpha\%$ melhores indivíduos
5: Reprodução($P, \oplus, P_{max}$) ▷ Preenche a população P com novos indivíduos
6: Mutação($P, \uplus, \omega$) ▷ Introduz $100\omega\%$ mutantes na população
7: Avalia(P) ▷ Mantém os indivíduos ranqueados por aptidão
8: $I^* \leftarrow$ Melhor indivíduo de P
9: **retorna** I^* ▷ Retorna o melhor indivíduo

1.3.4 Algoritmo Genético de Chaves Aleatórias Viciadas

O **Algoritmo Genético de Chaves Aleatórias Viciadas** ou *Biased Random-Keys Genetic Algorithm (BRKGA)* é uma meta-heurística classificada como um algoritmo evolucionário. Portanto, os conceitos descritos na Seção 1.3.3 também valem são utilizados nesta seção. O método é uma variação do **Algoritmo Genético de Chaves Aleatórias**, também denominado como *Random-Keys Genetic Algorithm (RKGA)*, que é uma variação do algoritmo genético. O RKGA foi originalmente proposto em (BEAN, 1994) para problemas de otimização combinatória de sequenciamento. Uma das principais vantagens do RKGA é que os mecanismos para cada fase são efetuados por métodos definidos, de modo que só um componente do processo depende das características do problema. Ademais, o algoritmo não requer nenhum parâmetro de entrada adicional além dos estabelecidos. Portanto, trata-se de um método difícil de ser calibrado, em uma intensidade menor que a do algoritmo genético. Esta seção descreve o RKGA e o BRKGA seguindo como referência o trabalho (GONÇALVES; RESENDE, 2011), que propôs o BRKGA como uma extensão do RKGA.

A ideia principal é evoluir uma **população codificada**. Os indivíduos são representados por cromossomos, denominado chaves aleatórias ou *random-keys*, na forma de vetores

Figura 7 – Construção de uma geração do RKGA ou BRKGA.



de números reais no intervalo $[0, 1]$. A implementação desse processo requer um componente chamado decodificador. Este componente deve receber um vetor de *random-keys* para fornecer uma representação do indivíduo que pode ser avaliada, ou seja, o decodificador é utilizado para obter a própria solução e a aptidão do indivíduo codificado. O decodificador é o único componente que depende das características do problema. Assim, duas implementações da meta-heurística para diferentes problemas só necessitam de decodificadores diferentes.

A população inicial é composta por n indivíduos aleatórios que devem ser previamente avaliados e ranqueados. Desta maneira, a meta-heurística utiliza um parâmetro $\epsilon < \frac{1}{2}$ positivo, denominado TOP, para dividir a população P da iteração corrente em indivíduos elites e não elites. Quando ordenados do melhor para o pior, os $100\epsilon\%$ melhores indivíduos sintetizam a população elite $P_\epsilon \subset P \quad | \quad |P_\epsilon| < n - |P_\epsilon|$. A cada geração, uma nova população P' é construída com P_{max} indivíduos. Os indivíduos de P_e são copiados sem alterações para P'. Posteriormente, uma fração de mutantes é introduzida em P' para a diversificação da busca. A quantidade de indivíduos na população P_ω de mutantes é determinado por um parâmetro $\omega < \frac{1}{2}$ positivo, chamado BOT, tal que $n - |P_\epsilon| - |P_\omega| > 0$. Os mutantes são gerados da mesma maneira que a população inicial, isto é, indivíduos completamente aleatórios. Assim, a nova população P' é construída com a fração $100\epsilon\%$ elite de P, a fração $100\omega\%$ de mutantes e o restante é gerado através da fase de reprodução. A Figura 7 apresenta graficamente a construção da nova população a cada geração, destacando as populações elite e mutante, dadas pelos parâmetros TOP e BOT, respectivamente.

No RKGA, a fase de reprodução escolhe dois parentes I_1 e I_2 distintos aleatoriamente da população corrente p para construir um filho I_3, a fim de preencher a nova população P' da geração atual. Cada *random-key* de I_3 é uma cópia da respectiva *random-key* de um dos parentes escolhido aleatoriamente. O BRKGA estende este processo através de um parâmetro $\frac{1}{2} < \rho < 1$, tal que o parente I_1 deve ser um indivíduo elite e I_2 um não-elite. Assim, a i-ésima *random-key* de I_3 também é uma cópia da i-ésima *random-key* de I_1 ou I_2, porém a *random-key* do parente elite é escolhida com uma probabilidade $100\rho\%$. Então, a probabilidade da *random-key* do parente elite ser copiada é maior. O processo de reprodução entre dois indivíduos do BRKGA é ilustrado na Figura 8. A figura mostra um parente elite e um não-elite através de seu respectivo cromossomo. O exemplo apresenta um possível filho: a escolha é aleatória de acordo com ρ. Neste exemplo, o gene do parente elite foi escolhido quatro vezes e o não-elite duas vezes, ilustrando a persistência de chaves do indivíduo elite. Observe que o mesmo indivíduo pode ser escolhido várias vezes

Figura 8 – Exemplo de reprodução no BRKGA. A probabilidade de escolha do gene elite é dado pelo parâmetro ρ.

para reproduzir em uma mesma geração. A probabilidade de um mesmo elite reproduzir repetidas vezes é maior do que a de um não-elite no BRKGA. Portanto, as *random-keys* dos elites aparecem com alta frequência distribuídas nos indivíduos da nova população, logo justifica o nome da meta-heurística *biased random-keys* (chaves aleatórias viciadas). O Algoritmo 11 mostra o pseudocódigo do BRKGA, onde o único componente dependente do problema é o próprio decodificador que deve ser utilizado na função *Decodifica_ Avalia*.

Algoritmo 11 Algoritmo Genético de Chaves Aleatórias Viciadas

função BRKGA(**Quantidade de indivíduos:** P_{max}; **TOP:** ϵ,; **BOT:** ω; **Taxa da reprodução:** ρ)

1: $P \leftarrow$ PopulaçãoInicial(P_{max}) ▷ População inicial com n indivíduos aleatórios
2: Decodifica_Avalia(P) ▷ Mantém os indivíduos ranqueados por aptidão através do decodificador
3: **enquanto** critério de parada não satisfeito **faça**
4: $P' \leftarrow$ Seleção(P, ϵ) ▷ Copia os elites para P'
5: Mutação(P', ω) ▷ Introduz $100\omega\%$ indivíduos aleatórios em P'
6: Reprodução(P', ρ, P_{max}) ▷ Preenche P' com novos indivíduos
7: Decodifica_Avalia(P') ▷ Mantém os indivíduos ranqueados por aptidão através do decodificador
8: $P \leftarrow P'$ ▷ Finaliza a geração atual
9: $I^* \leftarrow$ Melhor indivíduo de P
10: **retorna** I^* ▷ Retorna o melhor indivíduo

2 REVISÃO DA LITERATURA

Este capítulo inicia com uma discussão sobre a complexidade computacional dos problemas de empacotamento estudados neste trabalho, evidenciando que os problemas são NP-Difíceis.

Em seguida, discutimos algumas abordagens exatas para os problemas de empacotamento, baseadas em *Branch & Bound* e programação linear inteira mista.

Na sequência, revisamos importantes trabalhos da literatura que apresentam aspectos matemáticos dos problemas ou propõem abordagens alternativas, tais como algoritmos aproximativos e heurísticas. Além disso, revisamos também algumas variantes comuns dos problemas de empacotamento.

2.1 Classe de Complexidade do 2BP e do 3BP

Esta seção mostra que os problemas de empacotamento bidimensional e tridimensional clássicos são da classe NP-Difícil. Os fundamentos teóricos aqui apresentados são baseados no livro clássico *Computers and Intractability: A Guide to the Theory of NP-Completeness* (GAREY; JOHNSON, 1979).

O problema tridimensional de empacotamento, denominado 3BP, pode ser definido formalmente da seguinte maneira: dado um conjunto $I = \{a_1, a_2, \cdots, a_n\}$ de n itens definidos com parâmetros inteiros positivos para sua largura w_i, altura h_i e profundidade d_i, $1 \leq i \leq n$, e um conjunto infinito de caixas $Q = \{C_1, C_2, \cdots\}$ homogêneas com largura W, altura H e profundidade D inteiros positivos, o 3BP consiste em determinar o menor número inteiro positivo λ e uma partição associada $S = C_1 \cup C_2 \cup \cdots \cup C_\lambda = \{a_1, a_2, \cdots, a_n\}$, tal que $C_j \cap C_k = \emptyset$, $1 \leq j < k \leq \lambda$, onde cada item $a_i \in C_j$ é empacotado ortogonalmente sem sobreposição. Na versão clássica, os itens têm orientação fixa. Quando todo item $a_i \in I$ tem profundidade $d_i = D$, o 3BP se reduz ao problema bidimensional de empacotamento, chamado 2BP. Analogamente, o 3BP e o 2BP generalizam o problema conhecido unidimensional de empacotamento 1BP.

Desta maneira, podemos definir o 1BP da mesma forma: dado um conjunto $Q = \{C_1, C_2, \cdots\}$ com infinitas caixas de tamanho W e um conjunto $I = \{a_1, a_2, \cdots, a_n\}$ com n itens caracterizados pelo seu tamanho w_i, o 1BP consiste em minimizar o valor inteiro positivo λ e determinar uma partição associada $S = C_1 \cup C_2 \cup \cdots \cup C_\lambda = \{a_1, a_2, \cdots, a_n\}$, tal que $C_j \cap C_k = \emptyset$, $1 \leq j < k \leq \lambda$.

Para provar que o 2BP e o 3BP são da classe NP-Difícil normalmente utiliza-se a seguinte estratégia:

1. Demonstra-se que o 1BP é NP-Difícil. Como instâncias de 1BP é uma restrição de instâncias do 2BP e 3BP, então estes também são NP-Difíceis.

2. Para provar que o 1BP é NP-Difícil utiliza-se o fato de que se o problema de decisão associado, denominado 1BP-D, é NP-Completo, então ele é NP-Difícil. O 1BP-D

consiste em responder se é possível empacotar sem sobreposição os itens de I com um número λ dado de caixas de tamanho W.

Para provar que 1BP-D é NP-Completo, primeiro devemos demonstrar que o 1BP-D $\in$ NP. Isso é simples, pois o certificado pode ser uma configuração com λ listas de itens e um limite de caixas. Para verificar este certificado basta conferir que: todos os itens foram empacotados apenas uma vez, não existe sobreposição entre os itens de cada conjunto e a quantidade de caixas λ é menor que o limite dado. Este procedimento pode ser feito claramente em um número de passos polinomial em relação ao número de itens.

Em seguida, deve ser mostrada uma redução polinomial de um problema NP-Completo para o 1BP-D. Esta demonstração não foi apresentada no livro citado. O autor apenas sugere utilizar um dos problemas PARTIÇÃO ou 3-PARTIÇÃO. Essa sugestão foi ampliada no trabalho (GAREY; JOHNSON, 1978) usando o problema PARTIÇÃO, que é definido da seguinte forma: dado um conjunto $A = \{a_1, a_2, \cdots, a_n\}$ com n números inteiros positivos, o problema PARTIÇÃO é determinar se existe um subconjunto $S \subset A$, tal que a soma dos elementos em S é igual à soma do elementos em $A - S$.

O processo de redução de PARTIÇÃO para 1BP-D pode ser feito da seguinte maneira: dada uma instância de PARTIÇÃO com $A = \{x_1, x_2, \cdots, x_n\}$ podemos criar uma instância do 1BP-D com $I = \{a_1, a_2, \cdots, a_n\}$ e $\lambda = 2$, tal que $W = \sum_{x_i \in A} x_i$ e $w_i = 2x_i$, $1 \leq i \leq n$. Esta transformação é polinomial em relação à quantidade de itens n. Para transformar uma solução de PARTIÇÃO para uma do 1BP-D basta simplesmente caracterizar os elementos do conjunto S como itens empacotados em uma caixa, e os elementos do complemento de S como os itens empacotados na outra caixa. A transformação inversa é análoga. As transformações entre as soluções podem ser feitas claramente em um número de passos polinomial em relação a n.

2.2 **Métodos exatos**

Esta seção descreve algumas das abordagens exatas encontradas na literatura para os problemas de empacotamento nos casos bidimensional e tridimensional.

Inicialmente, Martello et. al (MARTELLO; PISINGER; VIGO, 2000) propuseram um método exato *Branch & Bound* que resolve o 3BP sem rotação. Os autores solucionam repetidamente o subproblema associado de empacotar um conjunto de itens em uma única caixa, quando possível. O algoritmo seleciona alguns itens da entrada de maneira que cada nó da árvore de *Branch & Bound* é uma solução parcial do 3BP. Porém, Boef et al. (BOEF et al., 2005) demonstraram que esse algoritmo só era capaz de resolver corretamente uma variação do 3BP, chamada de *robot packing*, e não resolve o caso geral de empacotamento ortogonal. Assim, o algoritmo foi modificado em (MARTELLO et al., 2007) para resolver este erro. A correção foi feita em um dos procedimentos internos do algoritmo original que não considerava um caso especial que só é possível na versão clássica do 3BP. Anteriormente, em(MARTELLO; VIGO, 1998) foi proposto um algoritmo exato para a resolução do 2BP sem orientação de itens.

Em (FEKETE; SCHEPERS, 2004a) lida-se com o problema em sua versão multidimensional, modelando soluções através de uma representação com grafos. Os autores definem um grafo que descreve os itens que se sobrepõem através de sua posição relativa dentro da caixa e de sua projeção nos eixos ortogonais. Aliado a heurísticas que selecionam subconjuntos de itens que são possíveis de empacotar na mesma caixa, o resultado é um algoritmo *Branch & Bound* que determina soluções exatas para os problemas de

empacotamento sem rotação. Essa ideia foi ampliada em (FEKETE; SCHEPERS, 2004b; FEKETE; SCHEPERS; VEEN, 2007; FEKETE; VEEN, 2007) com um *framework Branch & Bound* para resolver instâncias grandes do problema clássico de empacotamento com dimensão arbitrária.

Um modelo de programação linear inteira mista (MPLIM) foi proposto por (HIFI et al., 2010; HIFI; NEGRE; WU, 2014). Este modelo pode ser utilizado para determinar a solução ótima do 3BP sem rotação. Porém, achar uma solução ótima para o modelo requer a verificação de um grande número de soluções viáveis. Tais soluções são difíceis de serem encontradas em um tempo prático de computação. Além disso, nos problemas de empacotamento, o espaço de soluções viáveis não é mais convexo após o empacotamento de um item. Um subconjunto V de um espaço vetorial real $\mathcal{R}^n$ é convexo se, e somente se, qualquer combinação linear $u = \alpha v + (1 - \alpha)w$ é um elemento de V, $\forall v, w \in V$ e $0 \leq \alpha \leq 1$. Assim, é difícil construir um modelo simples de programação linear inteira para o 3BP. A seguir descrevemos o MPLIM dos trabalhos citados anteriormente.

Sejam um conjunto $I = \{a_1, a_2, \cdots, a_n\}$ com n itens de orientação fixa definidos pelas dimensões (w_i, h_i, d_i), $0 \leq i \leq n$, para serem empacotados sem sobreposição nas caixas $Q = \{C_1, C_2, \cdots\}$ homogêneas de dimensões (W, H, D). O modelo começa com um limite superior $\hat{\lambda} \leq n$, as variáveis $(x_i, y_i, z_i, \lambda_i)$ definem a localização do item a_i. As coordenadas $x_i, y_i, z_i \geq 0$ denotam a posição de seu canto inferior esquerdo frontal, assumindo que a coordenada $(0, 0, 0)$ é o respectivo canto inferior esquerdo frontal da caixa, e a variável $\lambda_i \geq 1$ é o número que diz o rótulo da caixa em que está empacotado. O objetivo do modelo é minimizar o maior rótulo $\lambda = \text{máx}(\{\lambda_1, \lambda_2, \cdots, \lambda_n\})$ usado. Para escrever o modelo de programação linear inteira, as duas seguintes propriedades devem ser respeitadas para representar uma configuração válida de empacotamento:

- **Propriedade 1 (P1):** Todos os itens devem estar totalmente contido nas caixas.
- **Propriedade 2 (P2):** Não há sobreposição de itens.

Considere o item a com dimensões (w, h, d). Podemos garantir que as faces do item não ultrapassam os limites da caixa com largura W, altura H e profundidade D se, e somente se, for empacotado em alguma posição (x, y, z), tal que $0 \leq x \leq W - w$, $0 \leq y \leq H - h$ e $0 \leq z \leq D - d$. Dessa forma, a propriedade P1 pode ser modelada da seguinte maneira:

$$0 \leq x_i \leq W - w_i \tag{1}$$
$$0 \leq y_i \leq H - h_i \tag{2}$$
$$0 \leq d_i \leq D - d_i \tag{3}$$

A propriedade P2 é satisfeita se as duas seguintes condições são atendidas para cada par de itens: ambos estão na mesma caixa sem sobreposição ou estão empacotados em diferentes caixas. Para checar se os itens a_i e a_j, $i \neq j$, estão empacotados em caixas diferentes, uma variável decisão é usada com a seguinte regra: c_{ij} vale 1 se, e somente se, $\lambda_i < \lambda_j$. Assim, se os itens a_i e a_j estão em caixas diferentes, então:

$$c_{ij} + c_{ji} = 1$$

Quando os itens a_i e a_j estão na mesma caixa, então as seguintes variáveis de decisão são usadas: l_{ij} valendo 1 se, e somente se, a_i está na esquerda de a_j; u_{ij} sendo 1 se, e

somente se, a_i está encima de a_j e b_{ij} igual à 1 se, e somente se, a_i está na frente de a_j. Assim, para garantir que não há sobreposição, devemos certificar que:

$$\begin{aligned} l_{ij} = 1 &\implies x_i + w_i \leq x_j \\ u_{ij} = 1 &\implies y_i + h_i \leq y_j \\ b_{ij} = 1 &\implies z_i + d_i \leq z_j \end{aligned}$$

Desta maneira, podemos modelar P2 da seguinte forma:

$$l_{ij} + l_{ji} + u_{ij} + u_{ji} + b_{ij} + b_{ji} + c_{ij} + c_{ji} = 1 \quad (4)$$
$$x_i + w_i - x_j + W(l_{ij} - c_{ij} - c_{ji}) \leq W \quad (5)$$
$$y_i + h_i - y_j + H(u_{ij} - c_{ij} - c_{ji}) \leq H \quad (6)$$
$$z_i + d_i - z_j + D(b_{ij} - c_{ij} - c_{ji}) \leq D \quad (7)$$

Nas restrições (5), (6) e (7), quando a_i e a_j estão em caixas diferentes, então $c_{ij} = 1$ ou $c_{ji} = 1$. Isto garante que as inequações serão verdadeiras e nenhuma sobreposição precisa ser considerada. Caso contrário, uma das variáveis $l_{ij}, l_{ji}, u_{ij}, u_{ji}, b_{ij}$ ou b_{ji} deve valer 1. Consequentemente, essas restrições vão garantir que os limites do respectivo item não ultrapasse a posição do outro em qualquer dimensão de maneira análoga à P1. Finalmente, o modelo de programação linear inteira mista para o 3BP, denominado MPLIM, pode ser formulado da seguinte maneira:

Minimizar λ (MPLIM)

Sujeito a

$0 \leq x_i \leq W - w_i,$ $1 \leq i \leq n$ (1)

$0 \leq y_i \leq H - h_i,$ $1 \leq i \leq n$ (2)

$0 \leq z_i \leq D - d_i,$ $1 \leq i \leq n$ (3)

$l_{ij} + l_{ji} + u_{ij} + u_{ji} + b_{ij} + b_{ji} + c_{ij} + c_{ji} = 1,$ $1 \leq i < j \leq n$ (4)

$x_i + w_i - x_j + W(l_{ij} - c_{ij} - c_{ji}) \leq W,$ $1 \leq i \neq j \leq n$ (5)

$y_i + h_i - y_j + H(u_{ij} - c_{ij} - c_{ji}) \leq H,$ $1 \leq i \neq j \leq n$ (6)

$z_i + d_i - z_j + D(b_{ij} - c_{ij} - c_{ji}) \leq D,$ $1 \leq i \neq j \leq n$ (7)

$(\hat{\lambda} - 1)(l_{ij} + l_{ji} + u_{ij} + u_{ji} + b_{ij} + b_{ji}) + \lambda_i - \lambda_j + \hat{\lambda} c_{ij} \leq \hat{\lambda} - 1,$ $1 \leq i \neq j \leq n$ (8)

$0 \leq \lambda_i \leq \lambda \leq \hat{\lambda},$ $1 \leq i \leq n$ (9)

$l_{ij}, u_{ij}, b_{ij}, c_{ij} \in \{0, 1\},$ $1 \leq i \neq j \leq n$

tal que $l_{ij} = 1$ se o item a_i está na esquerda do item a_j, $u_{ij} = 1$ se o item a_i está embaixo do item a_j, $b_{ij} = 1$ se o item a_i está na frente do item a_j e $c_{ij} = 1$ se $\lambda_i < \lambda_j$. A condição de que todos os itens devem estar alocados dentro dos limites da caixa é imposta pelas restrições (1, 2, 3). A condição de que dois itens não se sobrepõem é imposta pelas restrições (4, 5, 6, 7). A restrição (8) garante que quando $c_{ij} = 1$ ou $c_{ji} = 1$, então os itens a_i e a_j estão localizados em diferentes caixas.Desse modo, quando algum dos parâmetros $l_{ij}, l_{ji}, u_{ij}, u_{ji}, b_{ij}, b_{ji}$ é igual a 1, então os itens a_i e a_j estão necessariamente na mesma caixa.

O parâmetro $\hat{\lambda}$ é um limite superior válido para λ. O valor pode ser obtido com uma heurística rápida, por exemplo, o *First Fit* (Primeiro Encaixe). Esta heurística encaixa

cada item a_i sequencialmente, na ordem em que eles são dados, na primeira caixa que é possível empacotar. Portanto, se trata de um algoritmo simples que utiliza no máximo n caixas. O Algoritmo 12 detalha o procedimento, onde a função *Empacota* segue uma determinada estratégia para encaixar o item atual na caixa.

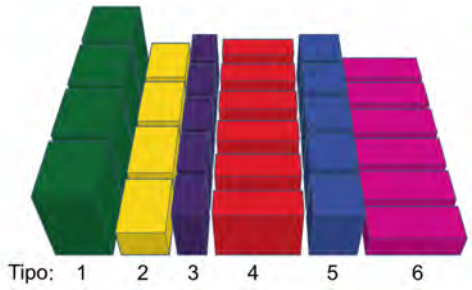
Por exemplo, podemos fixar um dos cantos da caixa como um ponto de referência, assim, os itens devem ser empacotados o mais próximo possível deste canto de referência. Ou seja, a estratégia de empacotamento é alocar o item de maneira a minimizar a distância entre o canto de referência e o respectivo canto do item. A Figura 9 ilustra a execução do *First Fit* com um exemplo do 3BP sem rotação que utiliza 30 itens dispostos em 6 tipos, denotados por uma cor cada. O *First Fit* foi executado com o canto de referência sendo o inferior esquerdo frontal. A ordem dos itens empacotados segue o seu tipo: um item de tipo 1 foi empacotado primeiro, em seguida, um de tipo 2 e assim por diante.

Algoritmo 12 First Fit

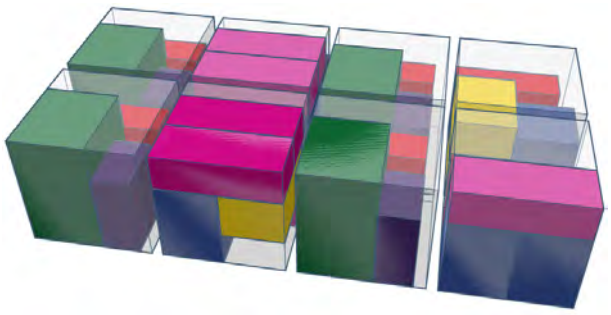
função FirstFit(**Lista de Itens:** I; **Lista de caixas:** Q)

1: $\hat{\lambda} \leftarrow 0$ ▷ Quantidade de caixas utilizadas
2: **para** $a_i \in I$ **faça** ▷ Para cada item a_i, $1 \leq i \leq n$, no conjunto I
3: $j \leftarrow 1$
4: **enquanto** $j \leq \hat{\lambda}$ e a_i não está empacotado **faça** ▷ Para cada caixa aberta
5: **se** a_i cabe em $C_j \in Q$ **então** EMPACOTA(a_i, C_j) ▷ Empacota a_i em C_j se couber
6: **se** a_i não foi empacotado **então** ▷ a_i não cabe nas caixas abertas
7: $\hat{\lambda} \leftarrow \hat{\lambda} + 1$ ▷ Abre uma nova caixa
8: EMPACOTA$(a_i, C_{\hat{\lambda}})$ ▷ Empacota na nova caixa
9: **retorna** $\hat{\lambda}$ ▷ Limite superior para o MPLIM

Figura 9 – Exemplo de uma instância do 3BP sem rotação solucionada através do *First Fit* apresentado no Algoritmo 12.



(a) Instância com $n = 30$ itens de 6 tipos distintos.



(b) Solução com 6 caixas do 3BP sem rotação com *First Fit*.

2.3 Abordagens alternativas

As abordagens alternativas encontradas na literatura são geralmente algoritmos aproximativos, heurísticas ou algoritmos *online*. Esta seção revisa algumas dessas abordagens em conjunto com trabalhos que estabelecem um limite inferior ou classificações para os problemas de empacotamento.

Os fundamentos matemáticos dos problemas de empacotamento unidimensional (1BP) começaram a ser estudados por M. R. Garey et al. (GAREY; GRAHAM; ULLMAN, 1972) por volta de 1970. O trabalho citado introduz as importantes heurísticas *First Fit*, *Best Fit*, *First Fit Decreasing* e *Best Fit Decreasing*, que são algoritmos de complexidade $O(n \log n)$, onde n é a quantidade de itens. Alguns meses depois, o trabalho (JOHNSON, 1972) estende os resultados do trabalho anterior, e estuda uma classe de algoritmos aproximativos. Os autores desses dois trabalhos colaboraram com o pesquisador A. Demers para publicar uma a primeira análise completa de algoritmos aproximativos para o 1BP (JOHNSON et al., 1974). Este trabalho pioneiro abriu espaço para a área de pesquisa: o modelo introduzido foi utilizado por uma grande variedade de diferentes problemas práticos, como as variantes dos problemas de corte, carregamento de caminhões e alocação de memória. Simultaneamente, Johnson D. S. completou sua tese de PhD, onde detalhou de maneira compreensiva as demonstrações abreviadas no último trabalho citado (JOHNSON, 1973).

Os problemas de empacotamento, em conjunto com os problemas fortemente relacionados de partição, são de extrema importância para a teoria da complexidade computacional, combinatória e na análise de algoritmos aproximativos, como destacado pelo trabalho (HOCHBA, 1997). A grande maioria da pesquisa referente ao problema de empacotamento é focada na análise combinatória de algoritmos, estudando limites para o pior caso. No começo de 1980, a pesquisa sobre a análise da complexidade dos problemas de empacotamento começou a crescer. Além disso, vários pesquisadores estudaram resultados sobre a taxa de aproximação do 1BP como Edward Coffman (COFFMAN JR.; GAREY M; S., 1984). Em (SIMCHI-LEVI et al., 1994), David Simchi-Levi provou que a taxa de aproximação do *First Fit Decreasing* e *Best Fit Decreasing* é $\frac{3}{2}$. Também demonstrou que a taxa de aproximação do 1BP não pode ser menor que $\frac{3}{2}$, a não ser que $P = NP$. Este trabalho foi um marco importante para o estudo dos problemas de empacotamento.

Mais resultados e algoritmos foram introduzidos, tais quais precisaram ser sumarizados em *surveys*. Os primeiros grandes *surveys* publicados de algoritmos para os problemas de empacotamento foram (GAREY; JOHNSON, 1981) e (COFFMAN JR.; GAREY M; S., 1984). O último trabalho citado foi atualizado em (COFFMAN JR.; GAREY M; S., 1996). Além de sumarizar os algoritmos tradicionais, este *survey* também descreve as abordagens *online* até a época, que são algoritmos capazes de lidar com valores de entrada que são passados sequencialmente pedaço por pedaço, tal que nem todos os parâmetros estão disponíveis desde o começo. Posteriormente, o trabalho foi novamente atualizado em (COFFMAN JR.; CSIRIK; LEUNG, 2006a, 2006b), onde a versão mais recente é (COFFMAN JR. et al., 2013). Nesta versão, o trabalho aborda a literatura recente sobre o 1BP, concentrado na análise combinatória, incluindo as variantes mais comuns do problema. Quando se trata da versão multidimensional do problema (nBP), o *survey* (CHRISTENSEN et al., 2017) detalha as abordagens mais recentes de algoritmos aproximativos e *online* para diversas variantes do problema.

Os problemas de corte também estão relacionados com os de empacotamento. Com o crescimento do número de publicações referentes nesta área, Dyckhoff (DYCKHOFF, 1990) apresentou uma tipologia para os problemas de empacotamento e corte. Este trabalho é uma referência para a organização e categorização de toda literatura. De maneira similar, (COFFMAN JR.; CSIRIK, 2007) também introduz uma classificação para os problemas de empacotamento em conjunto com uma revisão bibliográfica, ressaltando as abordagens e as variantes mais comuns do problema. Porém, a tipologia mais usada atualmente é a de Wäscher et al. (WÄSCHER; HAUSSNER; SCHUMANN, 2007), que foi

utilizada no Capítulo desta dissertação. Este último trabalho citado segue parcialmente a ideia original de Dyckhoff, introduzindo um novo critério de categorização. O trabalho provê um sistema de nomes para cada categoria de problemas.

O limite inferior é importante para as heurísticas que resolvem os problemas de empacotamento, pois, geralmente, não é conhecido o ótimo global das instâncias testadas. Tais limites são baseados no tamanho total dos itens que devem ser empacotados. Os trabalhos (MARTELLO; VIGO, 1998; MARTELLO; PISINGER; VIGO, 2000) estabelecem um limite inferior para o 3BP e 2BP através de uma extensão de um modelo para o limite inferior do 1BP proposto em (MARTELLO; TOTH, 1990). Neste trabalho, os autores analisam o limite inferior do 1BP. (FEKETE; SCHEPERS, 2001) provêm um novo limite inferior que domina o resultado demonstrado por (MARTELLO; PISINGER; VIGO, 2000) para o 3BP. (BOSCHETTI; MINGOZZI, 2003a, 2003b; BOSCHETTI, 2004) combinam os resultados dos trabalhos anteriores com heurísticas para melhorar os limites propostos. Recentemente, os autores de (LIAO; HSU, 2013) atualizaram os resultados sobre o limite inferior do 3BP sem rotação, sendo este o melhor limite encontrado na literatura.

Poucos trabalhos estudam o caso que permite rotação. O trabalho (DELL'AMICO; MARTELLO; VIGO, 2002) apresentou o primeiro limite inferior para o 2BP com rotação. Além disso, os autores de (BOSCHETTI; MINGOZZI, 2003a, 2003b) também estabelecem um limite inferior para o caso bidimensional com rotação, enquanto (BOSCHETTI, 2004) estende os conceitos para o 3BP com rotação. Finalmente, (LIAO; HSU, 2013) apresentam um limite inferior que domina o limite do trabalho anterior para o 3BP com rotação.

2.3.1 Abordagens com Meta-Heurísticas

A seguir descrevemos várias heurísticas propostas para os problemas de empacotamento. Alguns desses trabalhos são usados no Capítulo 4 para comparação com os métodos propostos nesta dissertação.

O método exato proposto por (MARTELLO; PISINGER; VIGO, 2000; MARTELLO et al., 2007) pode ser rodado com um limite de tempo ou de nós para executar uma heurística *Branch & Bound Truncada*. (LODI; MARTELLO; VIGO, 1999, 2002) desenvolveram algoritmos Busca Tabu baseados em procedimentos construtivos para o 2BP e 3BP sem rotação. Em (LODI; MARTELLO; VIGO, 2004) é proposta uma heurística unificada Busca Tabu que resolve o nBP. (FAROE; PISINGER; ZACHARIASEN, 2003) propõe uma Busca Local Guiada para o 3BP com orientação fixa que também resolve o 2BP. O algoritmo se baseia em soluções iterativas do problema de satisfação de restrições. Em (CRAINIC; PERBOLI; TADEI, 2009) foi desenvolvida uma heurística Busca Tabu de dois níveis utilizando a representação de (FEKETE; SCHEPERS, 2004a), onde o primeiro nível foca em reduzir o número de caixas utilizadas e o segundo otimiza o empacotamento dos itens.

Uma heurística híbrida GRASP-VND é proposta em (PARREÑO et al., 2010a). A fase de construção utiliza um valor máximo para as caixas, onde nem todos os itens precisam ser empacotados. O algoritmo utiliza uma estrutura VND com diversas estratégias de reempacotamento para obter uma solução onde todos os itens são empacotados. Desta maneira, na fase de construção da próxima iteração GRASP, o número máximo de caixas é subtraído por um. O espaço vazio das caixas abertas é modelado através de espaços maximais. O processo construtivo foi originalmente projetado para resolver o problema de carregamento de contêiner proposto em (PARREÑO et al., 2008). (CRAINIC; PERBOLI; TADEI, 2012) combinam um algoritmo guloso adaptativo com mecanismos de aprendizado e evolução para sintetizar um *framework* para os problemas de empacota-

mento tridimensional com caixas homogêneas ou heterogêneas. Este método pode ser aplicado em diversas variações do problema de empacotamento com itens orientados ou não.

Um dos primeiros algoritmos genéticos para o problema de empacotamento foi proposto em (FALKENAUER, 1996). Falkenauer detalha um algoritmo genético que foi modificado para resolver problemas de partição ou agrupamento. Dessa forma, ele utiliza esta heurística para resolver o 1BP. Com uma abordagem mais recente, o trabalho (GONÇALVES; RESENDE, 2013) introduz um algoritmo genético de chaves aleatórias viciadas para o 3BP e 2BP considerando itens que podem ser rotacionados ou não, utilizando a mesma estratégia de espaços maximais de (PARREÑO et al., 2010a) para efetuar o empacotamento. Este algoritmo genético é uma meta-heurística mais recente utilizada com sucesso em diversos problemas de otimização combinatória. Atualmente este é o trabalho que apresenta os melhores resultados em termos de qualidade para o 3BP e o 2BP. Os autores propõem uma nova heurística gulosa determinística para o empacotamento sequencial dos itens. Além disso, eles utilizam uma função objetivo alternativa que melhora consideravelmente a qualidade das soluções. A nova função objetivo acelera o processo evolutivo da metaheurística, adicionando ao número de caixas um coeficiente no intervalo $[0, 1[$ baseado na caixa utilizada com a menor carga. (MACK; BORTFELDT, 2012) apresenta uma heurística para o 3BP e o 2BP com rotação restringido a empacotamentos de corte guilhotina. Recentemente, (HIFI; NEGRE; WU, 2014) utiliza uma estratégia de programação linear inteira sem o auxílio de meta-heurísticas para resolver o 3BP sem rotação. O resultado é uma heurística separada em dois procedimentos. O primeiro método consiste de duas fases combinadas de aproximação e seleção e o segundo é uma extensão do primeiro com uma fase de otimização. Ambos são combinados para formar a heurística que determina a solução final.

Para a resolução do 2BP, (BOSCHETTI; MINGOZZI, 2003a, 2003b) propõe uma heurística construtiva que determina uma pontuação para cada item, considerando o empacotamento dos itens de acordo com sua pontuação em ordem descrescente. Deste modo, cada iteração atualiza os valores dos itens de acordo com sua estratégia específica de ranqueamento até que o critério de parada seja atingido. (MONACI; TOTH, 2006) apresenta uma heurística baseada em cobertura de conjuntos, onde a primeira fase gera uma coluna de itens para o empacotamento, assumindo que não há rotação. O método utiliza várias heurísticas, incluindo (BOSCHETTI; MINGOZZI, 2003b) e (MARTELLO; VIGO, 1998), limitadas por tempo de execução. Na segunda fase, as colunas são utilizadas com estratégias que resolvem o problema de cobertura de conjunto para fornecer uma solução final para o 2BP sem rotação. Abordagens do 2BP com a rotação de itens em 90^o são sumarizadas no trabalho (LODI; MARTELLO; MONACI, 2002). Este trabalho é um *survey* que sumariza algoritmos aproximativos e heurísticas para a resolução do 2BP e do problema de carregamento de contêiner, o qual tem por objetivo carregar uma única caixa. No trabalho (EPSTEIN; STEE, 2007), os autores detalham heurísticas de complexidade linear que provêm boas soluções para o 2BP clássico e a versão que empacota os itens por camadas. Além disso, o trabalho também descreve heurísticas para o empacotamento de itens de três ou mais dimensões, discutindo como abordar algumas variantes comuns nestas dimensões.

Podemos encontrar vários trabalhos que estudam o variantes do 3BP e o 2BP. Um algoritmo guloso para o 3BP com restrições de estabilidade é proposto por (SILVA; SOMA; MACULAN, 2003). As restrições impõem que os itens devem ser empacotados de uma maneira que as condições físicas de estabilidade, como a gravidade e o equilíbrio, sejam

respeitadas. A abordagem proposta é realizada em duas fases, onde a primeira efetua o empacotamento dos itens e a segunda avalia as condições de estabilidade. Em (ARMENTANO; ARAÚJO, 2007), o problema de empacotamento tridimensional em faixa foi resolvido através de uma adaptação de uma heurística de múltiplos inícios. Esta variante do 3BP consiste em empacotar os itens com um arranjo que tenha comprimento mínimo, onde restrições de estabilidade são consideradas. O algoritmo proposto constrói padrões de carregamento com cuboides de tamanho variado para maximizar o espaço ocupado. O trabalho (BRITO; MACULAN; BRITO, 2015) explica que a radiocirurgia é uma técnica terapêutica aplicada ao tratamento de lesões associadas à tumores cerebrais. Feixes esféricos de radiação com raio variado são aplicados sobre a lesão através de um equipamento especializado. Este equipamento, por sua vez, deve ser configurado no que diz respeito ao posicionamento, quantidade e tamanho dos feixes aplicados à lesão. Esta configuração é equivalente a resolver uma variante do 3BP que empacota esferas em um sólido, de maneira a ocupar o maior espaço possível. Os autores propuseram um algoritmo GRASP para resolver o problema. No trabalho (CÔTÉ; GENDREAU; POTVIN, 2014), um método exato foi proposto para o caso bidimensional de uma variante conhecida dos problemas de empacotamento, a qual aplica restrições de carga e roteamento. Este tipo de variante é uma mistura dos problemas de empacotamento sem rotação com o problema de roteamento de veículo. A abordagem é um algoritmo *branch & cut* que corta cada item em vários pedaços unitários. Desta maneira, eles determinam se estes pedaços podem ser alocados em uma certa quantidade de caixas homogêneas.

3 HEURÍSTICAS PARA O 2BP E 3BP

Os problemas de empacotamento tridimensional (3BP) e bidimensional (2BP) consistem em empacotar um dado conjunto de n itens em caixas homogêneas e são da classe NP-Difícil. Os algoritmos exatos conhecidos são impraticáveis de serem executados para instâncias grandes, devido ao longo tempo de computação. Desta maneira, heurísticas e meta-heurísticas são alternativas para resolver o problema, na prática, com soluções de boa qualidade. Este capítulo descreve os algoritmos propostos para a resolução do 3BP e 2BP em conjunto com os componentes utilizados. Os métodos descritos utilizam estratégias independentes da dimensão, podendo ser aplicados tanto para o caso tridimensional quanto para o bidimensional. Os mecanismos adicionais para os casos que permitem a rotação dos itens também são detalhados. Além disso, o capítulo também apresenta a análise de complexidade de cada algoritmo, a fim de descrever o comportamento prático e o processo de implementação de cada um.

3.1 Representação da Solução

No caso tridimensional, a representação computacional de um item a_i, $1 \leq i \leq n$, é dada pelo vetor $(h_i, w_i, d_i, x_i, y_i, z_i, \lambda_i)$ de variáveis inteiras positivas. Para cada ítem, h_i é a sua altura, w_i sua largura, d_i sua profundidade, (x_i, y_i, z_i) as coordenadas cartesianas que representas sua localização dentro de sua caixa e λ_i o número da caixa em que está empacotado. A posição do item depende de um ponto especial da caixa chamado de **canto de referência**. A origem $(0, 0, 0)$ é o canto de referência, assim, a localização (x_i, y_i, z_i) do item se refere à posição de seu respectivo canto. Nesta trabalho, o canto de referência utilizado em todos os exemplos e implementações é o inferior esquerdo frontal. Pode-se ver que as coordenadas (x_i, y_i, z_i) definem a localização do canto inferior esquerdo frontal do item. Quando um item não está empacotado, os valores de $(x_i, y_i, z_i, \lambda_i)$ permanecem iguais a zero. Para o caso bidimensional, a representação computacional de um item é definida de forma análoga através do vetor $(h_i, w_i, x_i, y_i, \lambda_i)$.

A representação da solução computacional do 3BP e do 2BP pode ser definida através de um vetor $S = \{C_1, C_2, \cdots, C_\lambda\}$ com λ caixas e as variáveis (H, W, D) inteiras positivas para a altura, largura e profundidade das caixas. Cada caixa $C_j \in S$, $1 \leq \lambda$ é um vetor com itens. Desta maneira, podemos construir uma solução física para os problemas de empacotamento através das configurações de itens presentes nas caixas, e a solução final do problema é o próprio número de caixas utilizadas λ em conjunto com os valores λ_i que ditam em qual caixa o item a_i está presente.

3.2 Função Objetivo

O problema de empacotamento consiste em minimizar o número de caixas para alocar um conjunto de itens. A função objetivo trivial (FOT) é o próprio número de caixas. Porém, diversas soluções utilizam a mesma quantidade de caixas. Para a grande maioria

das instâncias, há muitas soluções com configurações de itens que compartilham o mesmo número de caixas. Portanto, o FOT pode não avaliar bem o potencial de melhora de uma solução.

Para resolver este problema, em (GONÇALVES; RESENDE, 2013) é proposta uma função objetivo alternativa (FOA), a fim de diferenciar soluções que utilizam o mesmo número de caixas. Para isso, avaliamos a quantidade de carga de uma caixa computando o espaço ocupado pelos itens empacotados nela dividido por seu espaço total. Assim, o FOA mede o potencial de melhora de uma solução através de uma fração no intervalo $]0, 1[$. Dadas duas soluções que compartilham o mesmo número de caixas, a que apresenta a caixa com menor carga será a que tem mais potencial para melhora. A Equação 3.1 apresenta o FOA que avalia uma solução s qualquer, onde C é a capacidade da caixa ($W \times H \times D$ para o 3BP ou $W \times H$ para o 2BP) e $MC(s)$ é a carga da caixa que apresenta a menor carga da solução.

$$FOA(s) = FOT(s) + \frac{MC(s)}{C} \tag{3.1}$$

Por exemplo, considere duas soluções s_1 e s_2 distintas que utilizam 3 caixas cada, ou seja, $FOT(s_1) = FOT(s_2) = 3$. As caixas de s_1 e s_2 apresentam as respectivas cargas: $\{80\%, 50\%, 30\%\}$ e $\{70\%, 40\%, 50\%\}$. Como a caixa de menor carga de s_1 tem 30%, então $\frac{MC(s_1)}{C} = 0{,}3$. Para s_2 temos que: $\frac{MC(s_2)}{C} = 0{,}4$. Então, $FOA(s_1) = 3{,}3$ e $FOA(s_2) = 3{,}4$. Portanto, s_1 tem um maior potencial de melhora comparado a s_2.

3.3 **Espaço Maximal**

Esta seção apresenta um componente fundamental para o funcionamento do algoritmo de empacotamento da Seção 3.4, que, por sua vez, é utilizado na heurística construtiva, presente na Seção 3.5, que efetua o processo de empacotamento de uma sequência de itens. O componente descrito nesta seção é chamado de **Espaço maximal** (EM), e seu objetivo é modelar o espaço vazio de uma caixa. Isto é feito estruturando as possíveis localizações para a alocação dos próximos itens em uma caxa aberta, isto é, uma caixa com pelo menos um item já empacotado. Esta modelagem também é utilizada nos trabalhos (PARREÑO et al., 2010a) e (GONÇALVES; RESENDE, 2013).

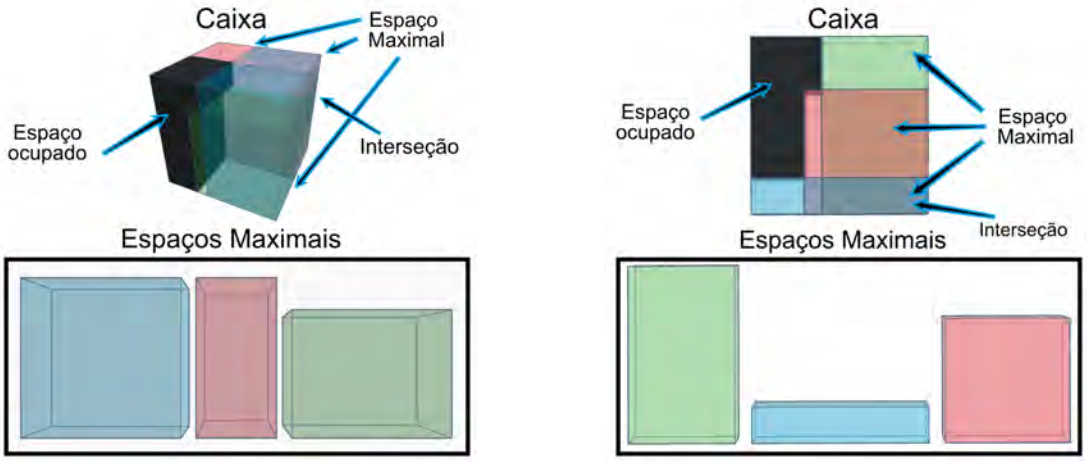
Um Espaço maximal é o maior paralelepípedo ou retângulo que pode ser contido em um espaço ou área vazia de uma caixa. Ou seja, o EM é um espaço vazio dentro de uma caixa, tal que este espaço é o máximo comportado pelas restrições físicas impostas perante os itens já empacotados na caixa e a sua própria dimensão. Este espaço recebe o nome de maximal por englobar o maior espaço vazio dentro de uma caixa. Os EM são sempre os candidatos avaliados para o empacotamento de um item. Esta estratégia modela de forma simples o espaço vazio em uma caixa, tornando o processo de empacotamento uma tarefa que considera os subespaços possíveis para a alocação de um item sem sobreposição.

A representação computacional de um EM tridimensional é similar à de um item, e deve estar associado diretamente com sua respectiva caixa. A representação computacional da caixa, além de guardar a configuração dos itens empacotados, também deve armazenar uma estrutura que mantém seus EM. De acordo com a Seção 3.1, a origem $(0, 0, 0)$ é o canto de referência inferior esquerdo frontal. A representação de um EM consiste de uma coordenada cartesiana (x, y, z) que identifica a localização de seu canto inferior esquerdo frontal, três inteiros (w, h, d) para armazenar suas dimensões e um ponto especial chamado de **canto de inserção** (CI). Analogamente, a estrutura bidimensional é definida através

dos parâmetros (x, y), (w, h) e o seu CI. O canto de inserção é um ponto especial que identifica a localização na qual o item será empacotado caso o EM seja escolhido e não deve ser confundido com o canto de referência. O CI é determinado no momento em que o EM é criado. O critério de escolha é detalhado na Seção 3.5, pois depende da heurística construtiva de empacotamento. Quando um EM é escolhido para conter o item, o respectivo canto do item é empacotado no CI. Por exemplo, se o CI de um EM é o canto superior direito posterior, então o item será alocado dentro do EM com seu respectivo canto superior direito posterior exatamente no CI.

As Figuras 10a e 10b apresentam dois exemplos de configurações de caixas para os respectivos casos tridimensional e bidimensional. As figuras mostram duas caixas com itens empacotados, onde a área escura mostra o espaço ocupado pelas caixas. O espaço vazio é modelado por cada EM, que estão ilustrados nas respectivas figuras. Cada heurística construtiva deve usar a estrutura de EM associada à caixa para avaliar qual deles deve receber o item que está sendo empacotado. Observe que é possível ter interseções entre EM. Isto é necessário para o funcionamento do algoritmo de empacotamento descrito na Seção 3.4.

Figura 10 – Exemplos de EM modelando o espaço vazio em duas caixas.



(a) Espaços maximais 3D.

(b) Espaços maximais 2D.

3.4 **Algoritmo de Empacotamento**

Esta seção descreve o algoritmo de empacotamento que tem como parâmetro de entrada o item que será alocado, a caixa e o EM que receberão este item. Este algoritmo é utilizado pela heurística construtiva apresentada na Seção 3.5, que estabelece qual EM deve conter o item de entrada. A heurística construtiva orienta o item e seleciona uma caixa com seu respectivo EM baseado em certos critérios também introduzidos na Seção 3.5.

O algoritmo de empacotamento deve formar novos EM quando um item é empacotado em uma caixa aberta. Como descrito na Seção 3.3, o item será alocado no CI do EM de entrada. As dimensões do item são menores ou iguais que as dimensões do EM escolhido. Este EM deixará de existir e novos devem ser sintetizados. O procedimento que constrói os novos EM é chamado de *Corte*, recebendo como parâmetro de entrada o EM e o próprio item empacotado. Este procedimento realiza cortes no EM baseado nas dimensões do item,

como ilustrado nos exemplos das Figuras 11 e 12 para os respectivos casos tridimensional e bidimensional. As figuras ilustram o EM selecionado como um paralelepípedo cinza, apresentando posteriormente os novos EM que são construídos, tal que o espaço escuro é o item empacotado e cada paralelepípedo com cor, um EM distinto. Como o item é sempre empacotado em um dos cantos, até 2 novas áreas são formadas no caso bidimensional e no máximo 3 espaços para o caso tridimensional. Existe a possibilidade de um novo EM ser inválido após o corte. Isto acontece quando alguma dimensão do item é igual a uma dimensão do EM cortado: pelo menos um EM formado teria seu espaço igual a 0. Também há a possibilidade de um novo EM estar totalmente contido em um EM já existente. O CI é determinado na construção do EM, porém, os possíveis cantos candidatos e o seu critério de escolha dependem do critério de avaliação de um EM feito pela heurística construtiva. Tais critérios são introduzidos nas Seções 3.5.1 e 3.5.2.

Após o procedimento de corte, um novo procedimento deve ser aplicado durante o empacotamento do item, para **atualizar** o restante dos EM existentes na caixa aberta. A existência de interseções entre os EM implica que um item pode ser empacotado no mesmo espaço ocupado por outros EM. O procedimento de atualização deve corrigir esse problema através de cortes efetuados em cada EM afetado. Quando este EM é identificado, o processo deve agir de maneira similar ao método de corte: remover o EM da caixa associada e realizar cortes de acordo com as dimensões do item. Porém, este novo processo de atualização pode cortar o EM em até 4 ou 6 novos para os casos bidimensional e tridimensional respectivamente.

Além dos EM com área zero e totalmente contidos em outros EM, podemos filtrar mais alguns durante o procedimento de alocação. Se a heurística construtiva sequencial mantiver cada menor dimensão existente no subconjunto de itens a serem empacotados, atualizando estas variáveis após empacotar cada item, o empacotamento pode desconsiderar qualquer EM que apresenta pelo menos uma dimensão menor que a respectiva dimensão mínima presente no subconjunto de itens. Por exemplo, se no subconjunto de itens restantes a serem empacotados a menor largura tiver 3 unidades de comprimento, não é necessário manter qualquer EM com uma uma largura menor que 3 em qualquer caixa aberta. Isto significa que existirão espaços vazios não modelados dentro de algumas caixas abertas, que não serão utilizados para o restante dos itens. Os EM com área zero são automaticamente filtrados, o que ocasiona uma performance melhor no empacotamento. Nos testes empíricos deste trabalho, este filtro aumentou o desempenho das aplicações em aproximadamente 33%. O Algoritmo 13 detalha o procedimento de empacotamento de um item que atualiza a configuração da caixa e as informações $(x_i, y_i, z_i, \lambda_i)$ do item. O método recebe como parâmetro de entrada o item, uma referência para o EM selecionado e sua respectiva caixa.

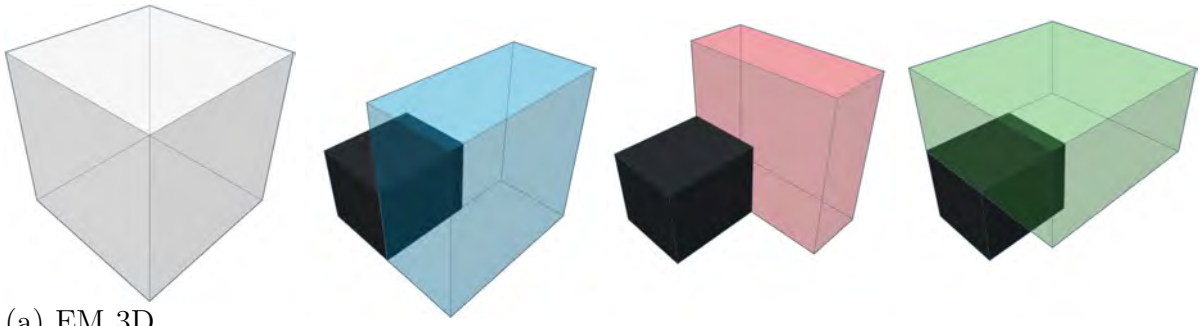
Algoritmo 13 Algoritmo de Empacotamento

Requerimento: O item deve estar orientado, tal que a alocação seja possível. O EM válido deve ser da respectiva caixa de entrada.

procedimento Aloca(**Item:** a; **Caixa:** C; **Espaço Maximal:** E)

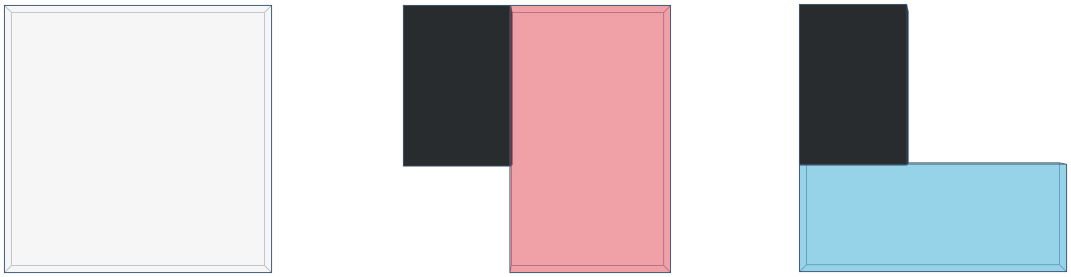
1: EMPACOTA(a, E, C) ▷ Empacota a no CI de E, e remove E de C
2: CORTE(E, a, C) ▷ Corta o antigo EM baseado no item e insere os novos na caixa
3: ATUALIZA(C, a) ▷ Filtra e atualiza o restante dos EM em C

Figura 11 – Exemplo de corte em um EM tridimensional.



(a) EM 3D selecionado. (b) EM do 1º corte. (c) EM do 2º corte. (d) EM do 3º corte.

Figura 12 – Exemplo de corte em um EM bidimensional.



(a) EM 2D selecionada. (b) EM do 1º corte. (c) EM do 2º corte.

3.4.1 Complexidade

Esta seção apresenta complexidades de tempo, estimadas pela contagem de instruções elementares, do método de empacotamento presente no Algoritmo 13.

De acordo com as representações computacionais de um item e um EM, descritas respectivamente nas Seções 3.1 e 3.3, um item que ainda não está empacotado e um EM são estruturados da seguinte forma: um ponto cartesiano, (x, y, z) ou (x, y), para identificar a localização do seu canto inferior esquerdo frontal, e as variáveis (w, h, d) ou (w, h) para as dimensões. Então, considere que a estrutura dinâmica associada à caixa que armazena os EM, que modelam seu espaço vazio, seja um vetor de acesso direto, tal que as chaves são os próprios EM. Assim, o acesso e a inserção são efetuadas em tempo constante $\Theta(1)$, pois cada EM é um conjunto de variáveis inteiras positivas. Seja m o parâmetro que majora o número de EM presentes em qualquer caixa aberta utilizada durante toda a execução da heurística construtiva. A remoção de uma chave deste vetor tem complexidade linear $\mathcal{O}(m)$. Portanto, esta é a complexidade do procedimento presente na linha 1 do Algoritmo 13.

O algoritmo de corte, presente na linha 2 do Algoritmo 13, forma novos paralelipípedos e retângulos. Posteriormente, os introduz na caixa. O número de cortes é limitado pela dimensão do problema, como exemplificado nas Figuras 11 e 12. Cada corte é feito através de de uma quantidade fixa de operações com números inteiros. Então, o procedimento que efetua todos os cortes pode ser implementado através de uma quantidade constante de passos. Como a inserção também tem complexidade constante, o algoritmo presente na linha 2 tem complexidade $\Theta(1)$. Observe que o filtro adicional para espaços redundantes, especificado na Seção 3.3, não afeta a complexidade do corte, pois é constituído de algumas

cláusulas condicionais.

A estrutura dinâmica que armazena cada EM sofre diversas alterações pelo procedimento presente na linha 3 do Algoritmo 13 em cada iteração de um item empacotado. Este procedimento de atualização é um algoritmo de dois passos, tal que o primeiro consiste em percorrer o próprio vetor associado à caixa em busca de interseções entre o item e cada EM. Ou seja, o primeiro passo verifica a interseção entre um paralelepípedo ou retângulo com no máximo m outros no espaço contido pela caixa. Assim, é necessário verificar cada EM no máximo uma vez. Quando o EM tem interseção com o item, este passo deve sintetizar novos EM através do mesmo processo de corte da linha 2, totalizando $k \leq m$ novos EM após todas as verificações. Além disso, cada EM que tem interseção com o item deve ser removido. Portanto, o primeiro passo da atualização tem complexidade $\mathcal{O}(m^2)$, devido à complexidade da remoção e à busca de complexidade $\Theta(m)$ por interseções. Observe que o processo de verificar a interseção entre dois paralelepípedos ou retângulos tem complexidade constante, pois, como a representação computacional especificada para o item e o EM é um conjunto de números inteiros positivos, a interseção entre eles pode ser verificada com duas cláusulas condicionais.

O processo de atualização prosegue para o segundo passo, que consiste em verificar se os k novos EM estão contidos nos outros. Este passo tem complexidade $\mathcal{O}(m \times k)$. Então, a complexidade final do procedimento de atualização é $\mathcal{O}(m^2)$, pois k é, no máximo, m. A Tabela 1 sumariza as complexidades de tempo associadas aos procedimentos que sintetizam o Algoritmo 13 em conjunto com a própria complexidade final do método. A tabela mostra que a complexidade final do algoritmo de empacotamento utilizando um vetor de acesso direto para armazenar os EM em cada caixa aberta é $\mathcal{O}(m + 1 + m^2) = \mathcal{O}(m^2)$.

Vale a pena enfatizar que as estratégias descritas nesta seção e na Seção 3.3 permanecem inalteradas caso o problema de empacotamento permita rotação. Observe que a modelagem através de EM pode ser utilizada independente dos itens apresentarem orientação fixa ou não. Assim, o algoritmo de empacotamento que utiliza os procedimentos de corte e atualização deve ser chamado após a heurística construtiva decidir qual EM deve conter o item já orientado, sempre assumindo que o EM escolhido pode conter o item nesta orientação. Portanto, é desnecessário realizar qualquer tipo de rotação durante este procedimento.

Tabela 1 – Complexidade dos procedimentos que compõem o Algoritmo 13.

Procedimento	**Passo**	**Complexidade**	**Observação**
Empacota	1	$\mathcal{O}(m)$	Utilizando um vetor
Corte	2	$\Theta(1)$	de acesso direto como
Atualiza	3	$\mathcal{O}(m^2)$	estrutura auxiliar, onde
Algoritmo 13	Final	$\mathcal{O}(m^2)$	cada elemento é um EM.

3.5 **Heurística Construtiva**

A qualidade das soluções encontradas nas heurísticas para o 3BP e o 2BP está fortemente relacionada com a qualidade do processo de empacotamento dos itens. Esta seção apresenta a heurística construtiva que efetua este processo. O método é utilizado nas heurísticas GRASP, Busca Tabu e BRKGA propostas neste documento, e constitui um algoritmo guloso adaptativo, tal que os candidatos avaliados são espaços maximais. A

heurística de empacotamento realiza o mesmo procedimento independente da dimensão tridimensional ou bidimensional.

Considere o conjunto de n itens $I = \{a_1, a_2, \cdots, a_n\}$. A heurística construtiva é um processo iterativo guloso adaptativo, começando sem nenhuma caixa aberta, isto é, sem caixas utilizadas. A heurística empacota um item a cada iteração até que uma solução final com todos os itens seja sintetizada. O método recebe como parâmetro de entrada uma permutação P de números inteiros, o próprio conjunto de itens e as dimensões (W, H, D) das caixas. A permutação de inteiros dita a ordem de empacotamento dos itens, que são empacotados sequencialmente. Para cada inteiro $i \in P$ na ordem que é dado, $1 \leq i \leq n$, o respectivo item a_i é submetido ao processo de empacotamento descrito na Seção 3.4. A heurística construtiva deve escolher uma caixa com um EM apropriado para o empacotamento. Caso o problema não restrinja a rotação, a heurística construtiva também deve determinar a orientação do item.

Considere o conjunto $S = \{C_1, C_2, \cdots, C_\lambda\}$ de caixas abertas em uma iteração arbitrária $1 \leq k \leq n$, isto é, a iteração em que o item a_k está sendo empacotado e λ caixas estão sendo utilizadas. A escolha de uma EM apropriada para a_k é feita através da verificação sequencial de cada caixa em S. A primeira verificação que deve ser feita é se o volume do espaço vazio de cada caixa é menor que o volume do item a_k. Assim que uma caixa passa por essa verificação, uma segunda etapa deve ser efetuada para verificar quais EM associados a esta caixa podem receber a_k. Para o caso sem rotação, um EM pode receber a_k se, e somente se, seu volume é maior ou igual que o volume de a_k e suas dimensões são maiores ou iguais que as respectivas dimensões de a_k. Caso o problema permita rotação, além da condição do EM apresentar um volume maior ou igual que o volume de a_k, ele só é um candidato válido se for capaz de conter a_k em pelo menos uma orientação. O 3BP e 2BP com rotação permitem o empacotamento dos itens com 6 e 2 diferentes orientações, respectivamente. A Figura 13 ilustra as possíveis orientações para ambos os casos através de um exemplo que apresenta um item sendo rotacionado.

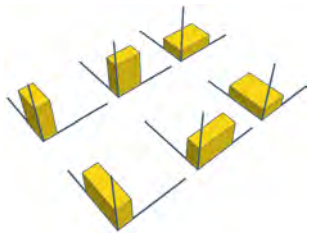
Finalmente, uma vez que o processo identifique um EM apropriado, ele deve ser avaliado através de um dos critérios que são detalhados nas Seções 3.5.1 e 3.5.2. O processo deve manter o melhor EM* candidato avaliado. Então, o item é empacotado em EM* através do Algoritmo 13 da Seção 3.4. Porém, se não há EM candidato que possa alocar o item a_k em nenhuma caixa aberta $C_j \in S$, $1 \leq j \leq \lambda$, então uma nova caixa deve ser aberta e λ deve ser incrementado em 1. Para o caso com rotação, após a seleção de EM*, a orientação do item é escolhida aleatoriamente dentre as possíveis orientações que este pode assumir para EM*.

O Algoritmo 14 detalha todo o procedimento descrito nesta seção, apresentando a heurística construtiva que efetua o empacotamento dos itens, onde f é a função que avalia o EM de acordo com um dos critérios apresentados nas Seções 3.5.1 e 3.5.2, e *Empacota* é o Algoritmo 13 da Seção 3.4.

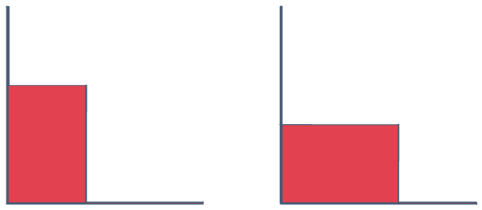
3.5.1 Best Fit e Best EM

O **Best Fit** e o **Best EM** são critérios utilizados para avaliar o potencial de um EM candidato para receber o item sendo empacotado na iteração corrente da heurística construtiva. Além disso, quando um EM é construído através do processo de corte ou atualização apresentado na Seção 3.4, o uso do critério dita como o processo de construção deve proceder para a escolha de um canto de inserção (CI) apropriado. Estes dois critérios foram propostos em (PARREÑO et al., 2010a).

Figura 13 – Exemplos que ilustram as possíveis orientações de um item.



(a) Orientação tridimensional.



(b) Orientação bidimensional.

Algoritmo 14 Heurística construtiva de empacotamento

função Construtiva(**Permutação:** P; **Itens:** I; **Dimensões:** (W, H, D))

$S \leftarrow \emptyset$ ▷ Solução construída começa vazia
$\lambda \leftarrow 0$ ▷ Número de caixas abertas começa com 0
para cada $i \in P$ **faça** ▷ Para cada item na ordem da permutação
  $a_i \leftarrow i$-ésimo item de I ▷ Item a_i será empacotado
  $\text{EM}^* \leftarrow \emptyset$ ▷ Melhor candidato
  para cada $C \in S$ **faça** ▷ Para cada caixa aberta
    se Vazio(C) $\geq$ Espaço(a_i) **então** ▷ Se o espaço vazio de C pode conter a_i
      para cada EM $\in C$ **faça**
        se Verifica(a_i, EM) **então** ▷ Se o EM pode conter a_i
          se $f(\text{EM}) < f(\text{EM}^*)$ **então** $\text{EM}^* \leftarrow \text{EM}$ ▷ Mantém o melhor EM
  se $\text{EM}^* \neq \emptyset$ **então** $C \leftarrow$ Caixa do EM^* ▷ Referência para o empacotamento
  senão ▷ Não há candidatos
    $C \leftarrow$ Caixa(W, H, D) ▷ Uma nova caixa é aberta
    $S \leftarrow S \cup \{C\}$ ▷ Adiciona a nova caixa em S
    $\lambda \leftarrow \lambda + 1$ ▷ Atualiza o número de caixas abertas
    $\text{EM}^* \leftarrow$ Espaço(C, W, H, D) ▷ EM^* recebe o único EM da nova caixa
  Orienta(a_i, EM^*) ▷ Se há rotação, escolhe uma orientação aleatória para a_i.
  Empacota(a_i, C, EM^*) ▷ Empacota a_i através do Algoritmo 13
retorna S, λ ▷ Retorna a solução S e o número de caixas abertas

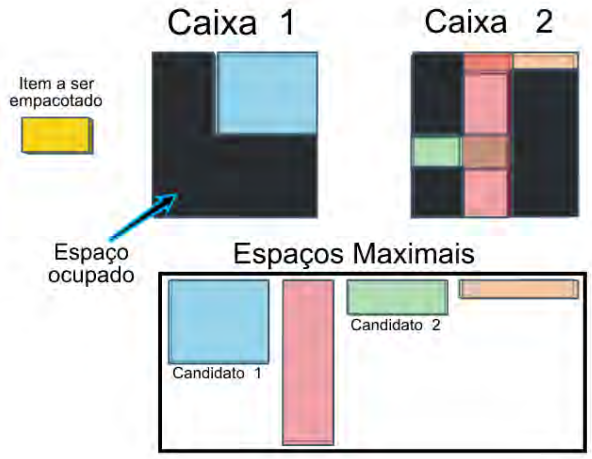
O Best Fit avalia o EM de acordo com a quantidade de volume ocupado da caixa. Esta estratégia depende dos itens que já estão empacotados. O melhor EM candidato é aquele que está associado à caixa com o menor espaço vazio. Caso haja empate, o método deve selecionar o primeiro EM da primeira caixa avaliada, ignorando qualquer outro EM encontrado com valor igual. Assim, o processo que utiliza o Best Fit tem por objetivo carregar o máximo possível as caixas abertas priorizando as primeiras.

O Best EM é independente dos itens já empacotados. Este critério determina o valor do EM baseado somente no próprio candidato. O melhor EM candidato é aquele que minimiza o espaço vazio do próprio EM após o empacotamento do item. O processo que utiliza este critério tem por objetivo minimizar os buracos formados nas caixas já abertas. A Figura 14 mostra um exemplo com uma solução que possui duas caixas abertas no momento que um item deve ser empacotado com orientação fixa. A figura ilustra os EM contidos em ambas as caixas destacando os dois candidatos para receber o item que será empacotado. Se o critério Best Fit for utilizado, então o processo empacotará o item no

EM azul da primeira caixa, pois este candidato está associado à caixa que tem a menor quantidade de espaço vazio. Porém, se o critério for o Best EM, então o EM verde da segunda caixa receberá o item, pois o volume total que sobra do próprio é menor.

Como especificado na Seção 3.3, o CI é o canto onde o item será posicionado quando empacotado. Desta maneira, quando a heurística utiliza um destes dois critérios, o CI deve ser selecionado de maneira a minimizar a distância entre o canto do item e o respectivo canto da caixa. Desta maneira, o item deve ser empacotado o mais próximo possível do respectivo canto da caixa. Caso exista empate, então um dos cantos candidatos deve ser escolhido aleatoriamente. A Figura 15 ilustra um exemplo para determinação do CI de dois novos EM no caso bidimensional. A figura mostra uma caixa aberta da solução, tal que um item acabou de ser empacotado. Logo, dois novos EM foram formados pelo procedimento de corte. Para o EM representado pela cor azul, o canto inferior direito tem a distância mínima com o respectivo canto da caixa. Na construção do EM vermelho, o processo checa que há um empate entre dois cantos: o superior esquerdo e o inferior esquerdo. Então, a escolha do CI apropriado pode ser qualquer um desses dois candidatos escolhido aleatoriamente. Assim, a ideia fundamental da utilização do Best Fit ou Best EM é realizar o empacotamento dos itens começando pelos cantos, em seguida preenchendo os lados e finalizando no centro da caixa.

Figura 14 – Exemplo bidimensional do critério de avaliação Best Fit e Best EM.



(a) Configuração das caixas e lista de EM disponíveis.

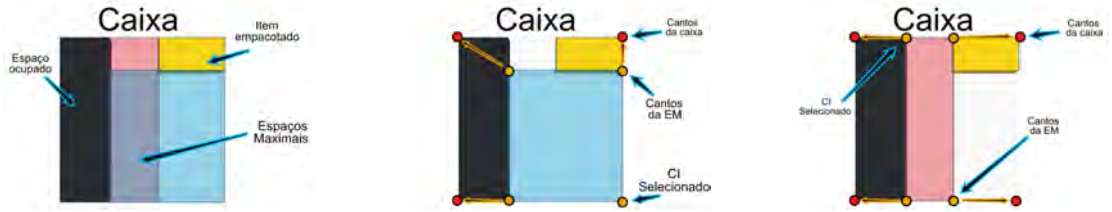
(b) EM selecionado pelo Best Fit.

(c) EM selecionado pelo Best EM.

3.5.2 DFTRC

O **DFTRC** (*Distance to the Front Top Right Corner*) é um critério proposto por (GONÇALVES; RESENDE, 2013). Sua finalidade é avaliar um EM candidato no mo-

Figura 15 – Exemplo bidimensional da seleção do CI de dois novos EM com o Best Fit ou Best EM.



(a) Dois novos EM após o empacotamento.

(b) CI como canto inferior direito por distância 0.

(c) Empate de 2 candidatos. CI por escolha aleatória.

mento em que a heurística construtiva de empacotamento precisa selecionar uma localização para o empacotamento de um item. Como mencionado anteriormente, a determinação do CI na construção de um EM depende deste critério. Para o DFTRC, o CI é sempre o canto inferior esquerdo posterior, pois a ideia fundamental deste método é compactar os itens o mais distante possível do canto superior direito frontal da caixa.

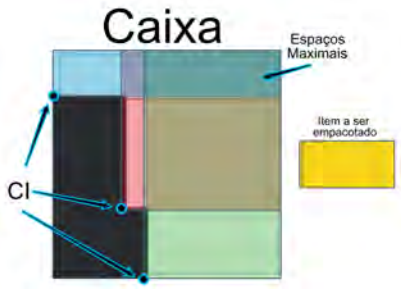
O melhor EM neste método é o candidato que maximiza a distância entre o canto superior direito frontal do item para o respectivo canto superior direito frontal da caixa. Caso o problema permita a rotação dos itens, então todas as possíveis orientações devem ser consideradas. Ou seja, o melhor EM candidato é aquele que maximiza a distância entre o canto superior direito frontal do item para o respectivo canto da caixa na possível orientação que apresenta a maior distância. A Figura 16 ilustra um exemplo bidimensional com rotação do DFTRC. A figura mostra uma caixa aberta com 3 EM que modelam o seu espaço vazio em conjunto com um item a ser empacotado, destacando os respectivos CI. O exemplo ilustra que o método que utiliza o DFTRC deve calcular cada distância entre os cantos superior direito frontal da caixa e do item após alocação. Além disso, o exemplo gráfico também apresenta que o critério considera a distância de todas as possíveis orientações que o item pode assumir para determinado EM. No terceiro EM, ilustrado pela cor verde, o gráfico destaca na cor azul a distância que maximiza o valor objetivo deste critério. Assim, o terceiro EM seria o melhor candidato disponível para a caixa deste exemplo. Em caso de empate entre dois candidatos, o EM selecionado é o primeiro que foi avaliado.

O DFTRC também é capaz de selecionar uma orientação para o item, escolhendo aquela que maximiza o critério usado. Um exemplo pode ser visto na Figura 16d, onde a orientação selecionada seria a destacada em azul. Nos testes empíricos do trabalho original que propôs o DFTRC, os resultados computacionais reportados pelos autores mostram que utilizar a orientação que maximiza o DFTRC gera resultados consideravelmente piores quando comparados à utilização de orientações aleatórias após a escolha do EM. As instâncias utilizadas foram as mesmas detalhadas na Seção 4.1 deste trabalho. O critério de seleção utiliza a melhor orientação do item para a avaliar o EM, porém, a orientação final do item a ser empacotado na heurística construtiva, que deve ser passado como parâmetro de entrada para o Algoritmo 13, é escolhida aleatoriamente após a seleção do melhor EM assim como detalhado no Algoritmo 14.

3.5.3 Complexidade

Esta seção apresenta a complexidade de tempo, determinada através da avaliação da quantidade de operações primitivas, da heurística construtiva detalhada pelo Algo-

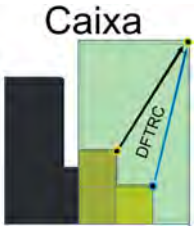
Figura 16 – Exemplo bidimensional com rotação do critério de escolha DFTRC.



(a) Configuração de uma caixa aberta com o respectivo item a ser empacotado, destacando os respectivos CI de cada EM.

(b) DFTRC para a 1^o EM.

(c) DFTRC para a 2^o EM.



(d) DFTRC para a 3^o EM.

ritmo 14.

Seja n a quantidade de itens de uma instância do 3BP ou 2BP e m um valor que majora o número de EM presentes em uma caixa aberta durante toda a execução da heurística construtiva. A cláusula de repetição da linha 3 do Algoritmo 14 utiliza cada termo da permutação P para empacotar uma sequência de itens. Como P tem n elementos, este *loop* se repete n vezes. A cláusula de repetição da linha 6 percorre cada caixa aberta da solução parcial S em busca de localizações válidas para empacotar o item da iteração atual. Sabendo que todo item pode ser empacotado em uma caixa vazia, o número máximo de repetições que este *loop* pode efetuar é o próprio limite superior n do problema, pois no pior caso temos uma instância onde é impossível empacotar qualquer par de itens. Isto é, no pior caso temos uma caixa aberta a cada iteração, e a cláusula de repetição 6 deve percorrer até n caixas.

O conjunto de passos presente nas linhas 7 até 10 são uma verificação para avaliar se na caixa há uma localização válida para o item. A cláusula condicional da linha 7 verifica se o espaço vazio da caixa é maior ou igual que o volume do item. Esta verificação pode ser implementada através de uma comparação entre números inteiros, pois todos os parâmetros do 3BP e do 2BP em relação à dimensão dos itens e da caixa são inteiros positivos. A cláusula de repetição da linha 8 verifica cada EM presente na caixa aberta atual em busca de EM candidatos para o empacotamento do item. Se a implementação utilizar um vetor de acesso direto para armazenar os EM associados à caixa, a complexidade de acesso é $\Theta(1)$. Assim, a verificação da linha 9 pode ser feita através de duas comparações entre inteiros positivos. Se o problema permitir a rotação dos itens, a complexidade da verificação permanece constante, pois os itens podem assumir até 6 ou 2 orientações nos casos tridimensional e bidimensional respectivamente (como ilustrado na Figura 13).

Caso o EM seja um candidato válido para o empacotamento, o passo 10 mantém o melhor EM na variável EM^*, mas a implementação deve antes verificar se o valor de EM^* é diferente de nulo. Se o valor for nulo, então EM^* deve receber o EM candidato, caso contrário, o melhor candidato deve ser mantido em EM^* de acordo com o critério

Best Fit, Best EM ou DFTRC detalhados nas Seções 3.5.1 e 3.5.2. O critério Best Fit avalia o EM candidato de acordo com a quantidade de volume ocupado da caixa. Desta maneira, a implementação do critério consiste de uma comparação com números inteiros, pois, como citado anteriormente, os parâmetros do 3BP e do 2BP relacionados com as dimensões são inteiros positivos. Analogamente, a implementação do Best EM também é uma comparação com números inteiros positivos, que, por sua vez, utiliza somente a quantidade de volume do próprio EM para a sua avaliação (as dimensões de um EM também são inteiros positivos). Se existir a possibilidade de rotação, uma das possíveis orientações deve ser escolhida aleatoriamente. A complexidade de ambos os critérios continua constante, por causa do número limitado de orientações. Finalmente, o critério DFTRC calcula a distância entre dois pontos cartesianos para avaliar o EM e também é um critério que pode ser implementado com complexidade constante. Observe que o DFTRC realiza um número de operações maior caso o problema permita a rotação dos itens, porém a verificação de possíveis orientações de um item em um EM consiste de comparações entre números inteiros positivos (6 para o 3BP e 2 para o 2BP). Então, a possibilidade de rotação também não afeta a complexidade do DFTRC. Como as verificações para EM* são feitas com complexidade $\Theta(1)$ e um EM é composto por um conjunto fixo de números inteiros, a complexidade do passo 10 é constante.

Os passos 11 até 16 armazenam os parâmetros de entrada para o Algoritmo 13, que é executado posteriormente no passo 18. A operação de cópia de um EM tem complexidade constante, então o passo 11 também tem complexidade constante. O método *Orienta* da linha 17 escolhe aleatoriamente uma orientação dentre as possíveis para o item baseado em EM*. Como mencionado anteriormente, a escolha pode ser implementada com um número constante de operações e comparações, por causa da quantidade limitada de possíveis orientações dos problemas de empacotamento. Vale enfatizar que o método *Orienta* necessita de um gerador de números pseudoaleatórios eficiente para não alterar sua complexidade, por exemplo, o Mersenne Twister. Finalmente, a complexidade do método *Empacota*, presente na linha 18, é detalhada na Seção 3.4.1, sendo $\mathcal{O}(m^2)$.

Então, a complexidade de tempo do Algoritmo 14 é $\mathcal{O}(n \times \text{máximo}(n \times m, m^2))$. O termo $n \times m$ majora m^2 em minorias particulares dos problemas de empacotamento, por exemplo, instâncias que consistem de itens grandes, tal que a maioria possui algumas ou todas as dimensões com o mesmo tamanho da própria caixa. Assim, a Tabela 2 sumariza a complexidade de tempo dos passos descritos nos parágrafos anteriores que compõem o Algoritmo 14 em conjunto com a própria complexidade final para a grande maioria das instâncias. O restante dos passos não mencionados, inclusive os passos 1 e 2, têm complexidade constante.

Tabela 2 – Sumário da complexidade de tempo do Algoritmo 14.

Instrução	**Passo**	**Complexidade**	**Observação**
1^o *Loop*	3	$\mathcal{O}(n)$	
2^o *Loop*	6	$\mathcal{O}(n)$	
1^o Condição	7	$\Theta(1)$	Utilizando um
3^o *Loop*	8	$\mathcal{O}(m)$	vetor de acesso
2^o Condição	9	$\Theta(1)$	direto como
Manter EM*	10	$\Theta(1)$	auxiliar para o
Best Fit ou EM/DFTRC	10	$\Theta(1)$	armazenamento
Cópia de uma EM	11	$\Theta(1)$	dos EM de uma
Orienta	17	$\Theta(1)$	caixa aberta.
Empacota	18	$\mathcal{O}(m^2)$	
Algoritmo 14	Final	$\mathcal{O}(n \times m^2)$*	

*Complexidade para a grande maioria das instâncias.

3.6 **Busca Local**

A heurística de empacotamento apresentada na Seção 3.5 pode ser combinada com meta-heurísticas para a resolução do 3BP ou do 2BP. Por exemplo, o método pode gerar uma solução inicial ou o ponto de partida de uma iteração. Porém, algumas meta-heurísticas, como o GRASP, necessitam de mecanismos para a busca de soluções viáveis no espaço de soluções do problema, com o objetivo de sintetizar uma fase de melhora e encontrar ótimos locais. Esta seção apresenta uma heurística que pode ser utilizada para este fim, recebendo uma solução já construída como parâmetro de entrada. O foco principal é propor um método de busca local, dentre os introduzidos na Seção 1.2.2, para servir como mecanismo capaz de encontrar soluções de alta qualidade a partir de uma solução construída pela heurística construtiva proposta para os problemas de empacotamento.

Uma estratégia de busca local para os problemas de empacotamento pode ser construída através da movimentação dos itens de uma solução viável já existente entre diferentes caixas abertas ou dentro de si própria. Quando se trata da execução prática, tais técnicas são difíceis de serem sintetizadas e aplicadas com sucesso. Por exemplo, se uma busca local movimenta itens ponto a ponto dentro do espaço cartesiano vazio de uma caixa, podemos sintetizar uma heurística construtiva com o mesmo mecanismo que minuciosamente escolhe pontos adequados para o item, verificando a melhor posição ponto a ponto. Um segundo exemplo seria sintetizar uma busca local que move o item considerando um novo EM de uma caixa aberta qualquer para situá-lo em um novo canto de inserção. A própria heurística construtiva proposta na Seção 3.5 já seria capaz de construir soluções que dificilmente seriam melhoradas através deste tipo de busca local ou de qualquer outra que aplique mecanismos que movimente itens através de diferentes EM. Portanto, a movimentação de itens entre caixas ou dentro de si própria dificilmente melhora as soluções obtidas por uma heurística construtiva que realiza em seu processo de execução um mecanismo equivalente para os problemas de empacotamento.

Desta maneira, uma estratégia alternativa com um potencial de encontrar soluções promissoras, partindo de uma sintetizada por uma heurística construtiva, é efetuar o reempacotamento dos itens através da própria heurística construtiva utilizada. Assim, as iterações da busca local devem sistematicamente esvaziar parcialmente ou completamente caixas abertas, de maneira a reempacotar os itens com o próprio mecanismo implemen-

tado pela abordagem construtiva. Então, a vizinhança desta busca local é composta por soluções que são obtidas a partir do reempacotamento de itens.

A busca local proposta neste trabalho consiste em esvaziar completamente um par de caixas ignorando qualquer uma com 100% de carga. Em seguida, utilizar a heurística construtiva proposta na Seção 3.5 para realizar o reempacotamento dos itens. Para isso, um novo parâmetro de entrada deve ser introduzido no Algoritmo 3.5, tal que esta variável armazene uma configuração de caixas abertas com seus respectivos espaços maximais, ou seja, uma solução parcialmente construída. Desta maneira, as únicas modificações necessárias no pseudocódigo da heurística construtiva seriam: a solução parcial S seria introduzida como parâmetro de entrada, o passo 1 seria removido e o passo 2 seria carregar a variável λ com o número de caixas abertas em S. Assim, a permutação P de entrada agora representa a ordem de empacotamento dos $\hat{n} \leq n$ itens presentes no conjunto I de entrada, os quais devem ser os itens que faltam para completar S. A permutação pode ser aquela que mantém a mesma ordem original de itens empacotados da solução original de entrada. Para isto, o parâmetro que dá a solução inicial da busca local deve carregar sua permutação P usada.

A ideia principal desta busca local é realizar o reempacotamento dos itens de maneira a encontrar soluções que utilizem até 2 caixas a menos que a solução original. O método de reempacotamento deve desistir da solução atual caso tenha de abrir mais do que uma ou duas novas caixas, dependendo da função objetivo utilizada. De acordo com a Seção 3.2, a função objetivo trivial (FOT) avalia a solução somente com o número de caixas utilizadas. Se o procedimento tiver de abrir uma segunda caixa, ele pode desistir imediatamente, pois trata-se de uma solução com qualidade equivalente ou pior do que a atual. Porém, se a função objetivo alternativa (FOA) for utilizada, então o processo de reempacotamento deve somente desistir quando identificar que três caixas devem ser abertas, pois uma solução com o mesmo número de caixas que a atual ainda pode ser melhor. Na prática, a busca local utiliza uma estrutura de vizinhança que realiza o empacotamento dos itens simulando efetivamente diferentes permutações de entrada para a heurística construtiva. Por exemplo, esvaziar duas caixas e reempacotar os itens é equivalente a alterar a permutação de entrada utilizada originalmente para a solução gerada pela heurística construtiva. A busca local é capaz de desistir de soluções não promissoras e preservar caixas abertas com boas configurações de itens. Desta maneira, como o processo de empacotamento dos itens tem um alto custo computacional de tempo, esta estratégia gera uma vizinhança composta por soluções distintas evitando o completo processo de empacotamento, economizando tempo de execução para checar à qualidade de diferentes soluções.

O Algoritmo 15 detalha o procedimento de busca local proposto com *Best Improvement*, tal que a função *Reempacota* é a heurística constritiva presente no Algoritmo 14 com a modificação detalhada nesta seção. O método *First Improvement* também pode ser utilizado, e o critério de escolha da heurística construtiva pode ser o Best Fit, Best EM ou DFTRC apresentados nas Seções 3.5.1 e 3.5.2 respectivamente. Observe que este método só pode ser aplicado caso a solução inicial s_0 apresente duas ou mais caixas abertas, pois não há como melhorar uma solução que utiliza somente uma caixa. A estrutura de vizinhança usada são as soluções geradas com sucesso pelo método *Reempacota*.

3.6.1 Complexidade de Uma Iteração

Esta seção apresenta a complexidade de tempo de uma iteração do Algoritmo 15, que realiza uma busca local através do reempacotamento de itens com a heurística construtiva

Algoritmo 15 Busca Local para o 3BP/2BP com *Best Improvement*

Requerimento: Solução inicial s_0 deve ter pelo menos 2 caixas abertas
Observação: a estrutura de vinhança V constitui-se das soluções geradas pelo método *Reempacota*, portanto, não é passada como parâmetro de entrada.
função BuscaLocal(**Solução inicial:** s_0; **Dimensões:** (W, H, D))

1: $s^* \leftarrow s_0$ ▷ Melhor solução recebe solução inicial
2: **faça**
3: $s \leftarrow s^*$ ▷ Solução atual recebe melhor solução encontrada
4: $\lambda \leftarrow$ Número de caixas de s ▷ Caixas abertas da solução atual.
5: **para** i de 1 até $\lambda - 1$ **faça**
6: $C \leftarrow$ Caixa C_i de s ▷ i-ésima caixa da solução atual.
7: **se** C estiver com 100% de carga **então**
8: Avançar o *loop* ▷ Ignora caixas promissoras.
9: **para** j de $i + 1$ até λ **faça**
10: $C' \leftarrow$ Caixa C_j de s ▷ j-ésima caixa da solução atual.
11: **se** C' estiver com 100% de carga **então**
12: Avançar o *loop* ▷ Ignora caixas promissoras.
13: $s' \leftarrow s$ ▷ Solução parcial recebe solução atual.
14: $I \leftarrow$ Todos os itens de C e C' ▷ Conjunto de itens para reempacotar.
15: **para** $a_k \in I$ **faça**
16: $P \leftarrow k$-ésimo termo da permutação de s^* ▷ Mantém a ordem original
17: Apagar as caixas C_i e C_j da solução s'
18: $s' \leftarrow$ REEMPACOTA(s', P, I, (W, H, D)) ▷ Retorna $\emptyset$ se desistir.
19: **se** $s' \neq \emptyset$ **então**
20: **se** s' é melhor que s^* **então** $s^* \leftarrow s'$ ▷ Nova melhor solução.
21: **enquanto** Houver melhora em s^*
22: **retorna** s^* ▷ Retorna a melhor solução

detalhada no Algoritmo 14.

O passo 2 consiste de uma cláusula de repetição que reinicializa o processo de busca até encontrar um ótimo local. O número de repetições máximo depende da qualidade da solução parcial de entrada. Suponha que a instância resolvida possua um ótimo global composto somente por uma caixa que empacota todos os itens. No pior caso, temos uma solução inicial s_0 de baixa qualidade que é melhorada a cada iteração da busca local. Por exemplo, uma solução trivial de baixa qualidade pode ser obtida através do empacotamento de cada item em uma caixa distinta, levando a uma solução com n caixas. Sabendo que a busca local proposta é capaz de melhorar até 2 caixas a cada iteração, suponha também que o método encontre uma nova solução com uma caixa a menos a cada iteração, até que, após $n-1$ iterações, a busca local finalmente retorne o ótimo global da instância com uma única caixa aberta. Desta maneira, no pior caso, o número de repetições máximas da cláusula 2 é $n - 1$. Este é um caso especial que raramente acontece na prática. Além disso, as iterações da busca local, que são efetuadas dentro desta cláusula de repetição, ficam cada vez mais rápidas de serem executadas, uma vez que melhores soluções são encontradas e um número maior de desistências no reempacotamento acontece.

Neste trabalho, as soluções parciais usadas são geradas pela heurística construtiva proposta na Seção 3.5, e geralmente apresentam uma boa qualidade. Na prática, a busca local raramente efetua mais do que 3 repetições. Muitas das soluções geradas, mesmo

que com permutações de entrada aleatórias, possuem potencial de melhora limitado por causa da própria natureza dos problemas de empacotamento. Na grande maioria dos casos a busca local realiza 2, 1 ou nenhuma repetição. Além disso, quando uma solução é melhorada com uma ou duas caixas a menos, esta nova configuração de caixas abertas apresenta uma carga média maior, ou seja, as caixas estão mais cheias e com um número reduzido de espaços vazios. Esta quantidade reduzida de espaço livre, por sua vez, pode ser modelada com uma quantidade menor de espaços maximais. Portanto, a iteração seguinte da busca local é, geralmente, mais rápida. Desta maneira, a complexidade no pior caso deste algoritmo não corresponde ao seu comportamento prático, pois dificilmente podemos prever a quantidade de repetições do passo 2. Então, esta análise é feita somente para uma iteração da busca local, a fim de melhor corresponder ao comportamento prático da abordagem.

Os passos 1, 3, 13 e 21 efetuam atribuições com variáveis cujos tamanhos variam de acordo com os parâmetros de entrada da instância. Portanto, a cópia de uma solução não tem complexidade constante. A representação computacional de uma solução é detalhada na Seção 3.1. Uma solução consiste de uma lista de caixas abertas e variáveis auxiliares que são implementadas através de tipos básicos de uma linguagem de programação. A representação computacional de uma caixa aberta possui uma lista de itens empacotados, uma lista de EM e algumas variáveis adicionais também de tipo básico. Finalmente, a representação de um EM e um item usa um conjunto fixo de variáveis positivas inteiras. Como uma solução viável apresenta todos os itens da instância, o conjunto total de caixas abertas possui os n itens empacotados. Então, a complexidade da cópia é independente do número de caixas da solução. Logo, a complexidade da cópia de uma solução depende da quantidade de itens n da instância e do valor m que majora a quantidade de EM presentes em qualquer caixa aberta durante a execução da iteração. Desta maneira, a complexidade final da atribuição presente nos passos 1, 3, 13 e 21 é $\mathcal{O}(\text{máximo}(n, m))$. A complexidade da cópia é dada por $\mathcal{O}(n)$ quando uma solução, por exemplo, apresenta caixas abertas que estão com carga elevada ou completamente cheias, isto é, caixas abertas com pouco espaço livre, que pode ser modelado através de poucos EM comparados com a quantidade de itens. Na grande maioria dos casos, as caixas possuem configurações complexas que necessitam de vários espaços maximais para modelar seu espaço vazio, tal que m é um valor de magnitude maior que n. Portanto, para a grande maioria dos casos, a complexidade que melhor representa o comportamento prático deste procedimento de cópia é $\mathcal{O}(m)$. Assim, esta é a complexidade considerada para a cópia de uma solução.

As cláusulas de repetição presentes nos passos 5 e 9 deste algoritmo iteram sobre as caixas abertas da solução atual. No pior caso, a solução utiliza o limite superior do problema de empacotamento, que é o próprio número de itens n presente na instância. Como citado anteriormente, uma solução viável fácil de construir seria empacotar cada item em uma caixa distinta. Desta maneira, os passos 5 e 9 efetuam no máximo n repetições.

O passo 14 deve esvaziar todos os itens de duas caixas da solução atual. No pior caso, a solução atual s pode ter duas caixas abertas, com todos os itens presentes. A função deverá iterar sobre todos os itens da instância para sintetizar o conjunto I. Assim, a complexidade deste passo é $\mathcal{O}(n)$. Mesmo se este caso for identificado e a implementação mantiver uma variável auxiliar que armazena a lista total de itens, a complexidade permanece a mesma: o pior caso, desta vez, seria uma solução inicial com 3 caixas abertas, onde ambas as caixas esvaziadas C_i e C_j empacotam um conjunto de $n - 1$ itens. Então, neste caso especial, a complexidade também não se altera. Sendo assim, o passo da linha 14 realiza no máximo

n repetições. O passo 15 realiza repetições baseadas no número de itens do conjunto I. Analogamente, este conjunto pode ter no máximo n itens. Consequentemente, o número de repetições máximo do passo 15 também é n.

O passo 17 apaga duas caixas da solução s' tornando-a parcial. Se a estrutura de dados utilizada na representação computacional de uma solução para armazenar suas caixas aberta for uma lista duplamente encadeada, onde cada elemento é uma caixa, a complexidade da remoção e inserção de um elemento no final é $\Theta(1)$. Desse modo, as complexidades previamente apresentadas nas Seções 3.4.1 , 3.5.3 e nesta não são afetadas. Todo o acesso necessário nesta estrutura é sequencial e as inserções podem ser feitas no final da lista encadeada. Portanto, o processo de remoção efetuado no passo 17 pode ser implementado com complexidade constante.

Finalmente, a função *Reempacota*, utilizada no passo 18, é uma reimplementação da heurística construtiva detalhada na Seção 14 pelo Algoritmo 14 com diferentes parâmetros de entrada. Com a modificação detalhada na seção anterior, a complexidade do método permanece a mesma e depende da quantidade de itens no conjunto I. Como citado anteriormente, o conjunto I pode conter até n itens no pior caso. Dessa forma, a complexidade deste passo, para a maioria das instâncias, é $\mathcal{O}(n \times m^2)$, tal que m é o valor que majora o número de EM presente em qualquer caixa aberta.

Então, para a maior parte das instâncias, a complexidade final de uma iteração da busca local proposta no Algoritmo 5 é dada pelos passos 5, 9 e 18, sendo $\mathcal{O}(n^3 \times m^2)$. O passo 3 não afeta a complexidade final, devido à clausula de repetição presente no passo 5. Assim como os passos 13, 14, 15 e 17 não afetam a complexidade final devido à complexidade do método *Reempacota* presente no passo 18. Nos casos especiais detalhados na Seção 3.5.3, a heurística de empacotamento apresenta uma complexidade no pior caso diferente. Nestes casos especiais, a complexidade de uma iteração da busca local é $\mathcal{O}(n^4 \times m \log m)$. A análise de complexidade desta seção é sumarizada pela Tabela 3. A tabela relaciona a instrução com o passo e sua complexidade, apresentando também a complexidade final de uma iteração da Algoritmo 15. Observe que a complexidade não se altera se o método utilizar o *First Improvement*, pois, no pior caso, uma melhora pode não ser encontrada ou somente ser determinada na última iteração dos passos 5 e 9, ou seja, na iteração em que o método esvazia as duas últimas caixas $i = \lambda - 1$ e $j = \lambda$.

Tabela 3 – Sumário da complexidade de tempo do Algoritmo 15.

Instrução	**Passo**	**Complexidade**	**Observação**
Cópia da solução	1, 3, 13 e 21	$\mathcal{O}(m)$*	
1^o *Loop* da iteração	5	$\mathcal{O}(n)$	Utilizando uma lista
2^o *Loop* da iteração	9	$\mathcal{O}(n)$	duplamente encadeada
Construção de I	14	$\mathcal{O}(n)$	como auxiliar para
3^o *Loop* da iteração	15	$\mathcal{O}(n)$	o armazenamento
Remoção de caixas	17	$\Theta(1)$	das caixas abertas
Reempacota	18	$\mathcal{O}(n \times m^2)$*	presente em uma
Uma iteração do Algoritmo 15	Final	$\mathcal{O}(n^3 \times m^2)$*	solução.

*Complexidade para a grande maioria das instâncias.

3.7 **Variable Neighborhood Descent**

Como mencionado na Seção 3.6, todas as meta-heurísticas necessitam de mecanismos para melhorar soluções, com o objetivo de sintetizar uma fase de melhora que começa de uma solução inicial já construída como entrada. Uma fase de melhora que utiliza uma busca local é capaz de analisar a qualidade de diversas soluções distintas através de modificações na inicial em busca de ótimos locais. Contudo, o processo de busca pode ficar atrelado a um determinado local ou direção no espaço de soluções impossibilitando alguns ótimos locais de serem encontrados, os quais podem ser um ótimo global para o problema. Logo, a fase de melhora também pode necessitar de um mecanismo de diversificação que amplifica o espaço de busca. Nos problemas de empacotamento, este é o caso da busca local proposta na Seção 3.6.

A busca local proposta é capaz de encontrar boas soluções através do reempacotamento dos itens, de maneira que uma nova solução é construída sem que todo o processo custoso de empacotamento seja efetuado. Porém, na prática, muitas caixas são utilizadas na solução final, e a busca local reempacota somente o conteúdo de duas caixas. Deste modo, as soluções encontradas podem se repetir com frequência quando a heurística que a utiliza realiza várias iterações durante muito tempo. Uma das maneiras de resolver este problema é realizar o reempacotamento de uma quantidade maior de itens, esvaziando um número maior de caixas. No entanto, boas configurações de caixas deixam de serem mantidas, e o processo de reempacotamento requer um tempo maior de execução para obter uma nova solução. Uma alternativa, então, é utilizar essas estratégias de reempacotamento em conjunto, tal que se imponha um balanço entre busca e diversificação. Para isso, esta seção apresenta uma heurística *Variable Neighborhood Descent* (VND), que combina a busca local proposta na Seção 3.6 com estratégias de reempacotamento que esvaziam um número maior de caixas. O foco principal desta estratégia VND é melhorar a solução atual mantendo as boas configurações de caixas abertas através da busca local, porém adicionando estratégias semelhantes com o mesmo processo de reempacotamento que diversificam e amplificam o espaço de busca introduzindo uma penalidade de tempo de processamento razoável.

O VND proposto segue o Algoritmo 6 introduzido na Seção 1.2.3, que detalha toda a fundamentação teórica do VND independente do problema aplicado. Um total de quatro vizinhanças V_k são utilizadas, onde V_1 é composto pelas soluções geradas com sucesso no passo 18 do Algoritmo 15, o qual detalha a busca local proposta neste trabalho. Dessa forma, a busca local da Seção 3.6 é usada diretamente quando $k = 1$, podendo ser utilizada tanto nas versões com o *Best Improvement* quanto *First Improvement*. As outras três vizinhanças V_2, V_3 e V_4 são compostas por uma única solução que é obtida através do reempacotamento do conteúdo de metade das caixas da solução corrente. Desta maneira, o tempo de execução para obter cada solução destas três vizinhanças não é tão alto quanto o de construir uma nova solução. Este reempacotamento é exatamente o processo detalhado na Seção 3.6: uma pequena modificação na heurística construtiva, presente no Algoritmo 14 e detalhada na Seção 3.5, deve ser feita de modo a receber uma solução parcialmente construída como parâmetro de entrada e sua permutação P que mantém a ordem dos itens empacotados, sendo o restante do método exatamente o mesmo detalhado no Algoritmo 14.

O vizinho em V_2 é obtido esvaziando a metade das caixas com menor carga da solução corrente e, em seguida, reempacotando seus itens através da heurística construtiva presente no Algoritmo 14. Analogamente, o único vizinho contido em V_3 é obtido da

mesma maneira, mas esvaziando metade das caixas com maior carga. De forma análoga, V_4 também possui uma única solução, porém o método esvazia metade das caixas aleatoriamente.

A ideia intuitiva da vizinhança V_2 é explorar o reempacotamento alocando os itens nas caixas com maior carga, pois estas caixas têm tendência a manter os itens que são difíceis de se empacotar em conjunto com outros. Seguindo esta mesma ideia, a vizinhança V_3 procura reempacotar os itens que aparecem com frequência nas caixas que possuem menor carga, com o objetivo de realizar empacotamentos que forcem estes itens a serem empacotados com os itens mais difíceis de serem alocados em conjunto. Finalmente, a vizinhança V_4 é uma tentativa caótica de empacotar itens, a fim de encontrar soluções diversas. A ideia de V_4 é esvaziar metade das caixas aleatoriamente. Os itens a serem reempacotados durante esta vizinhança **não** devem manter a mesma ordem da solução atual, e sim uma ordem aleatória através de uma permutação P aleatória. Note que o processo de reempacotamento deve desistir de sintetizar uma solução se uma quantidade elevada de caixas forem abertas comparada com a quantidade esvaziada, dependendo de qual função objetivo foi utilizada: o FOT ou FOA. Assim como detalhado na Seção 3.2, o FOT avalia uma solução somente com o número de caixas abertas. Neste caso, o procedimento de reempacotamento pode desistir caso o número de caixas abertas atinja a mesma quantidade de caixas esvaziadas, porque isto significa que a solução que será construída neste reempacotamento já é equivalente ou pior que a atual. No entanto, se o FOA for utilizado, duas soluções com o mesmo número de caixas abertas podem ter avaliações diferentes. Assim, o procedimento de reempacotamento deve desistir somente quando o número de caixas abertas for maior que a quantidade de caixas esvaziadas. Isto significa que o reempacotamento desiste rapidamente conforme melhores soluções são encontradas. O Algoritmo 16 detalha o processo das vizinhanças V_2, V_3 e V_4. A Tabela 4 sumariza as quatro vizinhanças, relacionando-as com o método utilizado. Dessa forma, o VND proposto utiliza essas vizinhanças e executa de acordo com o Algoritmo 6 apresentado no Capítulo 1.

Tabela 4 – Relação entre a vizinhança V_k e seu método no VND proposto.

Vizinhança	**Método**
V_1	Busca Local da Seção 3.6 com Best ou *First Improvement*
V_2	Esvazia metade das caixas com menor carga
V_3	Esvazia metade das caixas com maior carga
V_4	Esvazia metade das caixas aleatoriamente

3.7.1 Complexidade das Vizinhanças Propostas

O VND é um processo iterativo que utiliza múltiplas vizinhanças para melhorar uma solução inicial que é dada como parâmetro de entrada. Desta maneira, o Algoritmo 6, em conjunto com as vizinhanças introduzidas na seção anterior, compõem o VND proposto para os problemas de empacotamento. A cláusula de repetição externa presente no passo 3 do VND possui o mesmo problema que o passo 2 da busca local detalhada pelo Algoritmo 5: a análise da quantidade máxima de repetições deste passo dificilmente corresponde ao comportamento prático do algoritmo. O número máximo de repetições acontece raramente em um caso especial análogo ao detalhado na Seção 3.6.1: na suposição de que uma solução trivial de má qualidade seja passada e seja possível responder a

Algoritmo 16 Vinhança para $k \in \{2, 3, 4\}$

Requerimento: Solução s deve ter pelo menos 2 caixas abertas.
função V_k(**Solução:** s; **Dimensões:** (W, H, D))
1: $s' \leftarrow s$ ▷ Solução parcial recebe solução
2: Ordena as caixas de s' pela carga ▷ Para $k = 4$ realiza-se um embaralhamento
3: $C \leftarrow 1^o$ metade das caixas de s' ▷ Menores V_2; Maiores V_3; Aleatório V_4
4: $I \leftarrow$ Todos os itens nas caixas do conjunto C
5: **para** $a_i \in I$ **faça**
6: $P \leftarrow i$-ésimo termo da permutação de s^* ▷ Mantém a ordem original
7: Remove as caixas em C de s'
8: $s' \leftarrow$ REEMPACOTA(s', P, I, (W, H, D)) ▷ Retorna $\emptyset$ se desistir.
9: **se** $s' \neq \emptyset$ **então**
10: **se** s' é melhor que s **então** $s^* \leftarrow s'$ ▷ Nova melhor solução.
11: **retorna** s ▷ Retorna a melhor solução encontrada

instância do problema com um ótimo global composto por uma caixa que empacota todos os itens. Todas as melhorias na solução corrente teriam de ser encontradas na primeira e na quarta vizinhança. Este caso dificilmente deve acontecer fora do âmbito de testes controlados. Portanto, o objetivo desta seção é determinar a complexidade de tempo para gerar cada vizinhança relacionada com as abordagens da Tabela 4. A primeira busca local é a própria heurística proposta na Seção 3.6 detalhada no Algoritmo 3.6, e as três últimas vizinhanças são detalhadas no Algoritmo 16 compostas por uma única solução.

A complexidade de uma iteração da busca local é detalhada na Seção 3.6.1, sendo $\mathcal{O}(n^3 \times m)$ para a maioria das instâncias, onde n é a quantidade de itens a serem empacotados e m é o valor que majora o número de EM presentes em qualquer caixa aberta. A complexidade da geração das três últimas vizinhanças é dada diretamente pela análise do pior caso do Algoritmo 16, pois tratam-se de métodos que geram uma solução.

Os passos 1 e 11 de cada vizinhança são atribuições entre variáveis cujo tamanho depende da instância. Como descrito na Seção 3.6.1, a complexidade da cópia de uma solução com a representação computacional detalhada na Seção 3.1 é $\mathcal{O}(m)$ para a grande maioria das instâncias utilizando uma lista duplamente encadeada para armazenar as caixas abertas. Assim, a complexidade dos passos 1 e 11 é $\mathcal{O}(m)$.

O passo 2 para as vizinhanças V_2 e V_3 realiza a ordenação das caixas abertas da solução de entrada. No pior caso, o limite superior do problema de empacotamento é o próprio número de itens n presente na instância, onde uma solução de má qualidade pode ser construída de forma trivial através do empacotamento de um item em uma caixa distinta. Com um método de ordenação eficiente, como o *Merge Sort*, a lista duplamente encadeada pode ser ordenada com $\mathcal{O}(n \log n)$ instruções. Deste modo, a complexidade do passo 2 é $\mathcal{O}(n \log n)$. Para V_4, o passo 2 algoritmo realiza um embaralhamento das caixas abertas, que pode ser implementado com $\mathcal{O}(n)$ instruções: uma ou mais completas iterações sequenciais em toda a lista.

O passo 3 é um método que armazena o índice das caixas selecionadas para o esvaziamento no conjunto C. Utilizando o limite superior n para o pior caso, um total de $\frac{n}{2}$ índices é salvo. Então, o passo 3 tem complexidade $\mathcal{O}(n)$.

O passo 4 armazena q itens, que estavam presentes nas caixas selecionadas, dentro do conjunto I. Então, este passo realiza q iterações. O valor q é menor que n, porém, no pior caso, este valor é próximo de n. Desta maneira, o passo 4 tem complexidade $\mathcal{O}(n)$.

O passo 5 realiza uma iteração para cada item $a_i \in I$, a fim de obter a sua respectiva ordem de empacotamento da solução original. Analogamente, o número de itens presente é majorado por n. Sendo assim, o passo 5 também tem complexidade $\mathcal{O}(n)$.

O passo 7 realiza a remoção das caixas relacionadas em C da solução s' tornando-a parcialmente construída. Como citado anteriormente, no pior caso, o número de caixas em C é no máximo $\frac{n}{2}$. Então, o passo 7 é um método que realiza até $\frac{n}{2}$ repetições. Como descrito na Seção 3.6.1, a estrutura que armazena as caixas abertas de uma solução pode ser implementada através de uma lista duplamente encadeada. A remoção de uma caixa do começo da lista tem complexidade $\Theta(1)$. Como o método sempre remove a primeira metade das caixas, o passo 7 realiza $\frac{n}{2}$ remoções sequenciais do começo desta lista. Portanto, este passo tem complexidade $\mathcal{O}(n)$.

Finalmente, o método *Reempacota*, presente no passo 8, é a reimplementação da heurística construtiva presente no Algoritmo 14 para modificar seus parâmetros de entrada. Esta modificação é detalhada na Seção 3.6. A reimplementação do método tem a mesma complexidade da heurística construtiva, que por sua vez é analisada na Seção 3.5.3. Desta forma, a complexidade do passo 8 é $\mathcal{O}(n \times m^2)$ para a grande maioria das instâncias. Como a complexidade deste passo supera as outras, esta é também a complexidade final das vizinhanças V_2, V_3 e V_4. A Tabela 5 sumariza a complexidade de cada passo do Algoritmo 16 e apresenta a complexidade final das vizinhanças $V_k, k \in \{1, 2, 3, 4\}$.

Tabela 5 – Sumário da complexidade de tempo de cada passo do Algoritmo 16 em conjunto com a complexidade final das vizinhanças V_1, V_2, V_3 e V_4.

Instrução	**Passo**	**Complexidade**	**Observação**
Cópia da solução	1 e 11	$\mathcal{O}(m)$*	
Ordenação/Embaralhamento	2	$\mathcal{O}(n \log n)$ / $\mathcal{O}(n)$	Utilizando uma lista
Construção de C	3	$\mathcal{O}(n)$	duplamente encadeada
Construção de I	4	$\mathcal{O}(n)$	como auxiliar para
Construção de P	5	$\mathcal{O}(n)$	o armazenamento
Remoção de caixas	7	$\mathcal{O}(n)$	das caixas abertas
Reempacota	8	$\mathcal{O}(n \times m^2)$*	presente em uma
V_1 (Algoritmo 15)	Final	$\mathcal{O}(n^3 \times m^2)$*	solução.
V_2, V_3 e V_4 (Algoritmo 16)	Final	$\mathcal{O}(n \times m^2)$*	

*Complexidade para a grande maioria das instâncias.

3.8 **Greedy Randomized Adaptive Search Procedure**

A heurística construtiva proposta na Seção 3.5 é capaz de resolver os problemas de empacotamento. No entanto, mesmo combinando a solução construída com uma estratégia de busca local ou VND, a resposta final do método pode ficar atrelada em um determinado ótimo local de qualidade insatisfatória. Uma alternativa para evitar este problema é utilizar uma meta-heurística. Meta-heurísticas são algoritmos utilizados para a construção de heurísticas para problemas de otimização combinatória, aplicando mecanismos independentes do problema para escapar de ótimos locais. Esta seção apresenta uma extensão do algoritmo GRASP (*Greedy Randomized Adaptive Search Procedure*) proposto no trabalho (ZUDIO et al., 2017), utilizando como base a fundamentação teórica da Seção 1.3.1.

3.8.1 Fase de construção

A fase de construção do GRASP proposto é composta pela heurística adaptativa gulosa detalhada no Algoritmo 14. O algoritmo deve ser modificado, introduzindo um parâmetro de entrada adicional: o valor α, com o objetivo de introduzir aleatoriedade nas soluções obtidas a cada iteração. A heurística construtiva modificada torna-se um algoritmo semi-guloso. Os EM candidatos permanecem armazenados na lista de candidatos L, e a seleção de um candidato não é mais determinística: ao invés de manter o melhor candidato, o semi-guloso deve selecionar um dos $100\alpha\%$ melhores candidatos aleatoriamente. Esses melhores candidatos compõem o RCL (*Restricted Candidate List*). O Algoritmo 17 detalha a fase de construção do GRASP, com a modificação no parâmetro de entrada e na seleção do Algoritmo 14. O critério de escolha para determinar a qualidade de cada EM pode ser o Best Fit, Best EM ou DFTRC detalhados nas Seções 3.5.1 e 3.5.2 respectivamente.

Algoritmo 17 Fase de construção GRASP para o 3BP e o 2BP.

Requerimento: Parâmetro α no intervalo $[0, 1]$
função Semi-Guloso(**Permutação:** P; **Itens:** I; **Dimensões:** (W, H, D); **Número Real:** α)

1: $S \leftarrow \emptyset$ ▷ Solução construída começa vazia
2: $\lambda \leftarrow 0$ ▷ Número de caixas abertas começa com 0
3: **para** cada $i \in P$ **faça** ▷ Para cada item na ordem da permutação
4: $a_i \leftarrow i$-ésimo item de I ▷ Item a_i será empacotado
5: $L \leftarrow \emptyset$ ▷ Lista de EM candidatos
6: **para** cada $C \in S$ **faça** ▷ Para cada caixa aberta
7: **se** VAZIO(C) $\geq$ ESPAÇO(a_i) **então** ▷ Se o espaço vazio de C pode conter a_i
8: **para** cada EM $\in C$ **faça**
9: **se** VERIFICA(a_i, EM) **então** $L \leftarrow L \cup \{\text{EM}, C, f(\text{EM})\}$ ▷ Candidatos
10: **se** $L \neq \emptyset$ **então** ▷ Se existe EM candidato
11: EM $\leftarrow$ Um dos $100\alpha\%$ melhores de L ▷ Um candidato aleatório da RCL
12: $C \leftarrow$ Caixa do EM selecionado ▷ Referência para o empacotamento
13: **senão** ▷ Não há candidatos em L
14: $C \leftarrow$ CAIXA(W, H, D) ▷ Uma nova caixa é aberta
15: $S \leftarrow C$ ▷ Adiciona a nova caixa em S
16: $\lambda \leftarrow \lambda + 1$ ▷ Atualiza o número de caixas abertas
17: EM $\leftarrow$ ESPAÇO(C, W, H, D) ▷ EM recebe o único EM da nova caixa
18: ORIENTA(a_i, EM) ▷ Se há rotação, escolhe uma orientação aleatória para a_i.
19: EMPACOTA(a_i, C, EM) ▷ Empacota a_i através do Algoritmo 13
20: **retorna** S, λ ▷ Retorna a solução S e o número de caixas abertas

3.8.2 Fase de Melhora

A fase de melhora GRASP deve receber como parâmetro de entrada uma solução inicial construída na fase anterior, a fim de realizar uma busca por um ótimo local. Para o algoritmo proposto, podemos seguir a ideia original do GRASP, detalhada pelo Algoritmo 8, utilizando a busca local proposta na Seção 3.6 para os problemas de empacotamento. Desta maneira, a fase de melhora é o Algoritmo 15, onde a busca local pode ser implementada com o *First Improvement* ou *Best Improvement*. Como mencionado na Seção 3.7, a busca local proposta pode ficar atrelada em um determinado ótimo local ou espaço no

conjunto de soluções. Este caso se torna mais evidente quando a fase de construção recebe um valor α pequeno. Para contornar isso, um segundo método foi desenvolvido, chamado *GRASP-VND*, utilizando o VND proposto na Seção 3.7.

O GRASP-VND é uma heurística híbrida para os problemas de empacotamento que utiliza o semi-guloso proposto na fase de construção e o Algoritmo 16 na fase de melhora. Este segundo método aplica os mecanismos detalhados na Seção 3.7 para intensificar e diversificar o processo de busca. Cada um dos métodos de busca local que utilizam as vizinhanças V_1, V_2 e V_3 são executadas uma vez quando requisitadas. Entretanto, a busca local que utiliza V_4 é repetida várias vezes, escalando conforme mais iterações GRASP são efetuadas, com o objetivo de amplificar o processo de diversificação da fase de melhora. Inicialmente, a busca local com V_4 executa apenas uma vez e acumula mais uma tentativa a cada 10 iterações GRASP, tal que no máximo 15 execuções são feitas por iteração. Ou seja, as vizinhanças V_2 ou V_3 possuem sempre 1 solução, porém, no GRASP-VND, V_4 pode ter um ou mais vizinhos. Como detalhado anteriormente, os métodos de reempacotamento desistem quando identificam que a solução que está sendo construída é pior que a solução inicial de entrada. Conforme as iterações do VND prosseguem, o reempacotamento deve acelerar.

Vale a pena mencionar que este trabalho considerou a utilização de uma técnica chamada *Path-Relinking*, porém o resultado para o 3BP e o 2BP não foi satisfatório. Esta técnica intensifica o processo de busca através de ligações entre soluções de boa qualidade. Este método é um processo iterativo que obtém uma solução a cada iteração. Os parâmetros de entrada são soluções de boa qualidade. Particularmente, esta técnica é aplicada com sucesso em meta-heurísticas que recomeçam de uma nova solução a cada iteração, por exemplo, o GRASP, pois este método torna as iterações da meta-heurística dependentes (RESENDE; RIBEIRO, 2016).

3.8.3 Complexidade de Uma Iteração

Esta seção estuda o comportamento prático do GRASP proposto através da análise da complexidade de uma iteração do método. O algoritmo finaliza seu processo de busca de acordo com um critério de parada, que por sua vez é, geralmente, determinado tempo de execução em CPU. Dessa forma, dificilmente podemos prever a quantidade de operações primitivas que serão efetuadas durante este processo iterativo. Então, o foco desta seção é analisar o pior caso de cada fase do GRASP através da contagem de instruções básicas.

Uma iteração GRASP começa na fase de construção, que se trata da execução de uma variante da heurística construtiva descrita na Seção 3.5 e detalhada no Algoritmo 17. Para a grande maioria das instâncias, esta heurística tem a mesma complexidade do Algoritmo 14, a qual é sumarizada pela Tabela 2. A variante usada no GRASP altera somente o comportamento dos passos 9 e 11 relativos aos passos 10 e 11 da heurística original, introduzindo a lista de candidatos L.

O passo 9 armazena todos os possíveis candidatos em L. Como mencionado nas seções anteriores, n é o limite superior para a quantidade de caixas que uma solução pode ter. O valor m é o número que majora a quantidade de EM presentes em qualquer caixa aberta durante toda a execução da iteração GRASP. Assim como no método original, os passos 6 e 8 realizam, no máximo, n e m repetições respectivamente. Desta forma, a quantidade máxima de candidatos que L pode armazenar no pior caso é $k = n \times m$. Se L for implementado como um vetor de acesso direto, cujos elementos são uma tupla composta pela EM, sua avaliação $f(\text{EM})$ e uma referência para sua caixa, então a inserção no final de L tem complexidade constante, sendo esta a complexidade do passo 9. A cópia

de uma EM é uma operação que tem complexidade constante, devido a representação computacional deste elemento ser um conjunto de tamanho fixo de números inteiros.

No passo 11 uma ordenação deve ser feita nos elementos de L, priorizando os melhores avaliados. Logo em seguida, este passo deve selecionar um dos candidatos da RLC (um dos $100\alpha\%$ melhores). Sabendo que $k = n \times m$ é o tamanho máximo de L no pior caso, a implementação da ordenação pode ser feita através de um algoritmo eficiente com complexidade $\mathcal{O}(k \times \log k)$, por exemplo, o *Merge Sort*. Como o acesso em L tem complexidade constante, a complexidade do passo 11 é a própria ordenação, sendo $\mathcal{O}(nm \times \log nm)$.

Nesta variante, a complexidade do passo 11 é maior que a complexidade dos passos 6 e 8. Assim, a complexidade de uma iteração do semi-guloso é $\mathcal{O}(n \times \max(nm \times \log nm, m^2))$ dado pelo passos 3, 11 e 19. O valor de $nm \times \log nm$ é maior que m^2 em certos casos especiais detalhados na Seção 3.5.3, dentre os quais se tratam da minoria das instâncias. Então, a complexidade da fase de construção do GRASP é $\mathcal{O}(n \times m^2)$ para a grande maioria das instâncias.

A fase de melhora é composta pela busca local ou o VND proposto nas Seções 3.6 e 3.7 respectivamente. A complexidade de uma iteração da busca local é detalhada na Seção 3.6.1 e sumarizada pela Tabela 3. A tabela mostra que a complexidade final de uma iteração da busca local para a grande maioria das instâncias é $\mathcal{O}(n^3 \times m^2)$. Portanto, esta é a complexidade da fase de melhora caso esta opção seja utilizada. O comportamento prático e a complexidade do VND são detalhados na Seção 3.7.1, onde a Tabela 5 sumariza a complexidade de cada vizinhança proposta. Como as complexidades da busca local e do VND superam a da fase de construção, a complexidade de uma iteração GRASP é dada pela fase de melhora. A Tabela 6 sumariza a complexidade de cada componente das versões GRASP propostas nesta seção em conjunto com a complexidade final de uma iteração, relacionando o componente com a fase em que é aplicado e sua complexidade para a grande maioria das instâncias.

Tabela 6 – Sumário da complexidade dos componentes de cada GRASP proposto em conjunto com a complexidade final de uma iteração.

Componente	**Fase**	**Complexidade**
Heurística construtiva (Algoritmo 13)	Construção	$\mathcal{O}(n \times m^2)$*
Busca Local (Algoritmo 15)	Melhora	$\mathcal{O}(n^3 \times m^2)$*
VND (Algoritmo 3.7)	Melhora	$\mathcal{O}(n^3 \times m^2)/\mathcal{O}(n \times m^2)$**
1 iteração GRASP ou GRASP-VND	Final	$\mathcal{O}(n^3 \times m^2)$*

*Complexidade para a grande maioria das instâncias.
**Complexidade das vizinhanças utilizadas.

3.9 **Busca Tabu**

O método GRASP apresentado na seção anterior é um algoritmo que constrói uma solução aleatória a cada iteração. A heurística recomeça de uma nova configuração de empacotamento, a fim de melhora-lá posteriormente. A **Busca Tabu** (BT) é uma meta-heurística que não utiliza um mecanismo de recomeço, e sim constrói uma solução para melhora-lá durante suas iterações até o critério de parada ser atingido. As iterações do BT aplicam um mecanismo para escapar de ótimos locais independente de qualquer gerador de números pseudoaleatórios. A ideia principal desta meta-heurística é utilizar uma memória

adaptativa para guiar o processo de busca por soluções, de maneira que as iterações não fiquem atreladas a um ótimo local ou em um determinado local do espaço de soluções. Nos problemas de empacotamento, o BT proposto melhora a solução construída através do reempacotamento de itens, assim como o método proposto na seção anterior. O algoritmo GRASP constrói uma nova solução através do empacotamento completo de itens a cada iteração. Sabemos que o reempacotamento de itens exige uma quantidade menor de tempo de processamento. Assim, as iterações da BT tem o potencial de serem mais rápidas, pois são constituídas somente por métodos de reempacotamento. Consequentemente, a busca tabu tem um potencial maior de avaliar uma grande quantidade de soluções. O foco desta seção é detalhar um método busca tabu para o 3BP e o 2BP utilizando o fundamento teórico da Seção 1.3.2.

O BT proposto para os problemas de empacotamento segue a mesma ideia da busca local detalhada no Algoritmo 3.6, isto é, esvaziar um par de caixas da solução corrente para efetuar um reempacotamento. O primeiro parâmetro de entrada é uma solução inicial s_0. A heurística construtiva proposta na Seção 3.5 é utilizada para gerar esta solução, sendo que o critério de avaliação de um EM pode ser o Best Fit, Best EM ou DFTRC detalhados nas Seções 3.5.1 e 3.5.2. O segundo parâmetro de entrada no BT original, detalhado no Algoritmo 9, é a estrutura de vizinhança. No método proposto, este parâmetro não é utilizado, pois as soluções que constituem a vizinhança são as próprias geradas pelo processo de reempacotamento. O segundo parâmetro do BT é o valor T_{max}, que se refere ao tamanho máximo da lista Tabu T. Este parâmetro deve ser calibrado, sendo o único parâmetro calibrável da heurística. O processo iterativo de BT consiste em esvaziar as caixas par a par, ignorando as com 100% de carga e selecionando a melhor solução obtida no reempacotamento através do mesmo método detalhado na Seção 3.6: o Algoritmo 14 é parcialmente modificado para receber uma solução que já possui caixas abertas e um subconjuntos de itens a serem reempacotados nela. O processo de reempacotamento utilizado em todos os métodos anteriores desiste imediatamente caso seja identificado que a solução que está sendo construída terá uma qualidade pior do que a solução atual. Já o BT deve aceitar a melhor solução não tabu encontrada na vizinhança, mesmo que ela seja pior que a solução atual. O método de reempacotamento utilizado no BT desiste de construir uma solução somente quando é identificado que a mesma já é pior que um vizinho não tabu encontrado anteriormente. Assim, um novo parâmetro λ_{max} de entrada é utilizado. O método de empacotamento deve desistir quando for identificado que o valor da função objetivo da solução parcial é maior que λ_{max}.

A primeira estratégia utilizada para a implementação do tabu T foi manter o registro dos movimentos, onde, caso a solução final da iteração tenha sido obtida através do esvaziamento das caixas C_i e C_j, $1 < i \neq j \leq n$, então o par (i, j) que compõe este movimento era considerado tabu. Entretanto, os resultados de testes empíricos com essa abordagem não foram satisfatórios. A lista tabu do algoritmo proposto usa uma segunda estratégia: manter as próprias soluções finais obtidas durante as iterações. Ou seja, quando uma iteração determina sua solução final, ela é inserida na lista tabu. Caso a lista esteja cheia, então o tabu mais antigo é removido. Pode acontecer um caso raro onde todas as soluções geradas com sucesso são tabu. Desta maneira, a melhor solução tabu gerada atende o critério de aspiração, assim, ela é removida da lista e selecionada como final da iteração atual. Dessa forma, o BT é uma heurística que encontra diversas soluções distintas e raramente suas iterações selecionam uma final que utiliza um número superior de caixas do que a solução corrente, pois o mecanismo de esvaziamento proposto combinado com o reempacotamento é capaz de gerar diversas soluções distintas com o

mesmo número de caixas. A ideia é realizar o reempacotamento de itens para obter soluções com o mesmo número de caixas abertas até que uma iteração consiga efetuar uma melhora na configuração de até duas caixas.

Desta maneira, para o tabu, duas soluções são consideradas iguais se possuem o mesmo número de caixas com as mesmas configurações. Assim, para $1 \leq i, j \leq n$ não necessariamente distintos, duas caixas C_i e C_j de soluções diferentes são consideradas iguais se ambas têm a mesma quantidade q de itens empacotados. Além disso, para cada item $a \in C_i$ existe um respectivo item $a' \in C_j$ com as mesmas dimensões, tal que formemos q pares de itens semelhantes. Ou seja, as caixas são iguais se os itens empacotados em cada uma delas são iguais par a par independente de sua localização dentro de suas respectivas caixas. O Algoritmo 18 detalha o BT proposto, sendo que o reempacotamento pode utilizar o Best Fit, o Best EM ou o DFTRC.

O BT utiliza a mesma ideia da busca local detalhada na Seção 3.6. Como descrito na Seção 3.7, esta estratégia pode ficar atrelada em um determinado subespaço do espaço de soluções viáveis mesmo com o auxílio da memória adaptativa. Assim, uma segunda versão desta heurística foi desenvolvida denominada BT-VN (*Busca Tabu Variable Neighborhood*). O BT-VN é um algoritmo que utiliza as vizinhanças V_2, V_3 e V_4 detalhadas pelo Algoritmo 3.7 e propostas no VND apresentado na Seção 3.7, a fim de intensificar e diversificar o processo de busca. O método utiliza cada uma das três vizinhanças no inicio da iteração atual antes de começar o processo de esvaziar caixas par a par. Desta maneira, cada iteração realiza três reempacotados adicionais para gerar soluções distintas que intensificam e diversificam o processo de busca. A solução escolhida para ser submetida à vizinhança V_1 é a melhor solução entre a corrente e as geradas por V_2, V_3 e V_4. Portanto, o reempacotamento utilizado para gerar a única solução que compõe V_2, V_3 ou V_4 pode desistir imediatamente caso identifique que esta vai ser pior que alguma anterior. Além disso, no final da iteração, se uma solução s atende o mesmo critério de aspiração do BT, o BT-VN executa V_4 com s como parâmetro de entrada, aceitando a solução vizinha gerada independente de sua qualidade. Assim, a lista tabu é imediatamente atualizada acrescentando a nova solução gerada normalmente, porém, diferentemente da versão anterior, s não é removido. A finalidade deste último mecanismo é garantir que o método não fica atrelado em um determinado local do espaço de busca. O Algoritmo 19 detalha o BT-VN de acordo com as modificações descritas.

3.9.1 Complexidade de Uma Iteração

A busca tabu original executa seu processo iterativo até que o critério de parada seja atingido. As versões propostas nesta seção para o 3BP e o 2BP não mudam esse comportamento. A complexidade do BT e do BT-VN também depende do critério de parada utilizado. Geralmente, este critério é atingir um determinado tempo de execução. Assim, é difícil prever o comportamento prático desses algoritmos. Então, esta seção descreve a complexidade de tempo dos componentes que sintetizam os Algoritmos 18 e 19, a fim de detalhar o comportamento prático desses métodos e de determinar a complexidade de uma iteração de cada algoritmo.

Ambas as versões do algoritmo requerem que uma solução inicial qualquer seja construída. Uma maneira fácil de realizar este processo é empacotar cada item em uma caixa distinta. A qualidade desta solução trivial, em geral, é muito ruim comparada com a qualidade das soluções geradas pela a heurística construtiva da Seção 3.5. Desta maneira, o Algoritmo 14 deve receber uma permutação arbitrária para sintetizar a solução inicial, podendo utilizar o Best Fit, Best EM ou o DFTRC. Então, como descrito na Seção 3.5.3,

Algoritmo 18 Busca Tabu BT para o 3BP e o 2BP

Observação: a estrutura de vinhança V constituí-se das soluções geradas pelo método *Reempacota*, portanto, não é passada como parâmetro de entrada.
função BT(**Solução inicial:** s_0; **Tamanho máximo do Tabu:** T_{max})

1: $s^*, s \leftarrow s_0$ ▷ Melhor solução s^* e solução corrente s recebem solução inicial
2: $T \leftarrow \emptyset$ ▷ Lista tabu começa vazia
3: **enquanto** critério de parada não satisfeito **faça**
4: $s_V^* \leftarrow \emptyset$ ▷ Melhor vizinho não tabu
5: $\lambda_{max} \leftarrow \infty$ ▷ λ_{max} guarda a avaliação de s_V^*
6: $\lambda \leftarrow$ Número de caixas de s ▷ Caixas abertas da solução atual
7: **para** i de 1 até $\lambda - 1$ **faça**
8: $C \leftarrow$ Caixa C_i de s ▷ i-ésima caixa da solução atual
9: **se** C estiver com 100% de carga **então** Avançar o *loop* ▷ Ignora caixas promissoras
10: **para** j de $i + 1$ até λ **faça**
11: $C' \leftarrow$ Caixa C_j de s ▷ j-ésima caixa da solução atual
12: **se** C' estiver com 100% de carga **então** Avançar o *loop* ▷ Ignora caixas promissoras
13: $s' \leftarrow s$ ▷ Solução parcial recebe solução atual
14: $I \leftarrow$ Todos os itens de C e C' ▷ Conjunto de itens para reempacotar
15: **para** $a_k \in I$ **faça**
16: $P \leftarrow k$-ésimo termo da permutação de s^* ▷ Mantém a ordem original
17: Apagar as caixas C_i e C_j da solução s'
18: $s' \leftarrow$ REEMPACOTA(s', P, I, (W, H, D), λ_{max}) ▷ Retorna $\emptyset$ se desistir
19: **se** $s' \neq \emptyset$ **então**
20: **se** $s' \notin T$ **então** ▷ Se s' não é uma solução tabu
21: **se** s' é melhor que s_V^* **então**
22: $s_V^* \leftarrow s'$ ▷ Novo melhor vizinho
23: $\lambda_{max} \leftarrow f(s_V^*)$ ▷ Avaliação de s_V^*.
24: **senão** Marca s' em T ▷ Sinaliza que este tabu foi encontrado nesta iteração
25: **se** $s_V^* = \emptyset$ **então** ▷ Se não há melhor vizinho
26: $s_V^* \leftarrow$ Melhor tabu em T marcado nesta iteração ▷ Critério de aspiração
27: ATUALIZA(T) ▷ Remove s_V^* da lista tabu
28: **se** s_V^* é melhor que s^* **então** $s^* \leftarrow s_V^*$ ▷ Nova melhor solução
29: $s \leftarrow s_V^*$ ▷ Solução corrente recebe o melhor vizinho não tabu
30: ATUALIZA(T, T_{max}, s_V^*) ▷ Atualiza o tabu de acordo com T_{max} e s_V^*
31: **retorna** s^* ▷ Retorna a melhor solução encontrada

este passo externo do método tem complexidade $\mathcal{O}(n \times m^2)$ para a grande maioria das instâncias, tal que n é o número de itens e m é o valor que majora a quantidade de EM presentes durante toda a execução do BT ou do BT-VN em qualquer caixa aberta.

Seguindo o Algoritmo 18 como referência, os passos 1, 13, 22, 28 e 29 realizam atribuições entre variáveis que armazenam soluções cujo tamanho varia de acordo com o tamanho da entrada, ou seja, a complexidade destes passos não é constante. Como mencionado anteriormente na Seção 3.6.1, a complexidade da cópia de uma solução é $\mathcal{O}(m)$ para a grande maioria das instâncias. Portanto, esta é também complexidade dessas atribuições. O passo 7 é uma cláusula de repetição que itera nas caixas abertas da solução corrente assim como o passo 10. A finalidade destes passos é esvaziar um par de caixas. O método mencionado anteriormente para construir uma solução trivial para os problemas de empacotamento nos fornece o limite superior n. Desse modo, a quantidade de repetições que ambos os passos realizam é majorado pelo parâmetro n. Portanto, os passos 7 e 10 têm complexidade $\mathcal{O}(n)$. As análises de complexidade anteriores assumem que a estrutura

Algoritmo 19 Busca Tabu BT-VN para o 3BP e o 2BP

função BT–VN(**Solução inicial:** s_0; **Tamanho máximo do Tabu:** T_{max})
1: $s^*, s \leftarrow s_0$ ▷ Melhor solução s^* e solução corrente s recebem solução inicial
2: $T \leftarrow \emptyset$ ▷ Lista tabu começa vazia
3: **enquanto** critério de parada não satisfeito **faça**
4: $s_V^* \leftarrow \emptyset$ ▷ Melhor vizinho não tabu
5: $\lambda_{max} \leftarrow \infty$ ▷ λ_{max} guarda a avaliação de s_V^*
6: **para** $k \in \{2, 3, 4\}$ **faça** $s \leftarrow V_k(s)$ ▷ Aplica o Algoritmo 16 mantendo o melhor
7: $\lambda \leftarrow$ Número de caixas de s ▷ Caixas abertas da solução atual
8: **para** i de 1 até $\lambda - 1$ **faça**
9: $C \leftarrow$ Caixa C_i de s ▷ i-ésima caixa da solução atual
10: **se** C estiver com 100% de carga **então** Avançar o *loop* ▷ Ignora caixas promissoras
11: **para** j de $i + 1$ até λ **faça**
12: $C' \leftarrow$ Caixa C_j de s ▷ j-ésima caixa da solução atual
13: **se** C' estiver com 100% de carga **então** Avançar o *loop* ▷ Ignora caixas promissoras
14: $s' \leftarrow s$ ▷ Solução parcial recebe solução atual
15: $I \leftarrow$ Todos os itens de C e C' ▷ Conjunto de itens para reempacotar
16: **para** $a_k \in I$ **faça**
17: $P \leftarrow k$-ésimo termo da permutação de s^* ▷ Mantém a ordem original
18: Apagar as caixas C_i e C_j da solução s'
19: $s' \leftarrow$ REEMPACOTA(s', P, I, (W, H, D), λ_{max}) ▷ Retorna $\emptyset$ se desistir
20: **se** $s' \neq \emptyset$ **então**
21: **se** $s' \notin T$ **então** ▷ Se s' não é uma solução tabu
22: **se** s' é melhor que s_V^* **então**
23: $s_V^* \leftarrow s'$ ▷ Novo melhor vizinho
24: $\lambda_{max} \leftarrow f(s_V^*)$ ▷ Avaliação de s_V^*.
25: **senão** Marca s' em T ▷ Sinaliza que este tabu foi encontrado nesta iteração
26: **se** $s_V^* = \emptyset$ **então** ▷ Se não há melhor vizinho
27: $s_V^* \leftarrow$ Melhor tabu em T marcado nesta iteração ▷ Critério de aspiração
28: $s_V^* \leftarrow V_4(s_V^*)$ ▷ Sempre aceita a solução gerada e nunca desiste
29: **se** s_V^* é melhor que s^* **então** $s^* \leftarrow s_V^*$ ▷ Nova melhor solução
30: $s \leftarrow s_V^*$ ▷ Solução corrente recebe o melhor vizinho não tabu
31: ATUALIZA(T, T_{max}, s_V^*) ▷ Atualiza o tabu de acordo com T_{max} e s_V^*
32: **retorna** s^* ▷ Retorna a melhor solução encontrada

que armazena as caixas abertas de uma solução é uma lista duplamente encadeada. Os passos 8 e 11 podem ser implementados com variáveis que recebem uma referência para as caixas abertas C_i e C_j. Desta maneira, a complexidade dos passos 8 e 11 é constante. Analogamente, a verificação feita nos passos 9 e 12 pode ser implementada também em tempo constante através de uma variável auxiliar que armazena a carga da caixa em sua representação computacional.

O passo 14 sintetiza o conjunto I com os itens presentes nas caixas C_i e C_j. Este passo utiliza uma cláusula de repetição igual ao passo 14 do Algoritmo 15, que tem sua complexidade sumarizada na Tabela 3. A complexidade para a construção de I é $\mathcal{O}(n)$. Desta maneira, o passo 15 é uma cláusula de repetição que realiza no máximo n iterações, pois este passo itera sobre os itens em I. O passo 17 remove as caixas C_i e C_j da solução corrente, como o acesso é sequencial e a remoção de um elemento referenciado em uma lista duplamente encadeada é $\Theta(1)$, então a complexidade do passo 17 também é constante. O passo 18 é uma reimplementação do Algoritmo 14 que possui parâmetros de entrada

adicionais, com o objetivo de realizar o reempacotamento com uma solução parcialmente construída e com um alvo máximo de qualidade para a desistência. Como descrito na Seção 3.6, esta modificação não altera a complexidade original da heurística construtiva, portanto, o passo 18 tem complexidade $\mathcal{O}(n \times m^2)$ para a maioria das instâncias.

O passo 20 verifica se a solução s' é tabu. A comparação entre duas soluções não tem complexidade constante, pois usam variáveis cujo tamanho depende da quantidade de itens na instância. Como detalhado na seção anterior, duas soluções são iguais se todas suas caixas são iguais, e duas caixas são iguais se empacotam os mesmos itens independente de sua localização dentro da mesma. O número máximo de caixas que uma solução pode ter é majorado pelo parâmetro n, que é o limite superior do 3BP e do 2BP. Então, a comparação de duas soluções s e s' pode ser implementada através de uma cláusula de repetição que itera sequencialmente em cada caixa C de s, verificando se cada item $a_i \in C$ está todo empacotado em uma mesma caixa em s'. Além disso, nenhum outro item além de cada a_i pode estar nesta mesma caixa. De acordo com a representação computacional da Seção 3.1. Este passo pode ser feito através de uma segunda cláusula de repetição sobre o vetor que armazena cada λ_i, que por sua vez é o parâmetro que identifica o número da caixa em que o item i está empacotado, $1 \leq i \leq n$. Portanto, a comparação entre duas soluções tem complexidade $\mathcal{O}(n^2)$. Assim, o passo 20 é composto por uma cláusula de repetição que realiza no máximo T_{max} repetições, uma comparação com cada solução tabu. Consequentemente, a complexidade deste passo é $\mathcal{O}(T_{max} \times n^2)$.

O passo 24 é executado quando é identificado que a solução construída é tabu. Este passo marca a solução atual para sinalizar que ela foi encontrada nesta iteração. A marcação é feita com uma variável auxiliar relacionada à lista tabu que deve armazenar o número da iteração corrente da busca tabu em que ela foi encontrada. O passo 24 pode ser implementado em conjunto com a iteração realizada no passo 20. Portanto, este passo tem complexidade constante. O passo 26 é executado quando nenhum vizinho foi gerado com sucesso, isto é, todos os vizinhos são tabu. O passo 26 consiste de uma cláusula de repetição que deve iterar sobre cada solução tabu para selecionar a melhor que possui dentre as marcadas nesta iteração no passo 24. Então, o passo 27 realiza no máximo T_{max} repetições. O passo 27 remove da lista tabu a solução selecionada no passo anterior. A estrutura que armazena as soluções na lista tabu pode ser implementada como uma lista duplamente encadeada sem afetar as complexidades estabelecidas anteriormente, pois os passos que acessam esta lista realizam iterações sequenciais. Assim, esta remoção pode ser efetuada com complexidade constante, pois a referência para o elemento que deve ser removido pode ser obtido no passo 26. O passo 30 atualiza a lista tabu T de acordo com a solução final s_V^* estabelecida na iteração. Este passo deve inserir o novo tabu s_V^* na lista, posteriormente, checar se T está cheio de acordo com T_{max}. Quando T está cheio, a solução mais antiga deve ser removida. Se a inserção de um novo tabu for sempre realizada no final da lista duplamente encadeada, a complexidade inserção é $\Theta(1)$. Portanto, a complexidade da remoção também é constante, pois o tabu mais antigo é o primeiro elemento. Assim, a estrutura que armazena as soluções é uma fila especial, pois o critério de aspiração permite que um tabu no meio da fila seja removido. Então, assumindo que nenhuma cópia de soluções é feita durante o processo de atualização, a complexidade do passo 30 é $\Theta(1)$. Observe que uma cópia já é feita no passo anterior, com o objetivo armazenar a nova solução corrente para a iteração seguinte.

Para o Algoritmo 19, a maior parte dos passos são os mesmos que os realizados pelo BT, sendo de mesma complexidade. A principal modificação está presente nos passos 6 e 28. O passo 6 utiliza as vizinhanças V_2, V_3 e V_4 em sequência, onde o Algoritmo 16

detalha o processo de cada uma delas. A complexidade destes métodos é detalhada na Seção 3.7.1 e sumarizada na Tabela 5, sendo todas $\mathcal{O}(n \times m^2)$ para a maioria das instâncias. Finalmente, o passo 28 executa somente V_4, e também tem complexidade $\mathcal{O}(n \times m^2)$ para a maioria das instâncias. Assim, a complexidade de uma única iteração do BT e do BT/VN é dado pelas cláusulas de repetição que iteram nas caixas da solução corrente em conjunto com a função *Reempacota*, ou seja, os passos 7, 10 e 18 para o BT e 8, 11 e 19 para o BT/VN. Portanto, a complexidade de uma iteração de ambos os métodos para a grande maioria das instâncias é $\mathcal{O}(n^3 \times m^2)$. A Tabela 7 sumariza a complexidade dos componentes que compõem uma iteração de cada busca tabu proposta, relacionando o passo de seu respectivo método com sua complexidade e a instrução realizada.

Tabela 7 – Sumário da complexidade dos componentes de cada busca tabu proposta.

Instrução	**Passo**		**Complexidade**	**Observação**
	BT	**BT/VN**		
Solução inicial (Algoritmo 14)	Externo		$\mathcal{O}(n \times m^2)$*	
Cópia da solução	1, 13, 22, 28 e 29	1, 14, 23, 29 e 30	$\mathcal{O}(m)$*	
1^o e 2^o *Loop*	7 e 10	8 e 11	$\mathcal{O}(n)$	Utilizando
Armazenar C_i e C_j	8 e 11	9 e 12	$\Theta(1)$	uma lista
1^o e 2^o verificação	9 e 12	10 e 13	$\Theta(1)$	duplamente
Construir I	14	15	$\mathcal{O}(n)$	encadeada
3^o *Loop*	15	16	$\mathcal{O}(n)$	como
Apagar par de caixas	17	18	$\Theta(1)$	estrutura
Reempacota	18	19	$\mathcal{O}(n \times m^2)$*	auxiliar
Verificar tabu	20	21	$\mathcal{O}(T_{max} \times n^2)$	para
Marcar solução	24	25	$\Theta(1)$	armazenar
Selecionar tabu	26	27	$\mathcal{O}(T_{max})$	as soluções
Remover tabu	27		$\Theta(1)$	tabu.
Atualizar T	30	31	$\Theta(1)$	
V_2, V_3 e V_4		6 e 28	$\mathcal{O}(n \times m^2)$*	
1 iteração	Final	Final	$\mathcal{O}(n^3 \times m^2)$*	

*Complexidade para a grande maioria das instâncias.

3.10 **Algoritmo Genético de Chaves Aleatórias Viciadas**

Como descrito na Seção 3.6, estratégias de busca local para o 3BP e o 2BP que envolvem a movimentação de itens, seja entre diferentes caixas ou dentro de si próprias, dificilmente são capazes de melhorar a qualidade das soluções construídas por uma heurística construtiva que realiza o empacotamento aplicando um mecanismo equivalente. Por exemplo, uma busca local que movimenta itens através de diferentes EM em qualquer caixa aberta de uma solução viável dificilmente tem o potencial de melhorar uma configuração obtida através da heurística construtiva apresentada na Seção 3.5. Uma alternativa é utilizar um processo que esvazia caixas, parcial ou completamente, da solução corrente, para realizar um reempacotamento através de uma heurística construtiva. Na prática, o potencial de melhora das soluções construídas pode, ainda assim, ser baixo para algumas instâncias. A utilização de uma busca local nos problemas de empacotamento pode provocar, em alguns casos, outros fenômenos que impactam negativamente na eficiência

do processo de busca. Por exemplo, a busca ficar atrelada em um determinado ponto do espaço de soluções com um alto tempo de execução sem melhora.

Esta seção apresenta um método cuja estratégia é realizar a construção desde o princípio de múltiplas soluções em uma mesma iteração através de uma variante da heurística construtiva detalhada pelo Algoritmo 14. O procedimento independe de qualquer fase de melhora que necessita de algum tipo de busca local. A abordagem é um Algoritmo Evolucionário que a cada iteração constrói novas soluções desde o princípio, melhorando a qualidade média do conjunto de soluções conforme mais iterações são realizadas.

Os algoritmos propostos, denominados BRKGA e BRKGA-VCD, são uma extensão do método apresentado no trabalho (ZUDIO et al., 2018), que por sua vez é uma extensão do método BRKGA proposto em (GONÇALVES; RESENDE, 2013) para o 3BP e o 2BP. O BRKGA-VCD é uma heurística híbrida sintetizada através do Algoritmo Genético de Chaves Aleatórias Viciadas ou *Biased Random-Keys Genetic Algorithm* (BRKGA) combinada com uma nova técnica chamada *Variable Cross Descent* (VCD) proposta originalmente no primeiro trabalho citado. O VCD será detalhado na seção seguinte. A fundamentação teórica desta seção são todos os conceitos detalhados nas Seções 1.3.3 e 1.3.4 para algoritmos evolucionários e o próprio BRKGA.

O BRKGA e a variação BRKGA-VCD para o 3BP e o 2BP utilizam o Algoritmo 11, que descreve os passos realizados pela meta-heurística BRKGA original. Para os problemas de empacotamento, os parâmetros de entrada que fornecem a lista dos itens a serem empacotados e as dimensões da caixa devem ser adicionados ao algoritmo. Além destes, os parâmetros de entrada originais da meta-heurística que devem ser calibrados são definidos da seguinte maneira:

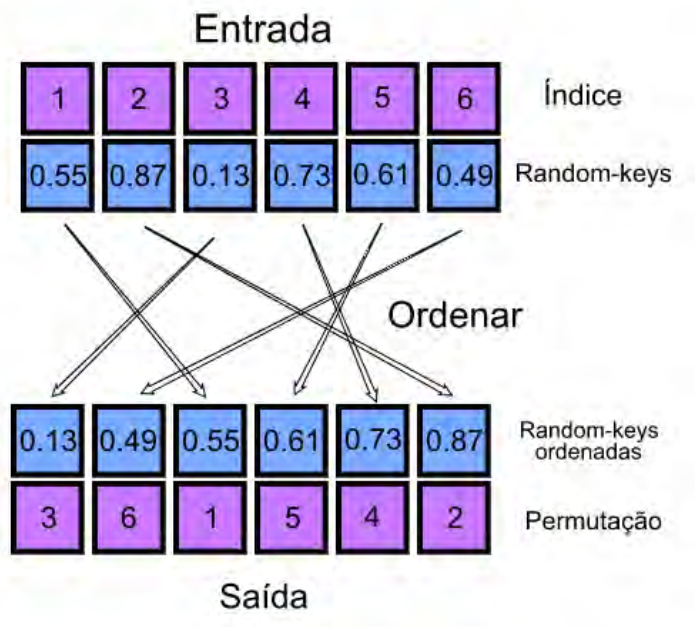
- Quantidade de indivíduos P_{max}: é o número de indivíduos presente em cada geração do método e no conjunto de soluções iniciais.
- Fração TOP ϵ: um número real no intervalo de $]0, 1[$ que determina a porção Elite de uma população. Os $100\epsilon\%$ melhores indivíduos são classificados como Elite e o restante é classificado como não-Elite.
- Fração BOT ω: é um número real no intervalo de $[0, \frac{1}{2}]$ que determina a quantidade de mutantes que são introduzidas na população em cada iteração. O processo de mutação introduz $100\omega\%$ mutantes na população atual.
- Taxa de reprodução ρ: é um número real no intervalo $[\frac{1}{2}, 1[$ que determina a probabilidade de favorecimento na escolha do gene do parente Elite em relação ao parente não-Elite no processo de reprodução. Para cada gene do filho, temos $100\rho\%$ de chance de copiar o respectivo gene do seu parente Elite.

Para o 3BP e o 2BP, um indivíduo é representado por um vetor com n números reais no intervalo $[0, 1]$, onde n é a quantidade de itens presentes na instância. Cada elemento deste vetor recebe o nome de chave aleatória ou *random-key*. A cada iteração uma nova população P' é sintetizada a partir da população P que compõe a geração anterior. O primeiro processo de uma iteração é a seleção, presente no passo 4 do Algoritmo 11. Este processo permanece inalterado, isto é, a seleção consiste em copiar os $100\epsilon\%$ indivíduos de P para P'. Posteriormente, o processo de mutação e reprodução é realizado, porém, somente a reprodução permanece inalterada para o método proposto. A mutação consiste em introduzir $100\omega\%$ mutantes em P' através do mesmo método utilizado para gerar a população inicial. Tal método de geração difere do BRKGA e será detalhado na seção

seguinte para o BRKGA-VCD. A reprodução consiste em selecionar aleatoriamente dois parentes de P', sendo um Elite e um não-Elite e, então, sintetizar um filho, tal que cada gene tenha $100\rho\%$ de probabilidade de ser uma cópia do respectivo gene do parente Elite. Caso contrário será uma cópia do respectivo gene do parente não-Elite. Finalmente, os indivíduos que ainda não foram decodificados da geração atual são decodificados e ranqueados de acordo com sua aptidão, no passo 7, pelo procedimento *Decodifica_Avalia*. Este procedimento utiliza o decodificador para avaliar a aptidão dos novos indivíduos introduzidos na geração atual. O decodificador é único componente do BRKGA que dependente do problema que está sendo resolvido.

Para os problemas de empacotamento, o decodificador deve receber um indivíduo como parâmetro de entrada para sintetizar uma solução viável com todos os itens empacotados, ou seja, o parâmetro de entrada é um vetor com n *random-keys* e a saída deve ser uma configuração válida com todos os itens empacotados sem sobreposição. A solução deve ser construída desde o princípio com uma variante da heurística construtiva detalhada pelo Algoritmo 14 que será descrita posteriormente nesta seção. Este componente deve formar uma permutação com n números para representar a sequência de itens a serem empacotados. O processo de decodificação utiliza a mesma ideia apresentada por (BEAN, 1994) para problemas de sequenciamento. A ideia consiste em associar cada *random-key* com um inteiro positivo distinto relacionado à sua posição. Ou seja, a primeira *random-key* é associada ao índice 1, a segunda ao índice 2 e assim por diante. Posteriormente, uma ordenação é realizada no vetor de *random-keys*, tal que a permutação é obtida com os respectivos índices associados anteriormente que devem ser ordenados em conjunto com as *random-keys*. Esta operação define uma função que associa um conjunto de números reais a uma permutação. Para exemplificar este processo, a Figura 17 apresenta um exemplo de decodificação com um indivíduo codificado através de 6 *random-keys*. A figura apresenta a ordenação das *random-keys* do indivíduo em conjunto com o seu respectivo índice, o que nos fornece a permutação para representar a sequência dos itens para o empacotamento.

Figura 17 – Exemplo do processo de ordenação efetuado pelo decodificador com $n = 6$.



Como mencionado anteriormente, o empacotamento dos itens é realizado através de

uma variante da heurística construtiva. Sabendo que a construção de uma solução é um processo com alto custo computacional, a finalidade desta variante é construir soluções com um tempo menor de execução comparado à heurística construtiva original, pois o BRKGA precisa sintetizar diversas soluções em uma mesma iteração. A heurística original empacota os itens sequencialmente buscando pelo melhor EM presente em qualquer caixa aberta de acordo com um critério de avaliação, e quando não existe uma caixa que pode receber o item, uma nova caixa é aberta. A nova implementação é uma modificação no número de caixas e EM a serem avaliados para efetuar o empacotamento. Desta forma, a estratégia da variante e os parâmetros de entrada originais permanecem. A variante deve decidir qual é o melhor EM somente da **primeira** caixa que pode alocar o item através dos mesmos critérios de avaliação utilizados no processo original. Quando encontramos uma caixa aberta que pode receber o item e identificamos que há pelo menos um EM válido para o empacotamento, então escolhemos o melhor EM válido desta caixa sem proceder à busca pelo melhor EM de toda a configuração. Desta maneira, a variante apresenta um potencial de economia no tempo de execução. A qualidade da solução com uma mesma permutação de entrada pode ser menor nesta variante comparada ao original, porém, as melhores configurações possíveis sempre podem ser obtidas com permutações específicas.

O decodificador é um componente determinístico. Quando utilizamos o critério Best Fit ou Best EM para a avaliação de um EM, a heurística construtiva pode gerar diferentes soluções para uma mesma permutação, devido à escolha aleatória do canto de inserção (CI) em caso de empate durante a construção do EM. O método que constrói cada EM deve manter uma lista auxiliar A com os k cantos candidatos para alocar o item. Um indivíduo no BRKGA deve conter um conjunto adicional de n *random-keys*, uma *key* nova para cada item na instância, para tornar o processo determinístico. Cada indivíduo deve possuir n *random-keys* para codificar sua permutação e n outras adicionais para desempatar o critério de escolha do melhor EM quando utilizamos o Best Fit ou Best EM. O novo conjunto de *random-keys* deve ser passado como parâmetro de entrada para a heurística construtiva. Ele deve ser utilizado para escolher um dos k cantos candidatos $A_i \in A$ através da Equação 3.2, tal que $1 \leq i \leq k$ é o índice do CI escolhido dentro da lista A e $0 < r \leq 1$ a respectiva *random-key* adicional associada ao item. Durante a reprodução, o mesmo processo aplicado no primeiro conjunto de *random-keys* também é aplicado neste novo conjunto sem alterações. Isto é, este novo conjunto também faz parte do cromossomo de cada parente.

$$i = \lceil k \times r \rceil \quad | \quad A_i \in A \tag{3.2}$$

Quando o problema permite a rotação dos itens, a heurística construtiva original escolhe uma orientação aleatoriamente dentre as possíveis. Portanto, este processo também deve ser modificado para tornar a variante utilizada no BRKGA e no BRKGA-VCD determinística. Analogamente, um novo conjunto com n *random-keys* adicionais deve ser utilizado de maneira semelhante ao processo descrito anteriormente para o desempate durante a escolha do CI de um EM. Com este novo conjunto, que também deve ser passado como parâmetro de entrada para a heurística construtiva, uma nova *random-key* é associada para cada item da instância. Assim, podemos utilizar a Equação 3.2 para escolher a orientação, considerando que desta vez A é a lista das possíveis orientações do item e r a sua respectiva *random-key* neste conjunto adicional referente à sua orientação. Assim como o primeiro conjunto de *random-keys* que determina a ordem de empacotamento dos itens e o conjunto adicional para o critério de desempate do CI, este novo conjunto adicional associado à orientação do item passa a fazer parte do cromossomo do indivíduo.

O Algoritmo 20 detalha a variante descrita da heurística construtiva. A lista L_1 deve ser utilizada somente como parâmetro de entrada caso o critério de escolha seja o Best Fit ou Best EM e a lista L_2 somente quando há rotação. Finalmente, o Algoritmo 21 apresenta o decodificador utilizado no BRKGA e no BRKGA-VCD, onde a função *Empacota* é o Algoritmo 20, podendo utilizar o Best Fit, Best EM ou o DFTRC como critério de avaliação para as EM.

Algoritmo 20 Heurística construtiva de empacotamento para o BRKGA

Observação: L_1 só é passado caso o Best Fit ou Best EM seja utilizado e L_2 caso haja rotação.

função Construtiva(**Permutação:** P; **Itens:** I; **Dimensões:** (W, H, D); **Lista de random-keys:** L_1, L_2)

1: $S \leftarrow \emptyset$ ▷ Solução construída começa vazia
2: $\lambda \leftarrow 0$ ▷ Número de caixas abertas começa com 0
3: **para** cada $i \in P$ **faça** ▷ Para cada item na ordem da permutação
4: $a_i \leftarrow i$-ésimo item de I ▷ Item a_i será empacotado
5: EM$^* \leftarrow \emptyset$ ▷ Melhor EM
6: **para** cada $C \in S$ **faça** ▷ Para cada caixa aberta
7: **se** Vazio(C) $\geq$ Espaço(a_i) **então** ▷ Se o espaço vazio de C pode conter a_i
8: **para** cada EM $\in C$ **faça**
9: **se** Verifica(a_i, EM) **então** ▷ Se o EM pode conter a_i
10: **se** f(EM) $< f$(EM*) **então** EM$^* \leftarrow$ EM ▷ Mantém o melhor EM
11: **se** EM$^* \neq \emptyset$ **então** Parar o *loop* ▷ Se existe um candidato pare a busca
12: **se** EM$^* \neq \emptyset$ **então** $C \leftarrow$ EM* ▷ Referência para o empacotamento
13: **senão** ▷ Não há candidatos
14: $C \leftarrow$ Caixa(W, H, D) ▷ Uma nova caixa é aberta
15: $S \leftarrow C$ ▷ Adiciona a nova caixa em S
16: $\lambda \leftarrow \lambda + 1$ ▷ Atualiza o número de caixas abertas
17: EM$^* \leftarrow$ Espaço(C, W, H, D) ▷ EM* recebe o único EM da nova caixa
18: $r_1 \leftarrow$ i-ésima *random-key* de L_1 ▷ *Random-key para o critério de escolha*
19: $r_2 \leftarrow$ i-ésima *random-key* de L_2 ▷ *Random-key para a orientação do item*
20: Orienta(a_i, EM*, r_2) ▷ Se há rotação, determina a orientação de a_i através de r_2
21: Empacota(a_i, C, EM*, r_1) ▷ Empacota a_i através do Algoritmo 13 utilizando r_1 caso Best Fit ou Best EM seja o critério de escolha
22: **retorna** S, λ ▷ Retorna a solução S e o número de caixas abertas

Algoritmo 21 Decodificador utilizado no BRKGA

função Decodifica(**Indivíduo:** i; **Itens:** I; **Dimensões:** (W, H, D))

1: P $\leftarrow$ Ordena(i) ▷ Obtém a permutação através de ordenação de *random-keys*
2: $L_1 \leftarrow$ *random-keys* referente ao CI ▷ Caso o critério seja Best Fit ou Best EM
3: $L_2 \leftarrow$ *random-keys* referente à orientação ▷ Caso seja possível rotacionar
4: s $\leftarrow$ Empacota(P, I, (W, H, D), L_1, L_2) ▷ Empacotamento com o Algoritmo 20
5: **retorna** s

3.10.1 Variable Cross Descent

O **Variable Cross Descent** (VCD) é um algoritmo inspirado no VND que é aplicado na geração da população inicial e mutantes, com o objetivo de melhorar a qualidade

média das soluções iniciais e de acelerar o processo de evolução do BRKGA para os problemas de empacotamento. Um dos maiores desafios de implementar o BRKGA é sintetizar estratégias algorítmicas que sejam capazes de utilizar a codificação em forma de *random-keys* de forma vantajosa. Pois, geralmente, alguma operação ou busca local no vetor de *random-keys* tem a mesma probabilidade de encontrar soluções de boa qualidade comparada ao processo de geração completamente aleatória do próprio. Por exemplo, na prática, uma busca local ou operação que permuta, troca ou desloca elementos de uma solução codificada somente randomizam a própria permutação gerada, ou seja, não há um verdadeiro potencial de guiar a decodificação em uma direção que gere soluções de qualidade superior comparada com um método aleatório. O VCD é uma estratégia algorítmica que é capaz de utilizar a codificação em forma de *random-keys* em conjunto com o método de reprodução para sintetizar soluções que têm uma alta probabilidade de apresentar uma boa qualidade comparada ao método aleatório.

No BRKGA original, a população inicial e os mutantes são indivíduos aleatórios compostos por n *random-keys* no intervalo $[0, 1]$, isto é, codificações de permutações aleatórias. Geralmente, a qualidade média das soluções geradas com estas permutações aleatórias é baixa comparada com o provável ótimo global conhecido para as instâncias. Através de testes empíricos utilizando a heurística construtiva proposta na Seção 3.5, observa-se que os prováveis ótimos globais do 3BP e do 2BP, para a grande maioria das instâncias, são gerados através de permutações de entrada que realizam o empacotamento de várias subsequências de itens ordenados pelo maior volume, altura, largura ou profundidade. Na prática, as soluções finais obtidas pelo BRKGA utilizam uma permutação de entrada na qual os itens estão parcialmente ordenados. Quando ordenamos os itens diretamente por maior volume, largura, altura ou profundidade para obter uma permutação que os empacote na respectiva ordem, a qualidade de cada solução construída pela a heurística proposta com essas permutações é superior comparada com a média dos indivíduos aleatórios. Em alguns casos, estas soluções com empacotamento ordenado apresenta uma qualidade próxima do ótimo global conhecido. Assim, o VCD é de uma estratégia que tem alta probabilidade de gerar vetores de *random-keys* que codificam permutações que empacotam várias subsequências de itens ordenadas durante a geração da população inicial e mutantes. Há uma chance alta da qualidade média da população inicial ser próxima do provável ótimo global ou, até mesmo, encontrar a própria durante este processo. Em consequência, o processo de evolução do algoritmo deve acelerar.

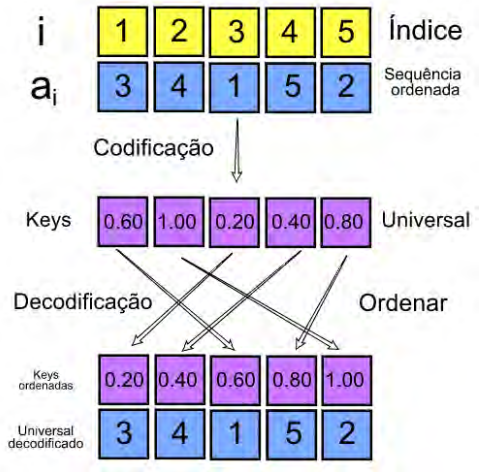
Para aplicar o VCD, devemos executar um processo preliminar para introduzir novos indivíduos que codificam permutações que representam sequências de itens ordenados para o empacotamento. No 3BP, o primeiro passo é ordenar os itens de entrada pelo maior volume, altura, largura e profundidade para obter 4 permutações que representam as respectivas sequências de empacotamento. No caso do 2BP, obtemos 3 indivíduos, pois os itens não têm profundidade. Logo após, este processo deve utilizar a heurística construtiva do BRKGA detalhada no Algoritmo 20 para empacotar os itens com cada permutação gerada como entrada. Dessa forma, obtemos 4 ou 3 soluções com uma alta probabilidade de serem próximas do ótimo global. Então, o procedimento deve codificar estas soluções em indivíduos chamados **Universais** através da Equação 3.3, tal que r_i é a *random-key* do i-ésimo item e p_i é o i-ésimo termo da permutação, isto é, a ordem do i-ésimo item. Caso o cromossomo do indivíduo necessite de algum conjunto adicional de *random-keys* além dos n primeiros devido ao desempate da escolha do CI ou a determinação da orientação, esses vetores devem ser gerados aleatoriamente e associados ao respectivo cromossomo do universal. Finalmente, os universais devem ser mantidos em uma variável auxiliar que os

armazena ordenados do melhor para o pior durante todo o processo do BRKGA-VCD e devem ser introduzidos na população inicial.

$$r_i = p_i \times \frac{1}{n}, 1 \leq i \leq n \tag{3.3}$$

A Figura 18 exemplifica o processo de codificação para gerar um universal. A figura mostra a permutação com a sequência de empacotamento de cada item a_i, $1 \leq i \leq n$, com o respectivo índice i. Neste exemplo, o item a_1 será o terceiro a ser empacotado, a_2 será o quarto e assim por diante. Desta maneira, a figura mostra a codificação através da Equação 3.3 destacando as *keys* que compõe o universal que codifica este exemplo. A figura também ilustra como seria a decodificação deste universal, a fim de mostrar que a permutação gerada é a sequência original. Vale enfatizar que o processo preliminar não efetua a decodificação do indivíduo, e somente realiza a codificação de cada sequência ordenada pelo maior volume, largura, altura e profundidade.

Figura 18 – Exemplo do processo de codificação de uma permutação com $n = 6$.



Cada *random-key* que compõe um indivíduo no BRKGA para os problemas de empacotamento tem um significado importante. Considere um vetor com n *random-keys*, tal que o elemento na posição k tem o valor 0,1. Sabendo que cada *random-key* é um número real no intervalo $[0, 1]$ uniformemente aleatório, isto significa que o item na posição k tem uma alta probabilidade de ser um dos primeiros a ser empacotado, pois sua *random-key* é próxima de zero. De forma análoga, considere que a *random-key* que está na posição k' é 0,9, assim, o k'-ésimo item tem uma alta probabilidade de ser um dos últimos a ser empacotado. Portanto, cada *random-key* representa a probabilidade relativa da ordem em que seu respectivo item está na sequência de empacotamento. O VCD combina a reprodução do BRKGA com os universais para gerar codificações que têm alta probabilidade de sintetizar permutações que representam ordens de empacotamento com várias subsequências ordenadas.

O VCD recebe como parâmetro de entrada uma população a ser preenchida, um tamanho alvo para esta população e os universais ordenados do melhor para o pior. Fora do processo iterativo da meta-heurística, o VCD é utilizado para gerar a população inicial. A população de entrada deve ser a inicial com os universais já presentes, por causa do processo preliminar descrito anteriormente, e o valor alvo é o próprio parâmetro calibrável P_{max} do BRKGA que dita a quantidade máxima de indivíduos na população. Durante as iterações da meta-heurística, o VCD é utilizado para gerar os mutantes. A população

de entrada deve ser a geração atual e o valor alvo de tamanho deve ser a quantidade de indivíduos que já estão na população somado a $\omega \times P_{max}$. Ou seja, o valor alvo de entrada para o tamanho da população é dado pelo parâmetro calibrável ω que é a fração BOT do BRKGA. O objetivo é introduzir $100\omega\%$ mutantes, assim como no método original. Além disso, o VCD recebe também como entrada os parâmetros utilizados pelo decodificador detalhado no Algoritmo 21 e o parâmetro ρ que é utilizado na reprodução. A reprodução realizada durante o VCD é a mesma do BRKGA original, onde ρ é o mesmo parâmetro de entrada calibrável da meta-heurística cujo objetivo é determinar a probabilidade de escolha Elitista. Como a aptidão de cada universal é próxima do provável ótimo global para a grande maioria das instâncias, durante o VCD assumimos que cada universal é o parente Elite. Assim, o objetivo do VCD é gerar indivíduos que codificam subsequências ordenadas com alta probabilidade utilizando o mesmo parâmetro ρ tanto na reprodução do processo iterativo do BRKGA quanto no VCD.

Este método realiza seu processo iterativo até que a população de entrada atinja o tamanho do alvo de entrada. Então, o primeiro passo da iteração do método é sintetizar um indivíduo completamente aleatório, logo após, decodificá-lo para manter sua aptidão armazenada em uma variável auxiliar A. Como o processo de empacotamento realizado pela a heurística construtiva exige um alto tempo de execução, este indivíduo aleatório é introduzido na população. Além disso, em algumas minorias, o próprio indivíduo aleatório pode codificar uma solução de boa qualidade. Portanto, o método tem uma chance de manter este indivíduo. Em seguida, o VCD sintetiza novos indivíduos através da reprodução dos universais com uma estratégia similar à troca de vizinhança do VND, sempre introduzindo o novo filho gerado na população de entrada. Com uma variável auxiliar k que começa com o valor 1, o VCD faz a reprodução com o universal U_k comparando a aptidão do filho gerado com A. Se a aptidão do filho gerado for melhor que A, a variável auxiliar A recebe a aptidão deste novo filho e k volta ao valor 1. Caso contrário, A permanece com o mesmo valor e k é incrementado em 1. Assim, a ideia deste mecanismo é reproduzir repetidamente com os universais de melhor aptidão um número maior de vezes, pois a lista de universais deve estar ordenada por qualidade por causa do processo preliminar. Se o valor de k atingir um número maior que a quantidade de universais, o VCD recomeça no primeiro passo gerando um novo indivíduo completamente aleatório, prosseguindo novamente para o processo iterativo de reprodução com os universais. O segundo indivíduo utilizado no processo de reprodução deve ser um novo vetor de *random-keys* que não é decodificado e nem introduzido na população de entrada.

A ideia principal do BRKGA é viciar as *random-keys* da população Elite através da reprodução que tem uma probabilidade alta de gerar filhos que mantém a respectiva *key* do seu parente Elite. Como a porção Elite é menor que a fração não-Elite, o BRKGA pode escolher o mesmo Elite diversas vezes, assim, em conjunto com o parâmetro ρ, o processo de evolução da meta-heurística vicia as chaves que codificam boas soluções. Desta maneira, o VCD é um processo que vicia as chaves que codificam sequências ordenadas de empacotamento. Os primeiros filhos gerados com o VCD devem codificar sequências com grandes subsequências de itens ordenados. Então, a reprodução realizada na própria meta-heurística deve utilizar os filhos introduzidos no VCD para gerar mais soluções que têm alta probabilidade de codificar subsequências cada vez menores de caixas ordenadas. A ideia desta estratégia é atingir rapidamente o provável ótimo global da instância, fazendo com o que a qualidade média da geração seja superior ao BRKGA original. O Algoritmo 22 detalha o VCD proposto, tal que o método *Reproduzir* é o processo de reprodução do BRKGA e a função *Decodifica* é o Algoritmo 21 utilizando a

heurística construtiva detalhada pelo Algoritmo 20. Assim, o BRKGA-VCD é o BRKGA original utilizando o Algoritmo 22 para gerar a população inicial e mutantes aliado ao procedimento preliminar descrito para introduzir os Universais.

Algoritmo 22 Variable Cross Descent

função VCD(**População:** P, U; **Tamanho:** q; **Taxa da reprodução:** ρ; **Itens:** I; **Dimensões:** (W, H, D))
1: **enquanto** $|P| < q$ **faça** ▷ Enquanto a população não estiver preenchida
2: $i \leftarrow$ Novo indivíduo completamente aleatório
3: $s \leftarrow$ DECODIFICA(i, I, (W, H, D)) ▷ Algoritmo 21
4: Aptidão de i, $A \leftarrow$ Avaliação de s
5: $P \leftarrow P \cup i$ ▷ Introduz i na população já com avaliação
6: $k \leftarrow 1$ ▷ Procedimento inspirado por VND
7: **enquanto** $k \leq |U|$ e tamanho de $P < q$ **faça** ▷ U armazena os Universais
8: $f \leftarrow$ REPRODUZIR(U_k, i, ρ) ▷ Filho do universal k com i
9: $s \leftarrow$ DECODIFICA(f, I, (W, H, D)) ▷ Algoritmo 21
10: Aptidão de $f \leftarrow$ Avaliação de s
11: $P \leftarrow P \cup f$ ▷ Introduz o filho em P já com avaliação
12: **se** Aptidão de $f < A$ **então**
13: $A \leftarrow$ Aptidão de f
14: $k \leftarrow 0$
15: **senão** $k \leftarrow k + 1$
16: **retorna** P

3.10.2 Complexidade de Uma Iteração

O processo iterativo do BRKGA e do BRKGA-VCD depende de um critério de parada que, geralmente, é uma quantidade máxima de iterações ou tempo de execução. Logo, é difícil prever a quantidade de iterações que o processo iterativo dessas abordagens vai realizar durante uma execução. Portanto, esta seção descreve a complexidade no pior caso dos componentes do BRKGA e do BRKGA-VCD proposto através da contagem de instruções básicas, com o objetivo de descrever o comportamento prático de uma iteração.

O decodificador utilizado em ambas as versões do BRKGA é detalhado pelo Algoritmo 21. O passo 1 deste componente é ordenar o vetor com n *random-keys* que compõem o indivíduo passado como parâmetro de entrada, onde n é o número de itens. A saída deste passo é uma permutação que representa a ordem dos itens no empacotamento. Esta ordenação pode ser feita através de um algoritmo eficiente, por exemplo, o *Merge Sort*. Então, este passo tem complexidade $\mathcal{O}(n \log n)$. O passo 4 realiza o empacotamento dos itens através do Algoritmo 20, que é uma modificação da heurística proposta na Seção 3.5. A modificação altera os parâmetros de entrada para receber duas novas listas de *random-keys* que torna determinística a escolha do CI, caso o critério de avaliação de um EM seja Best Fit ou Best EM e haja rotação. Esta modificação não altera a complexidade estabelecida na Seção 3.5.3. O procedimento que rotaciona o item pode escolher até 6 ou 2 orientações aleatoriamente no 3BP e no 2BP, respectivamente, para o método original. Isto não se altera nesta variante. A escolha agora depende de um valor previamente gerado em conjunto com a Equação 3.2. Analogamente, a complexidade do procedimento que escolhe o CI também não altera a complexidade, pois temos até 8 ou 4 CI candidatos escolhidos para o 3BP e o 2BP, respectivamente, nas duas versões. Além disso, há uma

modificação na maneira como escolhemos o local de empacotamento: o item é empacotado na primeira caixa que é possível encaixá-lo, mantendo os EM candidatos somente desta. A heurística original verifica todas as caixas, porém, esta modificação também não altera a complexidade final do método, pois, no pior caso, pode não haver caixas que possam conter o item atual. Desta maneira, a complexidade do Algoritmo 20 é a mesma da heurística construtiva original, sendo $\mathcal{O}(n \times \max(n \times m \log m, m^2))$, onde m é o valor que majora o número de EM presentes em todas as caixas durante toda a execução da heurística. Como mencionado na Seção 3.5.3, o valor m é significativamente maior que n, exceto em minorias particulares. Portanto, o procedimento de empacotamento nos fornece a complexidade final do decodificador, sendo $\mathcal{O}(n \times m^2)$ para a maioria das instâncias.

De acordo com o Algoritmo 11, o passo 1 sintetiza a população inicial baseado na quantidade P_{max} de indivíduos. O passo seguinte decodifica e avalia esta população mantendo os indivíduos ordenados do melhor para o pior. No BRKGA original, estes passos podem ser implementados em conjunto, de maneira a formar um processo preliminar com uma clausula de repetição que realiza P_{max} iterações gerando cada indivíduo e o decodificando imediatamente. Posteriormente, este processo preliminar deve ordenar a população do melhor para o pior e deve ser executado antes das iterações principais da própria metaheurística. Durante cada iteração da cláusula de repetição do processo preliminar, um indivíduo da população inicial é gerado aleatoriamente com um vetor de n *random-keys*, podendo também gerar dois vetores adicionais com n *random-keys* cada um dependendo do critério de avaliação do EM e da possibilidade de rotação. Portanto, a complexidade da geração de um único indivíduo é $\Theta(n)$. Proseguindo na mesma iteração, o indivíduo é decodificado em $\mathcal{O}(n \times m^2)$ passos para a grande maioria das instâncias e inserido no final de um vetor que formaliza a população. Após o término dessas iterações, o processo preliminar deve ordenar o vetor com os indivíduos do melhor para o pior com um algoritmo eficiente de ordenação em $O(P_{max} \log P_{max})$ passos. Portanto, a complexidade deste processo preliminar é $\mathcal{O}(P_{max} \times n \times m^2)$ para a grande maioria das instâncias.

O BRKGA-VCD também realiza um procedimento preliminar para gerar a população inicial, que pode ser implementado de maneira análoga. Nesta versão, o VCD detalhado pelo Algoritmo 22 deve substituir a cláusula de repetição do processo preliminar da versão original, mantendo o passo de ordenação da população. O procedimento preliminar do BRKGA-VCD também deve gerar e decodificar os universais antes de executar o VCD. No caso do 3BP, devemos gerar 4 universais através da ordenação dos itens de entrada pelo maior volume, altura, largura e profundidade. Analogamente, no 2BP devemos gerar somente 3 universais. Cada ordenação de itens pode ser efetuada com um algoritmo eficiente em $\mathcal{O}(n \log n)$ passos. Logo após, uma permutação que representa a sequências de itens a serem empacotados é gerada e passada como parâmetro de entrada para o Algoritmo 20 que sintetiza as soluções. Além disso, o processo de geração de cada universal deve construir até dois vetores de *random-keys* com n números aleatórios caso haja a possibilidade de rotacionar os itens ou critério de avaliação de um EM seja o Best Fit ou Best EM. A geração de cada vetor exige $\Theta(n)$ instruções. Posteriormente, os Universais devem ser ordenados do melhor para o pior e inseridos em um novo vetor. Portanto, a complexidade do passo de geração dos universais é dada pelo processo de empacotamento, sendo $\mathcal{O}(n \times m^2)$ para a maioria das instâncias. Sabendo que a complexidade da ordenação da população inicial realiza $\mathcal{O}(P_{max} \log P_{max})$ operações, o último passo que precisa ser analisado para a complexidade do processo preliminar do BRKGA-VCD é o próprio VCD.

A implementação do VCD recebe como parâmetro de entrada um número inteiro positivo q alvo para preencher a população P, também passada como entrada. Ela deve

receber também o vetor com os universais e os parâmetros adicionais para a reprodução e o empacotamento. O passo 1 do VCD é uma clausula de repetição que realiza até $|P| - q$ repetições. O passo 2 gera um vetor aleatório com até $3n$ *random-keys*, assim, tem complexidade $\Theta(n)$. O passo 3 utiliza o decodificador para obter uma solução, então tem complexidade $\mathcal{O}(n \times m^2)$ para a grande maioria das instâncias. O passo 4 introduz o indivíduo no final da população, a qual deve ser implementada como um vetor de acesso direto cuja as chaves são indivíduos. Assim, tem complexidade constante. O passo 8 executa o processo de reprodução para gerar um filho. Esta função é uma cláusula de repetição que itera sobre os genes do filho. Desse modo, realiza até $3n$ repetições. Para cada gene, um número aleatório deve ser gerado e comparado através da taxa de elitismo ϵ, tal que escolhemos o respectivo gene do parente Elite com $100\epsilon\%$ de chance, caso contrário, o respectivo gene do parente não-Elite. No VCD, assumimos que o parente Elite é o universal que está sendo utilizado e o outro é um novo indivíduo não decodificado. Assim, a complexidade do processo de reprodução e do passo 8 é $\Theta(n)$. O passo 9 decodifica o filho gerado no passo anterior. Então, tem complexidade $\mathcal{O}(n \times m^2)$ para a grande maioria das instâncias. O restante dos passos não mencionados, exceto o passo 7, são comparações e atribuições entre referências ou variáveis básicas, logo, têm complexidade constante.

O passo 7 é uma cláusula de repetição controlada pela variável k, análoga ao VND. O número de repetições depende da aptidão das soluções geradas pelo processo de reprodução. Este passo também depende do valor $|P| - q$, pois o objetivo do VCD é preencher a população de entrada. Assim, uma iteração do passo 7 diminui uma repetição que o passo 1 realiza, por causa da cardinalidade de P que é incrementada em 1 a cada iteração. Ou seja, o número de repetições que os passos 1 e 7 realizam em conjunto é exatamente $|P|-q$. Observe também que a complexidade do conjunto de passos internos da cláusula de repetição presente no passo 7 é a mesma do conjunto de passos internos da clausula presente no passo 1 (excluindo os passos 7 até 15), sendo dada pelo decodificador como $\mathcal{O}(n \times m^2)$ para a maioria das instâncias. Então, o conjunto de instruções internas que compõe as iterações dos passos 1 e 7 têm o mesmo objetivo: decodificar indivíduos e inseri-los em P. Assim, a complexidade final do VCD é $\mathcal{O}((|P| - q)(n \times m^2))$ para a grande maioria das instâncias. Isto é, a complexidade do VCD é o número total de indivíduos gerados multiplicado pela quantidade de instruções efetuadas pelo decodificador.

No procedimento preliminar efetuado no BRKGA-VCD, o valor q é o próprio P_{max} e a população a ser gerada é a inicial. A população que é passada como parâmetro de entrada para o VCD neste procedimento é o próprio vetor com os universais. Como a quantidade de universais é um número constante dependente da versão bidimensional ou tridimensional do problema, a quantidade de indivíduos gerados é próxima de P_{max}. Dessa forma, a complexidade do VCD majora as dos outros passos do procedimento preliminar. Portanto, a complexidade do procedimento preliminar no BRKGA-VCD é $\mathcal{O}(P_{max} \times n \times m^2))$. O processo preliminar do BRKGA também tem a mesma complexidade.

Os passos realizados durante o processo iterativo do BRKGA e do BRKGA-VCD é descrito pelo Algoritmo 11. Como mencionado anteriormente, considere que a população é um vetor de acesso direto, onde os elementos são indivíduos. Portanto, qualquer inserção de um novo elemento pode ser efetuado no final deste vetor com complexidade constante. O primeiro passo da iteração é o passo 4, que realiza a seleção. A seleção consiste em copiar os $100\epsilon\%$ melhores indivíduos da geração anterior para a atual, ou seja, a porção Elite de $P_{max} \times \epsilon$ indivíduos. Como um indivíduo não é representado por uma variável básica, a complexidade da cópia não é constante. A estrutura que armazena um cromossomo é um vetor com até $3n$ números reais que podem ser implementados como elementos de

precisão simples ou dupla. Então, a complexidade da cópia de um cromossomo é $\Theta(n)$. Consequentemente, a complexidade do passo 4 é $\Theta(P_{max} \times \epsilon \times n)$. O passo 5 realiza a mutação. No BRKGA, a mutação gera $100\omega\%$ indivíduos completamente aleatórios e os insere na geração atual. Posteriormente no passo 7, os mutantes devem ser decodificados, podemos decodifica-los assim que eles são gerados no passo 5. A implementação deste procedimento de mutação no BRKGA é uma clausula de repetição que deve se repetir $P_{max} \times \omega$ vezes, tal que cada iteração gera um novo indivíduo com até $3n$ *random-keys* aleatórias, sintetiza uma solução através do decodificador e insere este mutante na geração atual. A complexidade da geração de um indivíduo aleatório é $\Theta(n)$. A complexidade da decodificação deste mutante é $\mathcal{O}(n \times m^2)$ para a maioria das instâncias e a inserção na geração atual tem complexidade constante. Assim, o passo 5 no BRKGA em conjunto com as decodificações parciais do passo 7 tem complexidade $\mathcal{O}(P_{max} \times \omega \times n \times m^2)$.

Para o BRKGA/VCD, o procedimento de mutação é feito pelo próprio VCD e também deve introduzir $100\omega\%$ mutantes. Neste momento, a cardinalidade de P é $|P| = P_{max} \times \epsilon$ devida aos Elites copiados no procedimento de seleção. O parâmetro q de entrada é passado com o valor $(P_{max} \times \omega) + (P_{max} \times \epsilon)$, e a população de entrada é a geração atual. Logo, o número de indivíduos que o VCD deve gerar neste passo é $P_{max} \times \omega$. Assim, a complexidade do passo 5 no BRKGA/VCD é $\mathcal{O}(P_{max} \times \omega \times n \times m^2)$ para a grande maioria das instâncias. O passo 6 efetua o processo de reprodução até o restante da população ser preenchida completamente. Como descrito anteriormente, o processo de reprodução é uma cláusula de repetição sobre os genes do filho, que por sua vez têm até $3n$ *random-keys*. Desse modo, a reprodução tem complexidade $\Theta(n)$. Os filhos gerados devem ser decodificados no passo 7. A implementação deste passo pode ser uma clausula de repetição que executa a reprodução entre um parente Elite e um não-Elite aleatório e imediatamente decodifica o filho gerado. Desta maneira, a quantidade de instruções em conjunto com as necessárias para realizar parcialmente o passo 7 nos da a complexidade do passo 6. A quantidade de filhos que deve ser gerada é $P_{max} - (P_{max} \times \epsilon) - (P_{max} \times \omega) = P_{max}(1 - \epsilon - \omega)$. Portanto, a complexidade do passo 6 é $\mathcal{O}(P_{max}(1 - \epsilon - \omega) \times n \times m^2)$. Finalmente, o passo 7 decodifica e ordena a geração atual do melhor para o pior. Como as decodificações podem ser efetuadas durante os passos 5 e 6 no BRKGA e no BRKGA-VCD, o passo 7 consiste somente em ordenar a população com um algoritmo eficiente. Assim, a complexidade deste passo último passo é $P_{max} \log P_{max}$. A complexidade final de uma iteração do BRKGA e do BRKGA-VCD é dada pelos passos 5 e 6, sendo $\mathcal{O}(P_{max} \times n \times m^2 \times (1 - \text{máximo}(\epsilon, \omega)))$.

A Tabela 8 sumariza as complexidades dos processos que compõem o BRKGA e o BRKGA-VCD em conjunto com a complexidade final de uma iteração, relacionando a instrução com o respectivo passo no Algoritmo 11 e sua complexidade.

Tabela 8 – Sumário da complexidade dos componentes do BRKGA e do BRKGA/VCD.

Processo	**Passo**	**Complexidade**
Decodificador (Algoritmo 21)	2 e 7	$\mathcal{O}(n \times m^2)$*
Preliminar (BRKGA)	1 e 2	$\mathcal{O}(P_{max} \times n \times m^2)$*
Preliminar (BRKGA-VCD)	1 e 2	$\mathcal{O}(P_{max} \times n \times m^2)$*
VCD (Algoritmo 22)	1, 2, 5 e 7	$\mathcal{O}((\lvert P\rvert - q) \times n \times m^2)$*
Seleção	4	$\Theta(P_{max} \times \epsilon \times n)$
Mutação com decodificação (BRKGA)	5 e 7	$\mathcal{O}(P_{max} \times \omega \times n \times m^2)$*
Mutação com decodificação (BRKGA-VCD)	5 e 7	$\mathcal{O}(P_{max} \times \omega \times n \times m^2)$*
Reprodução com decodificação	6 e 7	$\Theta(n)$
1 iteração (BRKGA e BRKGA-VCD)	Final	$(P_{max} \times n \times m^2 \times (1 - \text{máximo}(\epsilon, \omega)))$

*Complexidade para a grande maioria das instâncias.

4 EXPERIMENTOS COMPUTACIONAIS

Este capítulo exibe os resultados computacionais dos algoritmos propostos no Capítulo 3 para o 3BP e o 2BP, a fim de de compará-los entre si e com os algoritmos estado da arte obtidos na literatura. Vamos mostrar que esses métodos são capazes de produzir soluções de alta qualidade para instâncias grandes dos problemas de empacotamento, com pouco tempo de execução em CPU quando executados em um notebook munido com um processador de 2,5GHz.

O capítulo mostra os resultados de um experimento computacional preliminar, que foi executado para calibrar as heurísticas GRASP, Busca Tabu e BRKGA propostas para os problemas de empacotamento, selecionando o melhor algoritmo que pode ser aplicado em cada fase ou componente de cada uma delas para as instâncias tridimensionais detalhadas na Seção 4.1.1. Posteriormente, os parâmetros calibráveis foram ajustados empiricamente com o auxílio de uma ferramenta computacional.

A análise comparativa entre as três heurísticas propostas é baseada nos resultados obtidos em um extensivo experimento computacional efetuado com o conjunto de 820 instâncias detalhadas na Seção 4.1. Todas as execuções foram feitas no mesmo ambiente computacional, e o mesmo critério de parada foi usado nos três algoritmos.

Finalmente, o método com o melhor resultado, o BRKGA-VCD, foi executado novamente para o mesmo conjunto de instâncias, com o objetivo de compará-lo, em termos de qualidade da solução obtida, com os métodos descritos no Capítulo 3, que representam o estado da arte encontrado na literatura, considerando itens de orientação fixa ou itens rotacionáveis.

4.1 Instâncias

4.1.1 Tridimensional

A base de testes utilizada para o 3BP é composta por 320 instâncias geradas pela aplicação proposta em (MARTELLO; PISINGER; VIGO, 2000). O código do gerador escrito na linguagem de programação *ANSI C* está disponível em <http://www.diku.dk/~pisinger/codes.html>. Os autores implementaram uma função para a geração de números pseudoaleatórios que recebe uma semente como parâmetro de entrada. Como esta semente é definida no próprio código, as instâncias geradas devem ser as mesmas, independente do ambiente computacional.

As instâncias são organizadas em 8 classes com 10 instâncias para cada quantidade de itens $n \in \{50, 100, 150, 200\}$, ou seja, 40 instâncias para cada classe. As 5 primeiras classes são uma generalização das instâncias utilizadas em (MARTELLO; VIGO, 1998), compostas por itens que podem assumir um dos 5 tipos detalhados na Tabela 9. Essa tabela relaciona o tipo com o intervalo que cada dimensão (w_i, h_i, d_i) inteira positiva pode assumir por sorteio, $i \in \{1, \cdots, n\}$. O segundo grupo de classes é uma generalização das instâncias propostas por (BERKEY; WANG, 1987). A configuração de cada classe

é detalhada abaixo através dos respectivos parâmetros (W, H, D) e (w_i, h_i, d_i) para a dimensão da caixas e dos itens.

- Classes 1 até 5: $W = H = D = 100$; Para cada classe $k \in \{1, 2, 3, 4, 5\}$, cada item é do tipo k com 60% de probabilidade e os outros 4 tipos com 10% cada de acordo com a Tabela 9.
- Classe 6: $W = H = D = 10$; (w_i, h_i, d_i) no intervalo $[1, 10]$.
- Classe 7: $W = H = D = 40$, (w_i, h_i, d_i) no intervalo $[1, 35]$.
- Classe 8: $W = H = D = 100$, (w_i, h_i, d_i) no intervalo $[1, 100]$.

Tabela 9 – Tipos de itens utilizados nas classes 1 até 5 das instâncias 3D relacionadas com o intervalo que cada dimensão inteira positiva pode assumir aleatoriamente.

Tipo	**Largura** w_i	**Altura** h_i	**Profundidade** d_i
1	$[1, \frac{1}{2}W]$	$[\frac{2}{3}H, H]$	$[\frac{2}{3}D, D]$
2	$[\frac{2}{3}W, W]$	$[1, \frac{1}{2}H]$	$[\frac{2}{3}D, D]$
3	$[\frac{2}{3}W, W]$	$[\frac{2}{3}H, H]$	$[1, \frac{1}{2}D]$
4	$[\frac{1}{2}W, W]$	$[\frac{1}{2}H, H]$	$[\frac{1}{2}D, D]$
5	$[1, \frac{1}{2}W]$	$[1, \frac{1}{2}H]$	$[1, \frac{1}{2}D]$

4.1.2 Bidimensional

Para o 2BP, a base de testes é composta por 500 instâncias divididas em 10 classes com 10 instâncias para os valores de $n \in \{20, 40, 60, 80, 100\}$, ou seja, 50 instâncias para cada classe. Inicialmente, (BERKEY; WANG, 1987) propuseram um conjunto de instâncias com 6 classes. Os autores de (MARTELLO; VIGO, 1998) estenderam esta base de testes definindo 4 novas classes, a fim de considerar uma distribuição de itens mais realista. Os itens com dimensões (w_i, h_i) das novas classes são gerados de acordo com os tipos relacionados na Tabela 10. Essa tabela relaciona o tipo do item com o intervalo que cada parâmetro inteiro positivo pode assumir aleatoriamente. As instâncias podem ser obtidas em <http://or.dei.unibo.it/library/two-dimensional-bin-packing-problem>. As classes são descritas abaixo de acordo com os respectivos parâmetros (W, H) e (w_i, h_i) para a dimensão da caixas e dos itens.

- Classe 1: $W = H = 10$; (w_i, h_i) no intervalo $[1, 10]$.
- Classe 2: $W = H = 30$, (w_i, h_i) no intervalo $[1, 10]$.
- Classe 3: $W = H = 40$, (w_i, h_i) no intervalo $[1, 35]$.
- Classe 4: $W = H = 100$, (w_i, h_i) no intervalo $[1, 35]$.
- Classe 5: $W = H = 100$, (w_i, h_i) no intervalo $[1, 100]$.
- Classe 6: $W = H = 300$, (w_i, h_i) no intervalo $[1, 100]$.
- Classe 7 até 10: $W = H = 100$; Para cada classe $k \in \{7, 8, 9, 10\}$, o item é do tipo k com 70% de probabilidade e os outros 3 tipos com 10% cada de acordo com a Tabela 10.

Tabela 10 – Tipos de itens utilizados nas classes 7 até 10 das instâncias 2D em conjunto com o intervalo que cada dimensão inteira positiva pode adotar através de sorteio.

Tipo	**Largura** w_i	**Altura** h_i
7	$[\frac{2}{3}W, W]$	$[1, \frac{1}{2}H]$
8	$[1, \frac{1}{2}W]$	$[\frac{2}{3}H, H]$
9	$[\frac{1}{2}W, W]$	$[\frac{1}{2}H, H]$
10	$[1, \frac{1}{2}W]$	$[1, \frac{1}{2}H]$

4.2 **Implementação e Ambiente Computacional**

A implementação para este trabalho de cada algoritmo proposto no Capítulo 3 utiliza a linguagem de programação *C++11* em conjunto com o *framework OptFrame*, proposto em (COELHO et al., 2011). O *framework* provê uma interface *C++11* simples para os componentes de meta-heurísticas, como as utilizadas neste trabalho, e é utilizada com sucesso para modelar e resolver problemas de otimização combinatória. A ferramenta fornece implementações eficientes de várias versões de cada meta-heurística, tal que elas são independentes do problema sendo resolvido. A principal vantagem é que o usuário pode focar somente nos aspectos específicos do problema para sintetizar uma heurística que o solucione. As implementações foram compiladas através do *GCC (GNU Compiler Collection)*[1] com a *flag* de otimização $-O3$. O ambiente computacional utilizado em todos os testes neste trabalho consiste de um notebook munido da seguinte configuração:

- Processador Intel Core i5-240CM @2.5 GHz (1 núcleo utilizado).
- 8 GB de memória RAM.
- Sistema Operacional Ubuntu 16.10 (x64).

4.3 **Selecionando as Melhores Estratégias**

No Capítulo 3, diversos algoritmos para a resolução do 3BP e do 2BP foram propostos como alternativas para serem aplicados nas heurísticas GRASP, Busca Tabu e BRKGA descritas nas Seções 3.8, 3.9 e 3.10 respectivamente. Antes de realizar uma comparação entre os três métodos, esta seção apresenta os resultados de um experimento computacional preliminar, com o objetivo de selecionar os algoritmos que devem ser usados em cada fase ou componente das heurísticas propostas. Por exemplo, a primeira fase do algoritmo GRASP proposto consiste em construir uma solução desde o princípio através de um método semi-guloso que pode realizar o empacotamento com o Best Fit, Best EM ou DFTRC. Assim, um dos objetivos deste experimento preliminar foi selecionar qual heurística construtiva deveria ser utilizada na primeira fase do GRASP durante o restante deste trabalho.

As instâncias utilizadas durante este experimento preliminar consistem de um subconjunto das instâncias do 3BP descritas na Seção 4.1.1: instâncias de número 1 das classes de 1 até 8 com $n \in \{50, 100, 150, 200\}$. Os algoritmos selecionados são aqueles que apresentam o melhor resultado neste conjunto de instância. Todas as execuções foram realizadas no ambiente computacional descrito na Seção 4.2 e utilizam itens com orientação fixa. Cada algoritmo comparado foi executado 10 vezes para cada instância. Os valores

[1] O GCC está disponível no seguinte sítio eletrônico: <https://gcc.gnu.org/>.

reportados são as médias dessas execuções. As execuções foram feitas com sementes inteiras distintas no intervalo $[1, 10]$ para o gerador de números pseudoaleatórios, a fim de controlar o âmbito de execução. As únicas exceções são a fase de construção GRASP e a geração da solução inicial da Busca Tabu, das Seções 4.3.1.1 e 4.3.2.1 respectivamente, pois os algoritmos comparados foram executados mais vezes para cada instância.

4.3.1 Greedy Adaptive Search (GRASP)

O algoritmo GRASP proposto para o 3BP e o 2BP é um método de duas fases com um parâmetro calibrável. A ordem em que os itens são dispostos para o empacotamento afeta a qualidade e o desempenho da primeira fase e, consequentemente, afeta também a solução final e o desempenho geral da heurística. A sequência em que os itens são empacotados depende de uma permutação, que é passada como parâmetro de entrada. Na segunda fase, o usuário tem a opção de usar a Busca Local proposta ou o algoritmo VND, descritos nas Seções 3.6 e 3.7, respectivamente. A implementação que utiliza a segunda opção é uma versão híbrida que chamamos de GRASP-VND.

Desta maneira, esta seção mostra os resultados de diversas baterias de testes preliminares que selecionam as estratégias para ambas as fases do GRASP proposto, o tipo de permutação e a função objetivo que devem ser implementados na versão final da heurística.

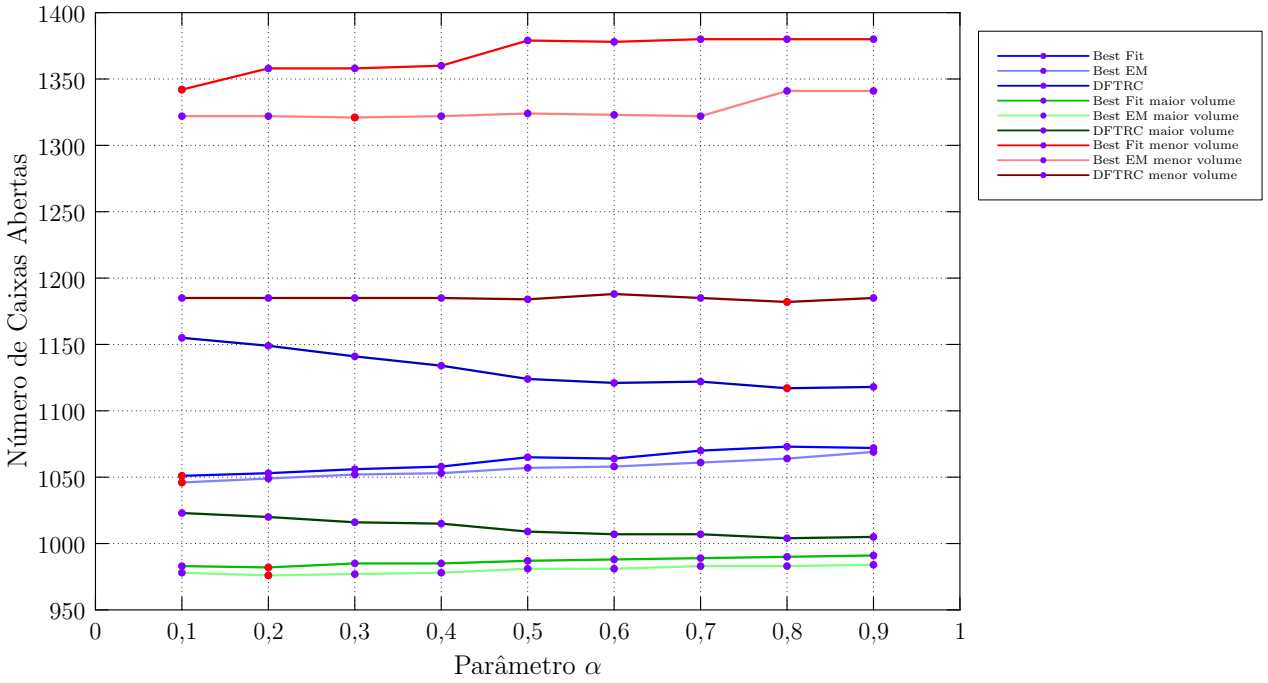
4.3.1.1 Fase de Construção

A primeira fase do GRASP proposto é um algoritmo semi-guloso que recebe como parâmetro de entrada uma lista de itens, uma permutação P e um número real $0 \leq \alpha \leq 1$, com o objetivo de construir uma configuração de empacotamento válida a cada iteração. A lista de itens é percorrida sequencialmente na ordem dada pela permutação P, onde seleciona-se um EM para cada item, o qual estabelece a localização de empacotamento. O parâmetro α, que dita a quantidade de EM que serão considerados no empacotamento, é um valor calibrável: o item será empacotado em um dos $100\alpha\%$ melhores EM aleatoriamente. Cada EM é avaliado de acordo com o critério Best Fit, Best EM ou DFTRC detalhados nas Seções 3.5.1 e 3.5.2, respectivamente.

Este primeiro teste empírico determina qual é o melhor critério de avaliação de um EM para o empacotamento na primeira fase do GRASP. Além disso, considera-se a utilização de diferentes tipos de permutações de entrada: sequências de empacotamento aleatórias ou ordenadas de acordo com o volume de cada item. Desta maneira, este teste consiste em executar somente a fase de construção do GRASP diversas vezes com cada critério de avaliação de um EM, considerando permutações que empacotam os itens, priorizando o maior volume, o menor volume ou sequências completamente aleatórias. Portanto, esta seção compara nove diferentes implementações da primeira fase do GRASP, oriundas das combinações de cada critério com cada tipo de permutação. O parâmetro calibrável α tem um impacto significativo no resultado final: um bom valor para este parâmetro em uma determinada implementação pode não ser boa para outra. Então, para manter a comparação justa, um total de 10 baterias de testes foram executadas para cada implementação, de modo que cada bateria utilizou exclusivamente um valor α distinto no conjunto $\{0{,}1, 0{,}2, \cdots, 0{,}9\}$. Uma bateria de teste consiste de 100 execuções da respectiva implementação com cada instância, utilizando sementes inteiras distintas no intervalo $[1, 100]$ para o gerador de números pseudoaleatórios. Os valores para α iguais a 0 ou 1 não foram considerados, pois significariam a eleição do melhor e único EM candidato ou a escolha completamente aleatória de um EM, respectivamente. O resultado para cada

instância é a média das execuções. Vale enfatizar que, em diferentes baterias de testes com sequências aleatórias, as mesmas permutações foram utilizadas para cada instância durante as execuções, a fim de manter a justiça na comparação.

Figura 19 – Acúmulo total de caixas obtidas com cada implementação e valor α.



A Figura 19 exibe o resultado deste experimento através de um gráfico que associa o acúmulo total de caixas com cada implementação e valor α. O gráfico apresenta 9 curvas: uma para cada implementação comparada, onde os pontos coloridos com a cor vermelha mostram o melhor acúmulo obtido com a respectiva implementação, enfatizando o melhor valor α para este conjunto de instâncias. O gráfico mostra que, para sequências ordenadas por maior volume e aleatórias, o critério para o empacotamento que fornece as melhores soluções é o Best EM, seguido pelo Best Fit e com o DFTRC em último lugar. Quando utilizamos permutações que codificam sequências de itens por menor volume, o DFTRC obtém os melhores resultados, seguido pelo Best EM e o Best Fit. O gráfico também mostra que a ordenação dos itens por maior volume tem um impacto positivo significativo na qualidade da solução encontrada, e que a ordenação dos itens por menor volume prejudica a qualidade da solução encontrada. Portanto, a estratégia selecionada para a fase de construção do GRASP é utilizar permutações ordenadas por itens de maior volume combinado com o empacotamento Best EM com $\alpha = 0{,}2$. Esta configuração foi utilizada durante todas as execuções posteriores do método apresentadas no restante desta seção.

Em relação ao tempo de execução em CPU, uma solução é construída em nanosegundos, e não há diferença relevante de desempenho comparando os diferentes critérios. Porém, nota-se um ganho de performance significativo quando utiliza-se permutações que representam sequências que ordenam os itens por maior volume, enquanto a ordenação de itens por menor volume causa uma perda de desempenho. Isto acontece porque, quando o empacotamento prioriza os menores itens, o número de EM presentes em cada caixa aberta aumenta, de modo que o espaço livre remanescente fica fragmentado. Outra razão

é que a quantidade de caixas da solução encontrada é maior, levando a um número maior de EM que precisam ser avaliadas. Além disso, os filtros descritos na Seção 3.5, que aceleram a execução do algoritmo, têm impacto menor quando a heurística empacota os itens de menor volume primeiro, pois os filtros são baseados nos itens com menor volume e dimensões que ainda não foram empacotados.

4.3.1.2 Fase de Melhora

A fase de melhora do GRASP proposto consiste em executar a busca local descrita na Seção 3.6 ou o método VND detalhado na Seção 3.7. Ambos os algoritmos recebem como parâmetro de entrada uma solução construída na fase anterior, e não introduzem novos parâmetros calibráveis. O objetivo desta seção é selecionar a estratégia que fornece as melhores soluções na segunda fase do GRASP para o conjunto de instâncias testados, porém o próprio VND utiliza a busca local proposta neste trabalho. Assim, os primeiros testes para esta fase focam em determinar as melhores estratégias para serem aplicadas na busca local.

A busca local proposta consiste em esvaziar um par de caixas da solução atual a cada iteração, onde todos os itens anteriormente presentes nestas caixas devem ser reempacotados nas outras através da heurística construtiva descrita na Seção 3.5. Desta maneira, devemos selecionar um dos critérios de avaliação dos EM para o reempacotamento: Best EM ou Best Fit. O DFTRC não deve ser considerado, pois a seção anterior mostra que a melhor estratégia para a primeira fase é o Best EM, então o reempacotamento da busca local não pode ser efetuado com o DFTRC devido à diferença na escolha do canto de inserção no momento da criação do EM. O Best EM e o Best Fit avaliam todos os cantos do EM, enquanto o DFTRC somente considera o canto inferior esquerdo frontal. Além disso, devemos decidir se a busca local deve utilizar o *First Improvement* ou o *Best Improvement*.

O objetivo da primeira bateria de testes para esta fase é selecionar um critério para o reempacotamento da busca local. Assim, três implementações do GRASP foram comparadas: uma para cada critério. Cada implementação foi executada 10 vezes com cada instância. Para manter a comparação justa, o *First Improvement* foi utilizado em todas as execuções, o critério de parada foi 10 segundos de tempo de execução em CPU e a primeira fase usou o Best EM com $\alpha = 0{,}2$. Além disso, a seção anterior mostra que a utilização de permutações que representam sequências ordenadas de itens por maior volume acelera o processamento do algoritmo e aumenta a qualidade das soluções, portanto, este é o tipo de permutação utilizada em todos os testes desta seção. A Tabela 11 apresenta o resultado da primeira bateria de testes. A primeira coluna informa a estratégia implementada na fase de melhora do GRASP. Os dados em cada linha da tabela são os resultados para um subconjunto de instâncias, tal que a segunda coluna informa a quantidade de itens n nas instâncias desse subconjunto. Os valores reportados nas colunas 3 a 7 são o mínimo, o máximo, a média e o desvio padrão do número de iterações GRASP. Os dados reportados nas colunas 8 a 11 são o mínimo, o máximo, a média e o desvio padrão do número de avaliações realizadas durante toda a execução da heurística. Os valores das três últimas colunas são referentes ao acúmulo mínimo, máximo e médio obtido nas execuções.

Quando se trata de desempenho, os resultados observados na tabela referente ao número de iterações e a quantidade de avaliações mostram que ambas as estratégias são equivalentes: há pouca diferença entre o mínimo, máximo e a média. Em termos de qualidade da solução encontrada, a tabela mostra que o reempacotamento com Best Fit obtém soluções equivalentes ou melhores que o Best EM durante fase de melhora do GRASP

para todas as instâncias usadas. Então, o critério selecionado para o reempacotamento dos itens na busca local do GRASP proposto é o Best Fit.

A segunda bateria de testes tem por objetivo determinar se a busca local deve utilizar a estratégia *First Improvement* ou *Best Improvement*. Esta comparação utiliza os dados obtidos nesta bateria de testes para contrastar com os dados reportados na Tabela 11 referentes à busca local *First Improvement* com Best Fit. Desta maneira, uma implementação do GRASP foi executada 10 vezes com cada instância utilizando a busca local *Best Improvement* com reempacotamento Best Fit. Para manter a comparação justa entre as duas estratégias, o restante da configuração da heurística permanece inalterada: o critério de parada é atingir 10 segundos de tempo de execução em CPU, a fase de construção consiste do semi-guloso Best EM com $\alpha = 0{,}2$ e o tipo de permutação mantém os itens ordenados por maior volume. A Tabela 12 reporta os resultados obtidos, seguindo a mesma disposição da Tabela 11.

A primeira tabela citada mostra que as soluções com o *Best Improvement* tem qualidades piores ou equivalentes do que as obtidas pela mesma implementação com *First Improvement* em todas as instâncias usadas. Além disso, a heurística realiza uma quantidade menor de iterações GRASP e um número maior de avaliações quando a busca local usa o *Best Improvement*. A função de reempacotamento é executada um número maior de vezes, consequentemente, um número menor de soluções são sintetizadas na fase de construção GRASP e um número maior de avaliações é realizada, aumentando o tempo de execução em CPU que uma iteração necessita para finalizar a avaliação completa de cada vizinhança de soluções. Apesar do aumento de avaliações, o número de iterações cai e a qualidade final das soluções piora. Portanto, a estratégia selecionada para a busca local do GRASP proposto é o *First Improvement*.

A terceira bateria de testes desta seção compara a utilização da busca local contra o algoritmo VND, proposto na Seção 3.7, durante a fase de melhora do GRASP. No entanto, cada vizinhança V_k, $2 \leq k \leq 4$, do VND requer a escolha de um critério para o reempacotamento dos itens, de forma análoga à busca local. Desta maneira, objetiva-se também selecionar um dos critérios de avaliação dos EM para o método de reempacotamento utilizado nestas vizinhanças. A análise comparativa é efetuada com os resultados reportados na Tabela 11 referentes ao GRASP com a busca local *First Improvement* e reempacotamento Best Fit contra os resultados obtidos nesta bateria de testes. A configuração usada no GRASP para o critério de parada, fase de construção e tipo de permutação é a mesma utilizada nas duas baterias de testes anteriores. Seguindo os resultados apresentados nesta seção, a vizinhança V_1 implementada no VND é a própria busca local *First Improvement* com o reempacotamento Best Fit. Assim, duas implementações do GRASP-VND híbrido, onde a fase de melhora é o VND proposto, foram executadas 10 vezes para cada instância. A primeira implementação utilizou o Best Fit como critério de avaliação das EM para o reempacotamento das vizinhanças V_2, V_3 e V_4, enquanto a segunda implementação usou o Best EM nestas vizinhanças. A Tabela 13 exibe os resultados desta bateria de testes com a mesma configuração das tabelas anteriores.

A tabela mostra que o número de iterações realizadas no GRASP-VND é menor, e a quantidade de avaliações é maior comparado com o GRASP que utiliza somente a busca local *First Improvement*. Este comportamento é consequência da necessidade adicional de tempo de execução em CPU que o método VND exige para gerar todas as vizinhanças. O número maior de avaliações no GRASP-VND é consequência da quantidade maior de reempacotamentos que são efetuados pelo conjunto de vizinhanças, principalmente V_4, que é executada múltiplas vezes por iteração. A tabela também mostra que o acúmulo

mínimo, máximo e médio obtidos para todas as instâncias é menor quando utilizamos o VND no lugar da busca local, sendo que os melhores resultados são obtidos quando todas as vizinhanças usam o reempacotamento com o critério Best Fit. Assim, o GRASP-VND com Best Fit obteve resultados melhores ou equivalentes comparado com o GRASP-VND com Best EM e a versão GRASP que utiliza somente a busca local. Então, o método VND com reempacotamento através do Best Fit para as vizinhanças V_2, V_3 e V_4 é o algoritmo selecionado para a fase de melhora do GRASP proposto.

Finalmente, uma última bateria de testes para a fase de melhora foi executada, a fim de analisar o impacto da utilização de V_2, V_3 e V_4 no VND. A última bateria de testes consiste da execução de diversas implementações do GRASP-VND que usam diferentes combinações de vinhanças. A configuração de todas as implementações utilizaram o reempacotamento Best Fit nas vizinhanças em conjunto com as configurações selecionadas nos testes anteriores. Desta maneira, o critério de parada foi atingir 10 segundos de tempo de execução em CPU. Cada implementação foi executada 10 vezes para cada instância. A Tabela 14 sumariza o resultado obtido, mostrando a média das execuções. A primeira coluna é a combinação de vizinhanças utilizadas no VND, e a segunda coluna é o acúmulo total de caixas para todas as instâncias.

A tabela mostra que a utilização de todas as vizinhanças tem um impacto positivo na qualidade da solução final estabelecida pelo GRASP-VND para as instâncias usadas, onde as vizinhanças V_2 e V_3 são as que causam um maior impacto na qualidade da solução. Podemos observar também que o resultado com qualquer combinação de três vizinhanças obtém resultados melhores do que as combinações que utilizam somente duas vizinhanças. Vale enfatizar que os testes empíricos efetuados para este trabalho sugerem que a ordem em que vizinhanças são aplicadas geralmente não afeta a qualidade final da solução encontrada. Entretanto, recomenda-se que V_1 seja a primeira vizinhança a ser executada devido ao aumento de desempenho no decorrer das iterações da heurística. A configuração selecionada para a fase de melhora do GRASP proposto é a versão com o VND que usa a busca local *First Improvement*, sendo que todas as vizinhanças utilizam o critério Best Fit no reempacotamento.

O autor desta dissertação considerou a utilização de diferentes combinações de critérios de avaliação no VND e estratégias mistas, onde cada critério é executado com certa probabilidade. Porém, nenhuma estratégia alcançou resultados melhores do que os obtidos com a configuração final selecionada para as instâncias consideradas.

Tabela 11 – Resultado da primeira bateria de testes da fase de melhora do GRASP. Comparação entre os critérios para o reempacotamento da busca local.

Fase de Melhora	n	Núm. de Iterações				Núm. de Avaliações				Z		
		Min	Máx	Med	Desvio	Min	Máx	Med	Desvio	Min	Máx	Med
	50	2470	3945	2852,3	1262,4	4172	8013	5341,5	3140,5	113	113	113,0
Busca Local	100	386	1462	674,4	910,9	854	3461	1564,5	2194,5	187	189	187,7
First Improvement	150	150	513	257,8	296,6	430	1784	775,0	1164,9	286	289	287,1
Best Fit	200	74	273	127,1	168,8	199	823	368,4	526,2	383	389	384,6
	Total acumulado em todas as instâncias:									**969**	**980**	**972,4**
	50	2463	3946	2849,8	1266,3	4165	8013	5336,7	3146,3	113	113	113,0
Busca Local	100	396	1461	679,4	903,8	860	3470	1572,2	2196,3	187	189	187,9
First Improvement	150	144	520	255,2	307,4	428	1788	775,4	1169,9	286	290	288,1
Best EM	200	64	280	126,8	178,1	195	830	369,9	534,0	383	392	385,8
	Total acumulado em todas as instâncias:									**969**	984	974.8

*Valores em **negrito** destacam a estratégia selecionada e o melhor acúmulo total de caixas.

Tabela 12 – Resultado da segunda bateria de testes da fase de melhora GRASP. Comparação entre as estratégias *First Improvement* e *Best Improvement*.

Fase de Melhora	n	Núm. de Iterações				Núm. de Avaliações				Z		
		Min	Máx	Med	Desvio	Min	Máx	Med	Desvio	Min	Máx	Med
	50	1277	1883	1572,5	429,2	6177	24067	14476,4	16573,6	113	133	113,0
Busca Local	100	179	506	291,5	262,2	2128	12088	7131,8	7279,5	188	190	188,5
Best Improvement	150	48	136	85,2	64,1	1429	7268	3950,3	5941,3	283	290	286,1
Best Fit	200	28	83	46,5	44,6	1107	6230	3170,1	5227,3	386	393	389,9
	Total acumulado em todas as instâncias:									970	986	977.5

*A estratégia relacionada nesta tabela não foi selecionada.

Tabela 13 – Resultado da terceira bateria de testes da fase de melhora GRASP. Comparação entre a busca local, o VND Best Fit e o VND Best EM.

Fase de Melhora	n	Núm. de Iterações				Núm. de Avaliações				Z		
		Min	Máx	Med	Desvio	Min	Máx	Med	Desvio	Min	Máx	Med
	50	715	1539	1086,2	591,3	12491	30085	20297,9	12589,9	113	114	113,2
VND	100	161	596	272,9	364,2	2218	10997	4354,9	7481,4	184	191	186,3
V_1: First Imp. Best Fit	150	62	213	106,6	122,0	574	3417	1254,7	2426,9	282	289	282,9
V_2, V_3 e V_4: Best Fit	200	31	102	50,3	58,6	209	1143	439,0	791,3	380	385	381,9
	Total acumulado em todas as instâncias:									**959**	**979**	**964,3**
	50	554	1436	1087,5	629,6	11998	29970	20825,7	13173,5	113	114	113,2
VND	100	141	562	251,6	351,3	2007	10708	4121,3	7212,5	185	191	187,3
V_1: First Imp. Best Fit	150	56	208	103,0	120,1	497	3302	1341,1	2363,7	282	289	284,7
V_2, V_3 e V_4: Best EM	200	32	103	50,8	59,3	208	1144	439,9	793,9	379	387	384,4
	Total acumulado em todas as instâncias:									**959**	981	969,9

*Valores em **negrito** destacam a estratégia selecionada e o melhor acúmulo total de caixas.

4.3.1.3 Função Objetivo

O último componente testado no GRASP proposto foi a utilização da função objetivo alternativa (FOA), detalhada na Seção 3.1. Todas os resultados exibidos nas seções anteriores para o GRASP utilizaram a função objetivo trivial (FOT), assim, esta seção objetiva avaliar o impacto do FOA nesta heurística. O FOA é uma função objetivo que adiciona uma fração ao número de caixas utilizadas, referente à caixa com menor carga presente na solução. Desta maneira, cada instância foi executada 10 vezes com uma implementação do GRASP-VND com o FOA em conjunto com as estratégias selecionadas nas seções anteriores: parâmetro $\alpha = 0{,}2$, fase de construção com empacotamento Best EM, permutações que empacotam itens ordenados por maior volume, VND com busca local *First Improvement* e reempacotamento Best Fit. O critério de parada foi atingir

Tabela 14 – Resultado da última bateria de testes da fase de melhora do GRASP. Comparação entre diferentes combinações de vizinhanças no VND.

Vizinhança Utilizadas	Total Caixas
V_1, V_2, V_3 e V_4	**964,3**
V_1, V_2	968,2
V_1, V_3	968,3
V_1, V_4	969,1
V_1, V_2 e V_3	965,3
V_1, V_2 e V_4	966,1
V_1, V_3 e V_4	966,0

*Valores em **negrito** destacam a combinação selecionada e o melhor acúmulo total de caixas.

10 segundos de tempo de execução em CPU. A Tabela 15 reporta os resultados obtidos, no mesmo formato que as tabelas apresentadas na seção anterior. Os resultados desta tabela devem ser comparados com os resultados do GRASP-VND Best Fit destacado em negrito na Tabela 13, que por sua vez apresenta o resultado da mesma implementação do GRASP-VND, mas com o FOT.

Quando comparamos o resultado das tabelas, observamos que o número de iterações diminui e a quantidade de avaliações cresce com a utilização do FOA. Isto acontece devido ao método de reempacotamento: são geradas inúmeras soluções com a mesma quantidade de caixas da solução atual, portanto, com o FOT, tais soluções são equivalentes, mas não são necessariamente iguais com o FOA. Os métodos de reempacotamento descartam soluções equivalentes, assim, a implementação do FOA no algoritmo causa um número menor de desistências. Um número maior de reempacotamentos leva ao decréscimo do número de iterações, pois o tempo de execução de uma iteração aumenta, analogamente ao comportamento descrito na seção anterior. A quantidade de avaliações aumenta devido à facilidade de criar novas soluções no reempacotamento comparado com configurações criadas desde o princípio na heurística construtiva. Então, o desempenho da heurística com o FOA é menor do que a mesma implementação utilizando o FOT. Os resultados em termos de qualidade da solução obtida mostram que o GRASP-VND com FOA é melhor do que a mesma versão com o FOT nas instâncias utilizadas, mesmo com um número menor de iterações. Assim, o FOA é a função objetiva selecionada para o GRASP-VND. Finalmente, a Tabela 16 resume a configuração final do método com as estratégias selecionadas nesta seção.

Tabela 15 – Resultado da bateria de teste com o GRASP-VND utilizando o FOA.

Método	n	Núm. de Iterações				Núm. de Avaliações				Z		
		Min	Máx	Med	Desvio	Min	Máx	Med	Desvio	Min	Máx	Med
GRASP-VND FOA	50	257	489	347,5	182,9	39283	212026	114265,5	124821,6	112	114	112,9
	100	34	102	54,5	54,1	24309	145085	68682,3	92670,0	182	184	182,9
	150	10	36	18,4	19,9	16058	110035	54277,1	73271,2	280	282	281
	200	7	17	9,7	8,4	14243	84343	47011,2	53851,4	378	382	379,4
	Total acumulado em todas as instâncias:									**952**	**962**	**956,2**

*Valores em **negrito** destacam a estratégia selecionada e o melhor acúmulo total de caixas.

Tabela 16 – Configuração final do GRASP proposto. Estratégias selecionadas para cada fase e componente do método.

GRASP-VND	
Fase de Construção	Permutações ordenadas por maior volume Empacotamento Best EM
Fase de Melhora	VND V_1 : *First Improvement* e Best Fit V_2, V_3 e V_4 : Best Fit
Função Objetivo	FOA

4.3.2 Busca Tabu (BT)

O algoritmo BT, detalhado na Seção 3.9, para o problema de empacotamento é uma heurística iterativa que recebe uma solução inicial como parâmetro de entrada. As ite-

rações do método consistem de um processo de busca, o qual realiza diversos reempacotamentos com o auxílio de uma memória adaptativa que guarda soluções. O BT-VN é uma versão modificada desta heurística que utiliza as vizinhanças propostas na Seção 3.7 durante o processo de busca. A solução inicial é sintetizada pela heurística construtiva proposta na Seção 3.5. Um dos parâmetros que afeta o desempenho e a qualidade dessas soluções iniciais é a sequência que estabelece a ordem de empacotamento dos itens, determinada por uma permutação. Consequentemente, a ordem de empacotamento também afeta o desempenho e a qualidade das soluções finais encontradas pelo BT e o BT-VN. Desta maneira, o objetivo desta seção é selecionar o tipo de permutação e os critérios de avaliação dos EM que devem ser utilizadas nestas heurísticas. Além disso, objetiva-se também comparar a versão BT com o BT-VN e determinar qual função objetivo deve ser aplicada.

4.3.2.1 Solução Inicial

A solução inicial é construída através da heurística construtiva proposta na Seção 3.5. Essa heurística é um método que empacota um item a cada iteração, tal que a ordem dos itens é dada por uma permutação passada como parâmetro de entrada. Cada item é empacotado no melhor EM, que é selecionado através de um dos seguintes critérios de avaliação: Best Fit, Best EM ou DFTRC detalhados nas Seções 3.5.1 e 3.5.2, respectivamente. Três tipos de permutações foram considerados: completamente aleatórias, permutações que empacotam os itens priorizando o maior volume e permutações que ordenam os itens por menor volume. Com o DFTRC, se a mesma permutação for utilizada em diferentes execuções para a mesma instância, então a mesma solução é construída. Quando utilizamos o Best Fit ou Best EM, execuções com sequências de empacotamento iguais geram soluções equivalentes com alta probabilidade. Portanto, sabendo que todas as implementações deste trabalho utilizam sementes sequenciais em cada execução para o gerador de números pseudoaleatórios, a seguinte regra foi implementada para os tipos de permutação que empacotam sequências ordenadas de itens:

- Se a semente for 1, então a permutação é a própria sequência ordenada.
- Caso contrário, dois elementos da permutação ordenada são trocados aleatoriamente $\frac{n}{3}$ vezes, onde n é o número de itens da instância.

Desta maneira, quando a semente é diferente de 1, os dois últimos tipos de permutações são sequências compostas por várias subsequências ordenadas de itens. Nove implementações foram comparadas: uma para cada combinação de critério de avaliação e tipo de permutação. Para cada implementação e instância 100 execuções foram feitas com sementes sequenciais inteiras no intervalo $[1, 100]$ para o gerador de números pseudoaleatórios. Para manter a comparação entre os métodos justa, o mesmo conjunto de permutações foi utilizado para as execuções com as diferentes implementações. A Tabela 17 apresenta o resultado dessa bateria de testes, relacionando a configuração implementada com o acúmulo de caixas obtido em todas as instâncias. O resultado para cada instância é a média das execuções.

Quando utilizamos permutações aleatórias ou parcialmente ordenadas pelo maior volume, a tabela mostra que o Best EM é o critério que obtém o melhor acúmulo total em todas as instâncias, seguido do Best Fit e por último o DFTRC. Quando a sequência de empacotamento é parcialmente ordenada pelo menor volume, o DFTRC obtém resultados melhores que o Best EM seguido pelo Best Fit. Além disso, os dados mostram que

Tabela 17 – Comparação entre as configurações da heurística construtiva para a sintetização da solução inicial do BT.

Tipo de Permutação	Empacotamento	Total de Caixas
Completamente Aleatória	Best Fit	1051,6
	Best EM	1045,4
	DFTRC	1167,2
Parcialmente ordenado por maior volume	Best Fit	977,0
	Best EM	**975,9**
	DFTRC	1024,0
Parcialmente ordenado por menor volume	Best Fit	1151,2
	Best EM	1114,6
	DFTRC	1051,4

*Valores em **negrito** destacam a estratégia selecionada e o melhor acúmulo total.

a estratégia de ordenar parcialmente os itens por maior volume causa um impacto positivo na qualidade final das soluções geradas. Enquanto a ordenação por menor volume somente melhora as soluções da implementação com o DFTRC. Assim, o Best EM é o critério selecionado para a construção da solução inicial em conjunto com permutações que empacotam itens parcialmente ordenados por maior volume.

Sobre o tempo de execução em CPU, não houve diferença significativa entre o desempenho das heurísticas quando diferentes critérios foram utilizados. Porém, a ordenação parcial dos itens por maior volume diminui consideravelmente o tempo de execução necessário para construir as soluções, enquanto a ordenação parcial por menor volume afeta negativamente a performance do algoritmo.

4.3.2.2 Processo de Busca

O processo de busca do BT consiste de diversas iterações que esvaziam caixas abertas da solução atual, duas de cada vez, para realizar um reempacotamento com os itens que existiam nelas. Como o critério selecionado para a construção da solução inicial é o Best EM, este método de reempacotamento não pode utilizar o DFTRC, devido a incompatibilidade da escolha do canto de inserção no momento da criação do EM. Desta maneira, a primeira bateria de testes desta seção tem por objetivo selecionar o Best Fit ou Best EM para os reempacotamentos realizados durante as iterações do método. A heurística BT possui um parâmetro calibrável, denominado T_{max}, que restringe o número máximo de soluções tabu. Durante os testes desta seção, o valor usado para T_{max} foi 10, a fim de manter a justiça na comparação. Este valor foi selecionado baseado nos testes empíricos realizados durante a implementação inicial do algoritmo para este trabalho. O parâmetro é devidamente calibrado na Seção 4.4.

A primeira bateria de testes utiliza duas implementações do BT: uma com o reempacotamento Best Fit e a outra com o Best EM. Cada instância foi executada 10 vezes com cada implementação. A configuração utilizada na geração da solução inicial são as estratégias selecionadas na seção anterior: empacotamento Best EM e permutações que codificam sequências de itens parcialmente ordenadas por maior volume. O critério de parada foi atingir 10 segundos de tempo de execução em CPU, e o resultado para cada instância é a média das execuções. A Tabela 18 exibe os resultados da primeira bateria de testes do BT, relacionando o método utilizado com o mínimo, máximo e média do

acúmulo Z de caixas, o número de iterações e a quantidade de avaliações. Além disso, a tabela exibe o desvio padrão para os dois últimos resultados mencionados. Os valores de todos os resultados presentes em cada linha da tabela são referentes aos subconjuntos de instâncias que possuem n itens, relacionados na coluna 2.

Os dados obtidos na primeira bateria de testes mostram que o Best Fit obtém soluções equivalentes ou melhores que o Best EM no método de reempacotamento: o acúmulo de caixas mínimo, máximo e médio em todos os subconjuntos de instâncias é menor utilizando o critério Best Fit. A tabela também mostra que o número de iterações e a quantidade de avaliações é similar em ambos os critérios, o que significa que, em termos de desempenho, os dois critérios são equivalentes. Portanto, o critério selecionado para o reempacotamento durante o processo de busca do BT é o Best Fit. Além disso, podemos observar na tabela que, em alguns casos, o número de avaliações é menor que o número de iterações. Isto acontece por causa do alto número de desistências durante o reempacotamento dos itens: diversas soluções encontradas possuem o mesmo número de caixas comparado com a solução atual devido à natureza do problema, principalmente em instâncias com uma baixa quantidade de itens ($n \leq 50$).

A segunda bateria de testes foi executada para comparar as versões BT e BT-VN. O segundo algoritmo é uma versão alternativa da busca tabu proposta que utiliza as vinhanças V_2, V_3 e V_4, detalhadas na Seção 3.7, para amplificar e diversificar o processo de busca. Essas vizinhanças efetuam o reempacotamento dos itens. Entretanto, elas esvaziam metade das caixas da solução atual. As iterações do BT-VN executam as três vizinhanças antes de prosseguir com o mesmo processo de busca do BT. Além disso, o BT-VN usa V_4, caso o critério de aspiração da busca tabu seja atendido. O método de reempacotamento de cada vizinhança pode utilizar o Best Fit ou Best EM, por causa da compatibilidade com estratégia Best EM selecionada para a geração da solução inicial. Portanto, duas implementações do BT-VN foram testadas: a primeira utiliza o método Best Fit para o reempacotamento de cada vizinhança e a outra usa o Best EM. Cada instância foi executada 10 vezes para cada implementação. O critério de parada foi atingir 10 segundos de tempo de execução em CPU, e o reempacotamento das iterações, que esvaziam um par de caixas de cada vez, foi efetuado através do Best Fit. A Tabela 19 exibe os resultados desta bateria de testes com a mesma disposição da tabela anterior.

A tabela mostra que o resultado médio obtido pelo BT-VN para o número de iterações e a quantidade de avaliações em todos os subconjuntos de instâncias são similares aos valores observados na Tabela 18 para o BT com Best Fit. Isto significa que a utilização das vizinhanças não causa um grande impacto no tempo de execução durante as iterações da heurística: somente três reempacotamentos são introduzidos na versão BT-VN por iteração. Além disso, estes resultados também mostram que ambos os critérios, Best Fit e Best EM, têm desempenho equivalente no método de reempacotamento das vizinhanças, por causa das diferenças observadas na tabela referente ao máximo, mínimo e média. Em relação ao acúmulo total de caixas, o BT-VN, em conjunto com o reempacotamento Best Fit para as vizinhanças, gera os melhores resultados. Então, o BT-VN com esta configuração foi a estratégia selecionada para os testes computacionais que seguiram neste trabalho.

A terceira bateria de testes dessa seção tem por objetivo avaliar o impacto de cada vizinhança usada no BT-VN, a fim de selecionar a combinação que gera o melhor resultado para o conjunto de instâncias testadas. Desta maneira, o BT-VN foi executado com cada uma das 6 possíveis combinações distintas de vizinhanças em cada instância 10 vezes, onde o limite de tempo de execução em CPU foi 10 segundos. A Tabela 20 exibe o

acúmulo de caixas obtido em todas as instâncias, onde o resultado para cada uma é a média das execuções. A tabela mostra que as vizinhanças V_2 e V_3 são as que causam o maior impacto na qualidade final das soluções, porém todas melhoram o resultado. Podemos observar também que os resultados obtidos com as implementações que utilizam qualquer combinação de 2 vizinhanças são melhores do que as que usam somente uma única vizinhança. Finalmente, a tabela mostra que a utilização de todas as vizinhanças gera os melhores resultados para as instâncias testadas.

Vale a pena ressaltar que a qualidade do resultado final não é afetado pela ordem em que as vizinhanças são executadas no BT-VN, porém, o algoritmo tem um desempenho melhor se V_2 e V_3 forem executadas antes de V_4.

Além disso, o autor desta dissertação considerou a implementação de diferentes estratégias, por exemplo, implementar um critério misto baseado na escolha aleatória com peso entre o Best EM e o Best Fit durante os métodos de empacotamento e reempacotamento, porém estas estratégias não melhoraram a qualidade final das soluções obtidas.

Tabela 18 – Resultado da primeira bateria de testes da calibração do processo de busca do BT. Comparação entre os critérios para o método de reempacotamento.

Método	n	Núm. de Iterações				Núm. de Avaliações				Z		
		Min	Máx	Med	Desvio	Min	Máx	Med	Desvio	Min	Máx	Med
BT Best Fit	50	7038383	22931177	14056252,9	14661239,1	34484	257566	156228,2	218723,9	112	114	113,3
	100	422	17663041	2944428,9	16123431,5	13396	194850	98049,1	191939,0	183	186	183,7
	150	59	509	214,6	343,1	14497	173089	77039,6	134616,9	281	288	283,9
	200	30	292	115,9	203,4	7087	160971	71226,8	145900,8	380	387	383,1
	Total acumulado em todas as instâncias:									**956**	**975**	**964,0**
BT Best EM	50	8728364	20451547	13916419.5	8546474,4	23921	438887	147937,1	369616,4	113	114	113,1
	100	491	1354137	3277131,0	11973223,9	23350	205573	92794,0	168311,3	182	188	184,8
	150	86	449	238,7	309,9	15141	199077	85103,8	167148,4	283	289	285,1
	200	49	266	122,7	182,9	12703	168789	72929,5	150543,5	379	387	384,4
	Total acumulado em todas as instâncias:									957	978	967,4

*Valores em **negrito** destacam a estratégia selecionada e o melhor acúmulo total de caixas.

Tabela 19 – Resultado da segunda bateria de testes da calibração do processo de busca do BT. Comparação entre as versões BT e BT-VN.

Método	n	Núm. de Iterações				Núm. de Avaliações				Z		
		Min	Máx	Med	Desvio	Min	Máx	Med	Desvio	Min	Máx	Med
BT-VN Best Fit **V_2, V_3, V_4: Best Fit**	50	8728364	20451547	13916419,5	8546474.4	19977	438887	131455,6	388238,7	112	114	113,5
	100	400	13992121	2856103,1	1523409,2	9536	205573	91669,9	189646,6	182	184	182,9
	150	81	451	214,6	309,9	2526	199077	74865,1	186650,8	281	282	281,7
	200	50	291	115,9	182,9	1942	168789	72070,5	161594,9	379	379	379
	Total acumulado em todas as instâncias:									**954**	**959**	**957,1**
BT-VN Best Fit V_2, V_3, V_4: Best EM	50	3855081	10619610	12984990,2	7813672,7	18726	3866398	129541,6	339315,2	113	114	113,1
	100	421	9279282	2847102,1	1470178,5	7007	278774	95232,8	159858,5	182	188	184,7
	150	58	503	210,7	342,2	2864	231263	79251,2	183774,2	281	289	284,2
	200	41	309	122,5	218,1	1652	195254	71574,0	161843,3	379	387	383,2
	Total acumulado em todas as instâncias:									955	978	965,2

*Valores em **negrito** destacam a estratégia selecionada e o melhor acúmulo total de caixas.

4.3.2.3 Função Objetivo

O objetivo desta seção é apresentar uma bateria de testes que compara a utilização do FOT com o FOA no BT-VN. O FOA é uma função objetivo que utiliza uma fração no intervalo $[0, 1[$ baseada na caixa com menor carga da solução, a fim de diferenciar a qualidade de soluções que utilizam o mesmo número de caixas. A bateria de testes consistiu em executar, 10 vezes para cada instância, o BT-VN com as configurações selecionadas nas seções anteriores em conjunto com o FOA. O critério de parada utilizado foi atingir

Tabela 20 – Resultados obtidos com diferentes combinações de vizinhanças no BT-VN.

Vizinhança Utilizadas	Total Caixas
V_2, V_3 e V_4	**957,1**
V_2, V_3	960,0
V_2, V_4	961,1
V_3, V_4	961,0
V_2	962,3
V_3	962,4
V_4	963,5

*Valores em **negrito** destacam a combinação selecionada e o melhor acúmulo total de caixas.

10 segundos de tempo de execução em CPU, e o resultado é a média das execuções. A Tabela 21 exibe os resultados dessa bateria de testes. As diferenças observadas entre esta tabela e a Tabela 19 para o número de iterações mínimo, máximo e médio do BT-VN selecionado mostram que o desempenho da heurística com o FOA é equivalente à mesma versão com o FOT. Porém, as diferenças observadas para a quantidade de avaliações mostram que o BT-VN com FOA aceita um número maior de soluções. Além disso, o acúmulo total mínimo, máximo e médio do BT-VN com FOA mostra que esta implementação gera soluções melhores ou equivalentes comparadas com as obtidas pela implementação que utiliza o FOT para as mesmas instâncias. Assim, o FOA é a função objetivo selecionada para a versão final do BT-VN. A Tabela 22 resume a configuração final da heurística, exibindo as estratégias selecionadas nesta seção.

Tabela 21 – Resultado da bateria de testes referente à seleção da função objetivo. Comparação entre a FOT e FOA.

Método	n	Núm. de Iterações				Núm. de Avaliações				Z		
		Min	Máx	Med	Desvio	Min	Máx	Med	Desvio	Min	Máx	Med
	50	6110526	20587868	1375739,8	5504423,8	100326	31769312	11657317,1	33968837,4	111	114	113,6
BT-VN Best Fit	100	421	149187363	2797168,8	1956110,5	28768	27599619	3535014,3	25726466,9	182	184	182,4
V_2, V_3, V_4: Best Fit	150	57	213	183,6	289,1	23517	226053	98150,8	178756,2	279	281	280,5
FOA	200	48	315	125,4	102,9	16061	194610	86569,8	165394,4	376	380	378,8
	Total acumulado em todas as instâncias:									**948**	**959**	**955,3**

*Valores em **negrito** destacam a estratégia selecionada e o melhor acúmulo total de caixas.

Tabela 22 – Configuração final da Busca Tabu proposta. Estratégias selecionadas para cada fase e componente.

BT-VN	
Solução Inicial	Permutações parcialmente ordenadas por maior volume Empacotamento Best EM
Processo de Busca	VN Reempacotamento Best Fit V_2, V_3 e V_4 : Best Fit
Função Objetivo	FOA

4.3.3 Algoritmo Genético de Chaves Aleatórias Viciadas (BRKGA)

O BRKGA, proposto na Seção 3.10, é um algoritmo evolutivo que utiliza números reais no intervalo $[0, 1]$, denominados *random-keys*, para codificar os indivíduos. Este

método tem quatro parâmetros calibráveis e um componente chamado decodificador. Este decodificador utiliza uma variante da heurística construtiva, que interrompe a busca por EM em outras caixas abertas da solução atual quando pelo menos uma localização válida é encontrada. O BRKGA-VCD é uma versão alternativa que utiliza o método de reprodução durante a geração da população inicial e mutantes. O VCD é inspirado no VND e tem uma alta probabilidade de gerar indivíduos que codificam sequências parcialmente ordenadas de itens por maior volume ou dimensão. Esta seção objetiva selecionar um critério de empacotamento para o decodificador, uma estratégia para a geração das populações e a melhor função objetivo para a versão final do método.

4.3.3.1 Decodificador

O decodificador é um componente utilizado para avaliar um indivíduo. A entrada é um conjunto de *random-keys* e, a saída, uma solução com todos os itens empacotados. O algoritmo que realiza o empacotamento é uma variante da heurística construtiva proposta, a qual empacota cada item na melhor EM da primeira caixa aberta que possui uma localização válida para ele. Assim, o objetivo desta seção é selecionar um dos seguintes critérios de avaliação de EM para o decodificador: Best Fit, Best EM ou DFTRC.

Uma bateria de testes foi executada com três implementações do BRKGA sem o módulo VCD, uma para cada critério de avaliação durante a decodificação. O critério de parada foi atingir 10 segundos de tempo de execução em CPU. Cada implementação executou 10 vezes para cada instância com sementes distintas no intervalo $[1, 10]$ para o gerador de números pseudoaleatórios. O resultado mostrado para cada instância é a média das execuções. Para manter a comparação justa, os seguintes valores foram utilizados para os parâmetros calibráveis em todas as implementações: fração TOP $\epsilon = 0{,}15$, fração BOT $\omega = 0{,}10$, taxa de reprodução elitista $\rho = 0{,}70$ e quantidade de indivíduos na população $P_{max} = 30 \times n$, onde n é a quantidade de itens a serem empacotados. Estes são os valores utilizados no trabalho (ZUDIO et al., 2018), com as mesmas instâncias desta dissertação para os experimentos computacionais. A Tabela 23 apresenta o resultado dessa bateria de testes com a mesma disposição das tabelas apresentadas nas Seções 4.3.1 e 4.3.2.

A tabela mostra que as implementações têm desempenho equivalente, por causa das diferenças observadas na tabela para o número de iterações mínima, máxima e média. No BRKGA, a quantidade de avaliações é fixa por iteração. Portanto, as diferenças observadas na tabela referentes a esses valores também indicam que os critérios são equivalentes em termos de desempenho. Porém, os valores mínimo, máximo e médio para o acúmulo total de caixas mostram que as soluções obtidas para este conjunto de instâncias pela implementação DFTRC são melhores ou equivalentes que as outras versões. Assim, o DFTRC foi o critério selecionado para o método de empacotamento do decodificador.

4.3.3.2 População Inicial e Mutantes

Antes de executar o processo iterativo, o BRKGA gera uma população inicial composta por indivíduos aleatórios. Durante as gerações, uma fração da nova população é de mutantes, que é gerada da mesma maneira que a população inicial, ou seja, indivíduos aleatórios. O BRKGA-VCD é uma versão que aplica o método de reprodução com alguns indivíduos especiais, chamados de Universais, que são sintetizados através da ordenação dos itens por maior volume ou dimensão, para gerar a população inicial e mutantes. O objetivo do VCD é criar indivíduos que codificam soluções que empacotam os itens em ordem parcialmente ordenada por maior volume ou maior dimensão. Tais indivíduos têm

Tabela 23 – Resultado da bateria de testes da calibração do decodificador. Comparação entre os critérios para o empacotamento.

Método	n	Núm. de Iterações				Núm. de Avaliações				Z		
		Min	Máx	Med	Desvio	Min	Máx	Med	Desvio	Min	Máx	Med
BRKGA Best Fit	50	72	241	126,8	135,9	108000	361500	193150,6	205658,8	113	113	113,0
	100	15	41	26,0	21,1	45000	123000	81233,8	56758,3	184	187	186,6
	150	6	17	13,3	13,7	27000	76500	61350,0	41225,0	285	289	287,1
	200	2	9	7,3	7,8	12000	54000	44625,0	39832,1	388	392	389,9
	Total acumulado em todas as instâncias:									970	981	976,6
BRKGA Best EM	50	81	250	122,3	131,1	121500	375000	185401,8	199817,4	112	113	112,8
	100	15	49	23,4	19,4	45000	147000	82301,6	51789,4	187	188	187,1
	150	7	19	14,0	12,9	31500	85500	60997,0	45606,3	289	292	290,8
	200	2	8	7,1	7,1	12000	48000	42587,6	41247,5	378	382	380,0
	Total acumulado em todas as instâncias:									966	975	970,7
BRKGA DFTRC	50	79	247	131,8	136,2	118500	370500	189547,2	211136,0	112	113	112,6
	100	13	54	26,7	16,9	39000	162000	78305,9	55782,0	187	187	187,0
	150	7	17	13,8	12,4	31500	76500	62378,9	43187,5	287	289	288,0
	200	2	10	7,5	6,9	12000	60000	42831,5	38874,5	375	379	376,9
	Total acumulado em todas as instâncias:									**961**	**968**	**964,5**

*Valores em **negrito** destacam a estratégia selecionada e o melhor acúmulo total de caixas.

alta probabilidade de codificar uma solução de alta qualidade. Desta maneira, esta seção objetiva selecionar a estratégia que deve ser utilizada durante a geração da população inicial e mutantes através da comparação entre o BRKGA e o BRKGA-VCD.

O BRKGA-VCD com o DFTRC foi executado 10 vezes para cada instância com os mesmos valores utilizados na seção anterior para os parâmetros calibráveis: $\epsilon = 0{,}15$, $\omega = 0{,}10$, $\rho = 0{,}70$ e $P_{max} = 30 \times n$, onde n é a quantidade de itens a serem empacotados. Sementes inteiras distintas no intervalo $[1, 10]$ foram usadas para o gerador de números pseudoaleatórios, a fim de controlar o ambiente de execução. O resultado para cada instância é a média das execuções, e o critério de parada foi atingir 10 segundos de tempo de execução em CPU. A Tabela 24 exibe os resultados dessa bateria de testes.

Comparando os resultados da tabela com os dados exibidos na Tabela 23 para o BRKGA com DFTRC, podemos observar que as diferenças entre os mínimos, máximos e as médias para o número de iterações e avaliações é pequena. Portanto, o BRKGA e o BRKGA-VCD têm desempenho equivalente. Isto se deve ao fato de que o VCD realiza o mesmo número de empacotamentos que o método aleatório para gerar a população inicial e mutantes. Os acúmulos de caixas mínimo, máximo e médio obtido pelo BRKGA para o conjunto de instâncias com $n = 200$ foi melhor que os acúmulos obtidos pelo BRKGA-VCD para estas instâncias, mas os demais valores, incluindo o acúmulo total, mostram que o BRKGA-VCD obtém soluções melhores que o BRKGA nos outros conjuntos de instâncias. Desta maneira, o VCD é a estratégia selecionada para a geração da população inicial e mutantes.

4.3.3.3 Função Objetivo

Essa seção avalia o impacto do FOA no BRKGA-VCD, a fim de compará-la com o FOT para versão final da heurística. O FOA, proposto na Seção 3.1, é uma função objetivo que adiciona uma fração ao número de caixas utilizadas no momento em que a heurística deve avaliar a solução. O calculo é baseado na caixa de menor carga presente na solução avaliada.

A implementação do BRKGA-VCD com o FOA utilizou a configuração selecionada

Tabela 24 – Resultado da bateria de testes da calibração do método para a geração da população inicial e mutantes. Comparação entre o BRKGA e o BRKGA-VCD.

Método	n	Núm. de Iterações				Núm. de Avaliações				Z		
		Min	Máx	Med	Desvio	Min	Máx	Med	Desvio	Min	Máx	Med
BRKGA-VCD DFTRC	50	75	247	122,7	130,9	112500	370500	183434,7	200001,9	111	112	111,9
	100	14	47	22,5	20,3	42000	141000	79654,1	54781,2	182	182	182,0
	150	6	18	12,9	12,4	27000	81000	63544,1	42578,3	280	282	281,2
	200	2	9	6,9	7,1	12000	54000	40783,2	39751,0	379	380	379,9
	Total acumulado em todas as instâncias:									**952**	**956**	**955,0**

*Valores em **negrito** destacam a estratégia selecionada e o melhor acúmulo total de caixas.

nas seções anteriores e foi executada 10 vezes para cada instância. Os valores utilizados para os parâmetros calibráveis são os mesmos usados na seção anterior, com o objetivo de manter a justiça durante a comparação. O resultado de cada instância é a média das execuções. A Tabela 25 apresenta os resultados obtidos nesta bateria de testes, exibindo a mesma disposição das tabelas anteriores.

Os valores observados na tabela são comparados com os resultados apresentados na Tabela 24 referentes à implementação do BRKGA-VCD com empacotamento DFTRC em conjunto com a avaliação pelo FOT. As diferenças observadas para os valores mínimo, máximo e médio em todos os conjuntos de instâncias para a quantidade de iterações e o número de avaliações mostram que as implementações têm desempenho equivalente. Porém, os valores para o acúmulo de caixas mínimo, máximo e médio em todos os conjuntos de instâncias são menores quando utilizamos a implementação com o FOA. Portanto, esta implementação obtém soluções melhores ou equivalentes do que a mesma versão com o FOT para o conjunto de instâncias usado. Desta maneira, a função objetivo selecionada para a versão final do BRKGA-VCD é o FOA. A Tabela 26 resume a configuração final da heurística, apresentando as estratégias selecionadas nesta seção.

Tabela 25 – Resultado da bateria de testes para a seleção da função objetivo. Comparação entre o FOT e FOA.

Método	n	Núm. de Iterações				Núm. de Avaliações				Z		
		Min	Máx	Med	Desvio	Min	Máx	Med	Desvio	Min	Máx	Med
BRKGA-VCD DFTRC FOA	50	69	245	120,2	124,8	103500	367500	194132,1	194333,2	111	111	111
	100	17	48	23,8	19,0	42000	141000	79457,7	59911,0	182	182	182,0
	150	9	15	12,0	10,5	40500	67500	64421,9	43477,1	278	279	278,6
	200	2	8	6,9	7,4	12000	48000	38864,4	38432,4	376	378	377,5
	Total acumulado em todas as instâncias:									**947**	**950**	**949,1**

*Valores em **negrito** destacam a estratégia selecionada e o melhor acúmulo total de caixas.

Tabela 26 – Configuração final do BRKGA proposto. Estratégias selecionadas para cada componente.

BRKGA-VCD	
Decodificador	Empacotamento DFTRC
População Inicial e Mutantes	VCD
Função Objetivo	FOA

4.4 **Calibração**

Esta seção apresenta os resultados de uma calibração feita para ajustar os valores dos parâmetros de entrada do GRASP-VND e do BT-VN com as respectivas configurações finais. A calibração foi efetuada empiricamente através da ferramenta I-Race (LÓPEZ-IBÁÑEZ et al., 2016).

As heurísticas GRASP, BT e BRKGA, propostas no Capítulo 3, são algoritmos que utilizam parâmetros calibráveis que afetam o desempenho e a qualidade da solução final do algoritmo. Por exemplo, um dos parâmetros de entrada do BT é a quantidade máxima de soluções tabu que a memória adaptativa pode ter durante qualquer iteração. Assim, o objetivo desta seção é calibrar as heurísticas GRASP-VND e BT-VN através de uma ferramenta chamada *irace*[2]. O BRKGA-VCD não foi calibrado desta maneira, pois os trabalhos (ZUDIO et al., 2018; GONÇALVES; RESENDE, 2013) já fornecem um conjunto de valores para os parâmetros calibráveis que geram resultados de alta qualidade para este método quando executado com o conjunto de instâncias detalhadas na Seção 4.1.

O *irace* é uma ferramenta implementada na linguagem de programação *R* que executa o *iterated racing procedure*, que, por sua vez, é uma extensão do *iterated F-race procedure*. O propósito da ferramenta é encontrar automaticamente as melhores configurações de parâmetros para algoritmos de otimização. A ferramenta fornece diversos procedimentos que são utilizados com sucesso na configuração de vários algoritmos no estado da arte. Para sua execução, o usuário precisa fornecer instâncias de treino, um orçamento B, o próprio algoritmo A de otimização que será calibrado, um conjunto de especificações para os parâmetros que serão calibrados e um *script* que a ferramenta usa para executar A.

O orçamento B é um número inteiro que serve como critério de parada para a ferramenta. Ele dita o número máximo de execuções que a ferramenta pode realizar com A. Geralmente, o algoritmo A é uma aplicação que tem um critério de parada, por exemplo, atingir um determinado tempo de execução em CPU. Desta maneira, o algoritmo implementado pelo *irace* não é tão custoso em termos de tempo de execução em CPU comparado com A. Desta forma, o critério de parada é baseado em B. Alternativamente, o usuário pode fornecer um alvo máximo de tempo de execução para a ferramenta, desativando opcionalmente o orçamento B. O conjunto de especificações para os parâmetros é um arquivo de texto que relaciona o nome, tipo e os possíveis valores que cada um pode assumir. Opcionalmente, o usuário pode especificar um conjunto de regras lógicas para cada variável, onde certos valores podem ser excluídos ou obrigatoriamente utilizados em certas condições. Além disso, o usuário pode fornecer opcionalmente uma configuração inicial para as variáveis. (LÓPEZ-IBÁÑEZ et al., 2016)

As instâncias de treino utilizadas na calibração do GRASP-VND e do BT-VN são todas as tridimensionais, detalhadas na Seção 4.1.1, que têm uma quantidade alta de itens ($n \geq 150$). As heurísticas GRASP-VND e BT-VN foram configuradas para executar por 10 segundos de tempo de execução em CPU, e o orçamento B foi estabelecido como 2500. O *irace* foi configurado para utilizar o teste estatístico *Friedman*. Todos os parâmetros foram calibrados com 2 casas de precisão decimal, exceto T_{max} e P_{max} que são valores inteiros. A Tabela 27 exibe o resultado da calibração de cada método, relacionando a heurística com os seus parâmetros calibrados, os possíveis valores para cada parâmetro e as configurações iniciais. O resultado para o GRASP-VND e o BT-VN é a configuração que obteve o melhor ranque na ferramenta com as instâncias de treino mencionadas.

[2]Sítio eletrônico oficial do *irace*: <http://iridia.ulb.ac.be/irace/>

Tabela 27 – Resultado da calibração das heurísticas propostas para o 3BP e o 2BP.

Heurística	Parâmetros Calibrados	Possíveis Valores	Configuração Inicial
GRASP-VND	$\alpha = 0{,}56$	$]0, 1[$	$\alpha = 0{,}20$
BT-VN	$T_{max} = 10$	[5,40]	$T_{max} = 10$
BRKGA-VCD	$\epsilon = 0{,}15, \omega = 0{,}10, \rho = 0{,}70$ e $P_{max} = 30 \times n$		

*O BRKGA-VCD foi calibrado manualmente.

4.5 **Resultados Computacionais**

O objetivo desta seção é comparar as heurísticas GRASP-VND, BT-VN e BRKGA-VCD através de um teste com todas as 820 instâncias detalhadas na Seção 4.1, onde todos os itens possuem orientação fixa. O teste foi executado no ambiente descrito na Seção 4.2. Cada método foi executado 10 vezes com o mesmo critério de parada: 10 segundos de tempo de execução. Para controlar o âmbito de execução, sementes no intervalo $[1, 10]$ foram usadas para o gerador de números pseudoaleatórios. As configurações utilizadas na implementação de cada heurística consistem das estratégias selecionadas na Seção 4.3. Os valores utilizados para cada parâmetro calibrável foram os relacionados na Tabela 27.

A Tabela 28 mostra os resultados para as instâncias do 3BP, tal que a solução reportada é a média das execuções. Os valores descritos nas linhas são as médias das 10 instâncias da classe Cl para a quantidade de itens n relacionadas nas colunas 1 e 2. As colunas 3 e 4 são os resultados obtidos pelo GRASP/VND, onde Z é a média da qualidade obtida para o conjunto de instâncias e TM é a média do tempo total para encontrar a melhor solução. As colunas 5 até 8 fornecem as mesmas informações para o BT-VN e o BRKGA-VCD respectivamente. As duas últimas linhas destacam o acumulado total de caixas obtidos em cada método nas instâncias das classes $1, 4$ até 8 e todas respectivamente.

A tabela mostra que o BRKGA-VCD obtém resultado equivalente ou melhor do que o GRASP-VND e o BT-VN para todos os conjuntos de instâncias, com exceção de 3 casos. Em dois deles, o BRKGA-VCD perde por somente uma caixa no valor acumulado para as 10 instâncias do conjunto. A tabela também mostra que o BT-VN obtém soluções equivalentes ou melhores que o GRASP-VND para todas as instâncias. Os resultados referentes ao tempo necessário para encontrar a melhor solução mostram que os três métodos são capazes de produzir soluções de alta qualidade em pouco tempo de execução em CPU. Particularmente, as soluções encontradas para as instâncias da classe 4 são encontradas em aproximadamente 100 milissegundos. O teste de Wilcoxon pareado foi aplicado para decidir se as diferenças observadas na tabela são estatisticamente significantes. O nível de significância utilizado foi 5%. O teste mostra que o BRKGA-VCD é significativamente melhor que o BT-VN e o GRASP-VND, e que o BT-VN é melhor que o GRASP-VND.

A Tabela 29 exibe os resultados para as instâncias do 2BP com a mesma configuração da tabela anterior: os valores reportados em cada linha são as médias obtidas para o conjunto de 10 instâncias relacionadas pela coluna 1 e 2. Essas colunas relacionam a respectiva classe Cl e a quantidade de itens do conjunto. As colunas 3 e 4 apresentam os dados referentes ao GRASP/VND com a média de qualidade Z obtida para o conjunto e TM de tempo total para a melhor solução. As demais colunas reportam as mesmas informações para o BT-VN e o BRKGA-VCD. A última linha reporta o acumulado de caixas total obtido por cada heurística em todas as instâncias.

A tabela mostra que o BRKGA-VCD obtém resultados melhores ou equivalentes que o GRASP-VND e o BT-VN para todas as instâncias. A tabela também mostra que a maioria das soluções obtidas pelo BT-VN e o GRASP-VND são equivalentes. De acordo com o

acumulo total de caixas obtidas em todas as instâncias, o BT-VN economizou 3 caixas comparado com o GRASP-VND, e o BRKGA-VCD melhorou o resultado do BT-VN em 20 caixas. Os valores observados na tabela para TM sugerem que os métodos obtém as soluções finais para cada instância em poucas iterações e em baixo tempo de execução. O teste de Wilcoxon com significância 5% mostra que o BRKGA-VCD é significativamente melhor que o GRASP-VND e o BT-VN, o teste também diz que as diferenças observadas na tabela para os resultados do GRASP-VND e do BT-VN não têm diferença estatística.

Tabela 28 – Resultado obtido pelo GRASP-VND, BT-VN e BRKGA-VCD para todas as instâncias do 3BP sem rotação. O critério de parada foi atingir 10 segundos de tempo de execução em CPU.

		GRASP-VND		BT-VN		BRKGA-VND	
Cl	n	$TM(s)$	Z	$TM(s)$	Z	$TM(s)$	Z
1	50	2,5	**13,4**	0,7	**13,4**	0,1	**13,4**
	100	4,0	**26,6**	3,9	**26,6**	0,3	**26,6**
	150	4,8	36,8	5,9	36,5	2,1	**36,4**
	200	5,0	51,2	5,0	51,1	6,1	**51,0**
2	50	3,1	13,9	2,4	**13,8**	0,8	**13,8**
	100	3,8	25,7	3,1	25,6	0,3	**25,5**
	150	4,4	37,2	5,8	37,0	6,2	**36,7**
	200	4,9	50,1	5,2	49,8	4,7	**49,5**
3	50	2,5	13,4	0,3	**13,3**	0,1	**13,3**
	100	5,0	26,0	3,7	26,0	0,4	**25,9**
	150	4,6	37,8	5,8	37,6	8,1	**37,5**
	200	4,8	**50,1**	5,5	**50,1**	4,2	**50,1**
4	50	0,1	29,4	0,1	**29,4**	0,1	**29,4**
	100	0,1	**59,0**	0,1	**59,0**	0,1	**59,0**
	150	0,2	**86,8**	0,1	**86,8**	0,1	**86,8**
	200	0,1	**118,8**	0,1	**118,8**	0,1	**118,8**
5	50	1,0	**8,3**	0,1	**8,3**	0,2	**8,3**
	100	5,1	15,1	1,3	**15,0**	1,3	**15,0**
	150	4,4	20,6	3,8	**20,1**	0,9	**20,1**
	200	3,7	27,5	2,8	**27,2**	5,2	27,3
6	50	3,0	9,9	1,2	**9,8**	2,3	**9,8**
	100	3,5	19,0	0,8	19,0	0,9	**18,9**
	150	3,6	29,3	2,4	**29,2**	1,8	**29,2**
	200	4,4	37,6	2,9	37,6	3,4	**37,2**
7	50	1,5	**7,4**	0,2	**7,4**	1,7	**7,4**
	100	4,1	**12,3**	2,7	**12,3**	1,4	**12,3**
	150	4,3	15,5	4,2	15,5	3,8	**15,3**
	200	5,0	**23,4**	2,9	**23,4**	1,8	23,5
8	50	2,0	9,2	0,5	**9,2**	0,5	**9,2**
	100	3,0	**18,9**	1,6	**18,9**	0,7	**18,9**
	150	5,0	23,8	4,6	23,7	3,6	**23,6**
	200	5,0	29,7	3,6	**29,6**	4,8	30,1
(1, 4 – 8)			7295		7278		7275
(1 – 8)			9837		9810		9798

*Valores em negrito correspondem às melhores soluções de cada conjunto de instâncias

Tabela 29 – Resultado do GRASP-VND, BT-VN e BRKGA-VCD para todas as instâncias do 2BP sem rotação. O critério de parada foi atingir 10 segundos de tempo de execução em CPU.

		GRASP-VND		BT-VN		BRKGA-VCD	
Cl	n	$TM(s)$	Z	$TM(s)$	Z	$TM(s)$	Z
1	20	0,0	**7,1**	0,0	**7,1**	0,1	**7,1**
	40	1,3	**13,4**	0,1	**13,4**	0,4	**13,4**
	60	1,3	**20,0**	0,9	**20,0**	1,0	**20,0**
	80	1,4	**27,5**	1,5	**27,5**	1,2	**27,5**
	100	3,1	**31,7**	1,9	**31,7**	1,2	**31,7**
2	20	0,0	**1,0**	0,0	**1,0**	0,2	**1,0**
	40	2,8	2,0	0,1	2,0	5,7	**1,9**
	60	3,4	**2,5**	1,9	**2,5**	1,2	**2,5**
	80	4,3	**3,1**	1,9	**3,1**	1,2	**3,1**
	100	4,6	**3,9**	1,9	**3,9**	1,2	**3,9**
3	20	0,6	**5,1**	0,0	**5,1**	0,1	**5,1**
	40	1,9	**9,4**	0,8	**9,4**	0,6	**9,4**
	60	3,4	14,0	2,2	**13,9**	1,2	**13,9**
	80	3,3	**18,9**	0,4	19,1	1,2	**18,9**
	100	3,6	22,4	0,8	22,4	1,2	**22,3**
4	20	0,0	**1,0**	0,0	**1,0**	0,3	**1,0**
	40	0,7	**1,9**	0,2	**1,9**	0,9	**1,9**
	60	2,9	2,5	3,0	2,5	1,2	**2,4**
	80	5,0	3,2	4,5	3,2	4,1	**3,1**
	100	4,9	3,8	5,1	3,8	6,2	**3,7**
5	20	0,1	**6,5**	0,0	**6,5**	0,1	**6,5**
	40	1,4	**11,9**	0,3	**11,9**	0,5	**11,9**
	60	1,7	**18,0**	0,8	**18,0**	1,0	**18,0**
	80	1,5	**24,7**	0,5	**24,7**	1,2	**24,7**
	100	4,0	28,3	2,1	28,3	7,2	**28,1**
6	20	0,0	**1,0**	0,0	**1,0**	0,2	**1,0**
	40	1,6	1,9	1,0	1,9	1,0	**1,6**
	60	2,4	**2,1**	1,2	**2,1**	1,2	**2,1**
	80	6,1	**3,0**	5,3	**3,0**	4,2	**3,0**
	100	7,8	3,4	5,5	3,4	2,2	**3,2**
7	20	0,0	**5,5**	0,0	**5,5**	0,1	**5,5**
	40	2,9	11,2	1,2	**11,1**	0,5	**11,1**
	60	4,3	15,9	2,3	15,9	1,0	**15,8**
	80	3,6	23,2	2,7	23,2	1,2	**23,1**
	100	3,9	27,2	3,7	**27,1**	1,2	**27,1**
8	20	0,2	**5,8**	0,1	**5,8**	0,1	**5,8**
	40	1,8	**11,3**	0,7	**11,3**	0,6	**11,3**
	60	3,8	16,2	1,6	16,2	3,2	**16,1**
	80	2,8	**22,4**	2,7	**22,4**	2,2	**22,4**
	100	3,5	27,8	3,3	27,8	4,2	**27,7**
9	20	0,0	**14,3**	0,0	**14,3**	0,1	**14,3**
	40	0,0	**27,8**	0,0	**27,8**	0,4	**27,8**
	60	0,0	**43,7**	0,0	**43,7**	1,0	**43,7**
	80	0,0	**57,7**	0,0	**57,7**	1,2	**57,7**
	100	0,0	**69,5**	0,0	**69,5**	1,2	**69,5**
10	20	0,7	**4,2**	0,0	**4,2**	0,1	**4,2**
	40	2,6	**7,4**	0,3	**7,4**	0,6	**7,4**
	60	4,3	10,2	1,5	10,1	4,2	**10,0**
	80	4,9	12,9	1,1	12,8	2,2	**12,8**
	100	5,3	15,9	1,7	15,9	5,2	**15,8**
(1 – 10)			7253		7250		7230

*Valores em negrito correspondem às melhores soluções de cada conjunto de instâncias

4.6 **Comparação com Outras Abordagens da Literatura**

Com o objetivo de avaliar a qualidade das soluções obtidas pela melhor implementação deste trabalho, o BRKGA-VCD, o algoritmo foi executado novamente para as 320 instâncias do 3BP e as 500 do 2BP, detalhadas na Seção 4.1. Dois experimentos computacionais foram executados, onde o primeiro usa itens de orientação fixa, e o segundo utiliza itens que podem ser rotacionados. Desta vez, o critério de parada utilizado nas execuções foi atingir 200 gerações. A Seção 4.3 apresenta experimentos computacionais preliminares com o BRKGA-VCD, que selecionam as melhores estratégias para cada componente da heurística. Os parâmetros calibráveis foram manualmente calibrados e usados previamente no trabalho (ZUDIO et al., 2018), o qual apresenta resultados de alta qualidade para o conjunto tridimensional de instâncias utilizados neste trabalho. Assim, a Tabela 30 exibe a configuração da implementação do BRKGA-VCD executada nos experimentos. O recurso computacional utilizado é descrito na Seção 4.2. Em cada experimento, cada instância foi executada 10 vezes com sementes sequenciais no intervalo $[1, 10]$ para o gerador de números pseudoaleatórios, a fim de controlar o âmbito de execução. O resultado para cada instância reportado nesta seção é a média das execuções. As instâncias e os melhores resultados obtidos pelo BRKGA-VCD durante este experimento podem ser obtidos no seguinte sítio eletrônico: <https://gitlab.com/AndersonZM/bin-packing-instance-sol>.

Tabela 30 – Configuração dos componentes do BRKGA-VCD que foi comparado com os algoritmos da literatura.

Componente	**Configuração**
Critério de parada	200 gerações
Função objetivo	FOA
Heurística Construtiva	Empacotamento com DFTRC
Tamanho da população	$P_{max} = 30 \times n$
Fração TOP	$\epsilon = 0{,}1$
Fração BOT	$\omega = 0{,}15$
Taxa de elitismo	$\rho = 0{,}7$

Os resultados obtidos neste experimento foram comparados com os dados reportados nos trabalhos da literatura relacionados pela Tabela 31, que associa os algoritmos estado da arte para o 3BP e o 2BP. Estes trabalhos utilizam o mesmo conjunto de instâncias mencionado como base de teste para os experimentos computacionais. A tabela relaciona o respectivo pseudônimo do algoritmo com o trabalho de referência e a estratégia utilizada para resolver os problemas de empacotamento. Todos os dados reportados neste trabalho são os mesmos apresentados pelos autores das respectivas referências, com exceção do MVP. O MVP foi executado no mesmo ambiente computacional deste trabalho. Seu código implementado na linguagem de programação *C* e está disponível em: <http://www.diku.dk/~pisinger/codes.html>. A implementação do MVP foi compilada com a mesma configuração recomendada pelos autores, isto é, através do *GCC* (*GNU Compiler Collection*) com a *flag* de otimização $-O3$.

4.6.1 Resultados com Orientação Fixa

Esta seção apresenta os resultados para o 3BP e o 2BP com itens de orientação fixa. Antes de exibir os resultados completos, o BRKGA-VCD foi comparado diretamente

Tabela 31 – Métodos literatura utilizados na comparação com o BRKGA-VCD.

Algoritmo	Referência	Estratégia
MVP	(MARTELLO; PISINGER; VIGO, 2000)	Branch & Bound Truncado (3BP)
TS3	(LODI; MARTELLO; VIGO, 2002)	Busca Tabu (3BP/2BP)
GLS	(FAROE; PISINGER; ZACHARIASEN, 2003)	Busca Local Guiada (3BP/2BP)
GASP	(CRAINIC; PERBOLI; TADEI, 2012)	Heurística gulosa (3BP)
EHGH2	(HIFI; NEGRE; WU, 2014)	Heurística baseada em programação linear inteira (3BP)
GVND	(PARREÑO et al., 2010a)	GRASP/VND híbrido (3BP/2BP)
*BRKGA	(GONÇALVES; RESENDE, 2013)	*Biased random-key Genetic Algorithm* (3BP/2BP)
HPB	(BOSCHETTI; MINGOZZI, 2003b)	Heurística (2BP)
SCH	(MONACI; TOTH, 2006)	Heurística baseada em cobertura de conjunto (3BP)

*Único trabalho que reporta soluções com a possibilidade de rotação.

com o BRKGA. Ambos os algoritmos foram executados para as mesmas instâncias com o mesmo critério de parada, atingir 200 gerações, mas testados em diferentes ambientes computacionais. A Tabela 32 mostra o acumulado de caixas obtidas em todas as instâncias do 3BP e do 2BP sem rotação, através das gerações do BRKGA-VCD. A tabela também mostra a mesma informação para o BRKGA original com os valores reportados no trabalho (GONÇALVES; RESENDE, 2013).

Tabela 32 – Acumulado de caixas obtidas em todas as instâncias do 3BP e do 2BP sem rotação através das gerações.

Método	Número de gerações									
	1	10	25	50	75	100	125	150	175	200
BRKGA-VCD (3BP)	9951	9836	9810	9773	9772	9771	9770	9770	9769	9768
BRKGA (3BP)	10066	9959	9862	9806	9798	9791	9786	9781	9778	9777
BRKGA-VCD (2BP)	7286	7280	7251	7242	7232	7231	7231	7230	7230	7230
BRKGA (2BP)	7359	7299	7261	7248	7245	7241	7234	7234	7234	7234

Considerando que os números de avaliações do BRKGA-VCD e do BRKGA são o mesmo, o método proposto neste trabalho foi capaz de obter um acumulado final de caixas melhor com 50 iterações comparado com 200 iterações do BRKGA para o 3BP. Podemos observar também que o acúmulo de caixas utilizadas pela melhor solução na população inicial é significantemente menor que o reportado pelo método original. A quantidade de caixas economizadas através das gerações mostra que o VCD acelera consideravelmente o processo evolutivo do BRKGA. Além disso, as soluções encontradas pelo BRKGA-VCD são melhores que o BRKGA em qualquer geração para este conjunto de instâncias. Podemos observar que o resultado para as instâncias do 2BP conduzem às

mesmas conclusões, sendo que para este conjunto o BRKGA-VCD foi capaz de obter um acumulado final melhor com somente 75 iterações, comparado com o resultado reportado pelo trabalho original com 200 iterações. Finalmente, podemos concluir que o VCD tem um impacto positivo significativo na qualidade geral da população inicial e mutantes, sugerindo que a população gerada é muito boa comparando-se com o resultado original que utiliza codificações completamente aleatórias.

A Tabela 33 apresenta os resultados do BRKGA/VCD para as instâncias do 3BP sem rotação, porém detalhando cada média de cada subconjunto composto por 10 instâncias. As colunas 1 e 2 relacionam a classe Cl e a quantidade de itens n de cada subconjunto, notando-se que os elementos da tabela são as médias. A coluna 3 reporta a média $200g$ do tempo total de execução em CPU do BRKGA-VCD para atingir o critério de parada de 200 gerações e a coluna 4 exibe a solução média Z obtida. As demais colunas reportam os resultados dos trabalhos da literatura com exceção do algoritmo MVP. Como citado anteriormente, o MVP foi executado neste ambiente computacional com o critério de parada sendo 60 segundos de execução em CPU. A penúltima linha da tabela apresenta o acumulado de caixas obtidas nas instâncias das classes $1, 4$ até 8, e a última exibe o acumulado total de caixas em todas as instâncias.

A tabela evidencia que o BRKGA-VCD produz soluções melhores ou equivalentes comparado aos resultados reportados em todos os trabalhos comparados da literatura. O método não conseguiu atingir o provável ótimo global para somente um conjunto de instâncias com um acumulado que difere de 1 caixa do melhor resultado conhecido. Isto significa, com alta probabilidade, que somente uma instância deste subconjunto não está no provável ótimo global conhecido. Diversas soluções melhoram a qualidade da melhor reportada até este momento, sendo este resultado um novo candidato para o provável ótimo global do respectivo conjunto de instâncias. Particularmente, a última melhora na solução para o subconjunto de classe 4 com $n = 100$ foi reportada no trabalho (LODI; MARTELLO; VIGO, 2002). O BRKGA-VCD foi capaz de achar uma solução melhor com baixo tempo de execução em CPU no recurso computacional utilizado. O acumulado final de caixas mostra que o BRKGA-VCD é o melhor algoritmo dentre os comparados para este conjunto de instâncias. O teste de Wilcoxon pareado com 5% de nível de significância diz que as diferenças observadas na tabela são significativas, com exceção do BRKGA. O teste mostra que o resultado do BRKGA não tem diferença estatística significativa do BRKGA-VCD, porém, a Tabela 32 mostra que o método proposto neste trabalho é capaz de gerar soluções melhores que o reportado pelo BRKGA com somente 25% das iterações feitas.

A Tabela 34 reporta os resultados do BRKGA-VCD obtidos para o 2BP sem rotação. A configuração desta tabela é a mesma da anterior, onde cada elemento é a média obtida para o conjunto de 10 instâncias relacionadas nas colunas 1 e 2. Assim como na tabela anterior, as colunas 3 e 4 reportam os resultados para o BRKGA/VCD e as demais colunas são os dados reportados pelos respectivos trabalhos da literatura.

A tabela mostra, para o conjunto bidimensional, que o BRKGA-VCD provê soluções melhores ou equivalentes a todas as reportadas pelos trabalhos comparados da literatura. Todos os prováveis ótimos globais são encontrados para este conjunto em pouco tempo de execução de CPU no ambiente computacional utilizado. Além disso, vários resultados obtidos melhoram a qualidade reportada pelos trabalhos da literatura. O acumulado final de caixas mostra que o BRKGA-VCD obtém soluções melhores que todos os outros métodos comparados para todas as instâncias deste conjunto. O teste de Wilcoxon pareado mostra que as diferenças observadas na tabela implicam que o BRKGA-VCD é

significativamente melhor que todos métodos comparados, com exceção do BRKGA. Para este último algoritmo, o teste diz que os resultados da tabela não provê uma diferença estatisticamente significativa. Porém, a Tabela 32 mostra que o BRKGA-VCD é capaz de obter um acumulado final melhor que o BRKGA com 125 iterações comparado com 200 do método original.

Vale observar que o BRKGA-VCD e o BRKGA efetuam a mesma quantidade de avaliações por iteração. Isto se deve ao algoritmo VCD que decodifica o mesmo número de soluções que o método original, pois os parâmetros calibráveis do BRKGA-VCD receberam os mesmos valores utilizados em (GONÇALVES; RESENDE, 2013). Além disso, o tempo de execução para decodificar as soluções constitui 99,9% do tempo total de execução. Assim, o VCD é uma estratégia algorítmica que impõe um *overhead* insignificante comparado com a execução do método original. Quanto ao tempo de execução em CPU, este trabalho não apresenta nenhuma análise ou observação comparativa entre o algoritmo proposto e os métodos comparados da literatura, pois todos os algoritmos foram executados em ambientes computacionais distintos.

4.6.2 Resultados com Rotação

O objetivo desta seção é apresentar os resultados obtidos pelo BRKGA-VCD para o problema de empacotamento no caso tridimensional e no bidimensional com rotação. Nestes casos, há a possibilidade de empacotar cada item em 6 ou 2 diferentes orientações, caso se considere 3BP ou 2BP, respectivamente. Este trabalho compara os dados obtidos neste experimento com os resultados reportados em (GONÇALVES; RESENDE, 2013) , referentes ao BRKGA com rotação. As duas abordagens foram executadas com o mesmo critério de parada, mas em diferentes ambientes computacionais. Então, a comparação entre os dois métodos é baseada somente em termos de qualidade da solução obtida.

A Tabela 35 mostra os resultados do experimento com rotação, seguindo o mesmo padrão das tabelas exibidas na seção anterior. A tabela mostra que o BRKGA-VCD provê os melhores resultados: os dados mostram que as soluções do método proposto neste trabalho são equivalentes ou melhores do que o algoritmo comparado. Entretanto, há diversos resultados que são iguais nos dois métodos para vários conjuntos de instâncias, particularmente, todos os de classe 4 e 5, implicando que os resultados dessas instâncias são provavelmente ótimos locais alta qualidade. O acúmulo total mostra que o BRKGA-VCD economizou 64 caixas nas 320 instâncias testadas. O teste Wilcoxon pareado com 5% de nível de significância mostra que as diferenças observadas na tabela são significantes. Além disso, a tabela mostra que o tempo total de execução é menor para todos os conjuntos de instâncias comparado com os valores observados na Tabela 33, referentes ao experimento com orientação fixa. Isto acontece porque as soluções encontradas neste experimento utilizam uma quantidade menor de caixas. Assim, o número de EM que são avaliadas nesta versão do problema é menor que o caso que tem orientação fixa.

A Tabela 36 exibe os resultados para o conjunto bidimensional com a mesma estrutura das tabelas anteriores. A tabela mostra que os resultados dos dois métodos são equivalentes em todos os conjuntos de instâncias, com exceção do conjunto de classe 3 com $n = 100$. O BRKGA-VCD economizou somente 1 caixa nas 500 instâncias bidimensionais testadas com rotação. Assim, os dois métodos são equivalentes para este conjunto de instâncias. Porém, todos os experimentos apresentados anteriormente nesta seção concluem que o BRKGA-VCD obtém soluções melhores que o BRKGA nos outros conjuntos de instâncias em uma quantidade inferior de iterações. Então, há uma alta probabilidade dos valores reportados serem um ótimo local de alta qualidade para itens com rotação.

Tabela 33 – Resultados do BRKGA-VCD e dos algoritmos comparados para o 3BP sem rotação. O critério de parada do método proposto foi atingir 200 gerações.

Cl	n	$200g(s)$	Z	MVP	TS3	GLS	GASP	EHGH2	GVND	BRKGA
1	50	12	**13,4**	13,6	**13,4**	**13,4**	**13,4**	13,8	**13,4**	**13,4**
	100	65	**26,6**	27,4	**26,6**	26,7	26,9	27,6	**26,6**	**26,6**
	150	164	**36,3**	37,7	36,7	37,0	37,0	39,8	36,4	36,4
	200	322	50,7	52,0	51,2	51,2	51,6	**50,6**	50,9	50,8
2	50	12	**13,8**	13,9	**13,8**	–	–	–	**13,8**	**13,8**
	100	66	**25,5**	26,4	25,7	–	–	–	25,7	25,6
	150	165	**36,6**	38,1	37,2	–	–	–	36,9	**36,6**
	200	320	**49,3**	50,2	50,1	–	–	–	49,4	49,4
3	50	12	**13,3**	13,6	**13,3**	–	–	–	**13,3**	**13,3**
	100	64	**25,9**	27,1	26,0	–	–	–	26,0	**25,9**
	150	171	**37,5**	39,2	37,7	–	–	–	37,6	**37,5**
	200	331	**49,8**	51,0	50,5	–	–	–	50,0	**49,8**
4	50	11	**29,4**	**29,4**	**29,4**	**29,4**	**29,4**	**29,4**	**29,4**	**29,4**
	100	61	**58,9**	59,2	59,0	59,0	59,0	59,5	59,0	59,0
	150	159	**86,8**	87,5	**86,8**	**86,8**	**86,8**	90,4	**86,8**	**86,8**
	200	318	**118,8**	119,0	**118,8**	119,0	**118,8**	119,0	**118,8**	**118,8**
5	50	30	**8,3**	9,2	8,4	**8,3**	8,4	7,9	**8,3**	**8,3**
	100	121	**15,0**	17,1	**15,0**	15,1	15,1	14,6	**15,0**	**15,0**
	150	278	**19,9**	24,0	20,4	20,2	20,6	21,5	20,1	20,0
	200	531	**27,1**	28,2	27,6	27,2	27,7	29,6	**27,1**	**27,1**
6	50	9	**9,7**	9,8	9,9	9,8	9,9	11,8	9,8	**9,7**
	100	47	**18,9**	19,3	19,1	19,1	19,1	19,2	19,0	**18,9**
	150	127	**29,0**	30,3	29,4	29,4	29,5	29,8	29,2	**29,0**
	200	257	**37,2**	37,8	37,7	37,7	38,0	38,7	37,4	37,3
7	50	24	**7,4**	7,9	7,5	**7,4**	7,5	**7,4**	**7,4**	**7,4**
	100	99	**12,2**	15,5	12,5	12,3	12,7	13,5	12,5	**12,2**
	150	228	**15,2**	19,9	16,1	15,8	16,6	18,2	16,0	15,3
	200	428	**23,4**	24,2	23,9	23,5	24,2	24,1	23,5	**23,4**
8	50	27	**9,2**	9,5	9,3	**9,2**	9,3	9,4	**9,2**	**9,2**
	100	122	**18,8**	21,4	18,9	18,9	19,0	18,9	18,9	18,9
	150	232	**23,6**	27,5	24,1	23,9	24,8	26,0	24,1	**23,6**
	200	451	**29,3**	31,1	30,3	29,9	31,1	35,8	29,8	**29,3**
(1, 4 – 8)			7251	7585	7320	7302	7364	7565	7286	7258
(1 – 8)			9768	10180	9863				9813	9777

*Valores em negrito correspondem às melhores soluções de cada conjunto de instâncias

Tabela 34 – Resultados do BRKGA-VCD e dos algoritmos comparados para o 2BP sem rotação. O critério de parada do método proposto foi atingir 200 gerações.

Cl	n	$200g(s)$	Z	TS3	GLS	HBP	SCH	GVND	BRKGA
1	20	2,5	**7,1**	**7,1**	**7,1**	**7,1**	**7.1**	**7,1**	**7,1**
	40	10,0	**13,4**	13,5	**13,4**	**13,4**	**13,4**	**13,4**	**13,4**
	60	23,3	**20,0**	20,1	20,1	20,1	**20,0**	**20,0**	**20,0**
	80	40,2	**27,5**	28,2	**27,5**	**27,5**	**27,5**	**27,5**	**27,5**
	100	64,4	**31,7**	32,6	32,1	31,8	**31,7**	**31,7**	**31,7**
2	20	4,7	**1,0**	**1,0**	**1,0**	**1,0**	**1,0**	**1,0**	**1,0**
	40	15,9	**1,9**	2,0	**1,9**	**1,9**	**1,9**	**1,9**	**1,9**
	60	38,7	**2,5**	2,7	**2,5**	**2,5**	**2,5**	**2,5**	**2,5**
	80	69,0	**3,1**	3,3	**3,1**	**3,1**	**3,1**	**3,1**	**3,1**
	100	107,0	**3,9**	4,0	**3,9**	**3,9**	**3,9**	**3,9**	**3,9**
3	20	3,2	**5,1**	5,5	**5,1**	**5,1**	**5,1**	**5,1**	**5,1**
	40	12,8	**9,4**	9,7	**9,4**	9,5	**9,4**	**9,4**	**9,4**
	60	29,4	**13,9**	14,0	14,0	14,0	**13,9**	**13,9**	**13,9**
	80	58,6	**18,9**	19,8	19,1	19,1	**18,9**	**18,9**	**18,9**
	100	120,9	**22,3**	23,6	22,6	22,6	**22,3**	**22,3**	**22,3**
4	20	7,4	**1,0**	**1,0**	**1,0**	**1,0**	**1,0**	**1,0**	**1,0**
	40	28,2	**1,9**	**1,9**	**1,9**	**1,9**	**1,9**	**1,9**	**1,9**
	60	56,7	**2,4**	2,6	2,5	2,5	2,5	2,5	2,5
	80	100,0	**3,1**	3,3	3,3	3,3	3,2	**3,1**	**3,1**
	100	167,0	**3,7**	4,0	3,8	3,8	3,8	3,8	**3,7**
5	20	3,2	**6,5**	6,6	**6,5**	**6,5**	**6,5**	**6,5**	**6,5**
	40	13,1	**11,9**	**11,9**	**11,9**	**11,9**	**11,9**	**11,9**	**11,9**
	60	30,3	**18,0**	18,2	18,1	**18,0**	**18,0**	**18,0**	**18,0**
	80	54,9	**24,7**	25,1	24,9	24,8	**24,7**	**24,7**	**24,7**
	100	88,7	**28,1**	29,5	28,8	28,7	28,2	28,2	**28,1**
6	20	6,3	**1,0**	**1,0**	**1,0**	**1,0**	**1,0**	**1,0**	**1,0**
	40	28,1	**1,6**	1,9	1,8	1,8	1,7	1,7	**1,6**
	60	69,2	**2,1**	2,2	2,2	**2,1**	**2,1**	**2,1**	**2,1**
	80	117,1	**3,0**	**3,0**	**3,0**	**3,0**	**3,0**	**3,0**	**3,0**
	100	220,0	**3,2**	3,4	3,4	3,4	3,4	3,4	3,3
7	20	2,7	**5,5**	**5,5**	**5,5**	**5,5**	**5,5**	**5,5**	**5,5**
	40	11,9	**11,1**	11,4	11,3	**11,1**	**11,1**	**11,1**	**11,1**
	60	28,0	**15,8**	16,2	15,9	16,0	**15,8**	15,9	**15,8**
	80	53,4	**23,1**	23,2	23,2	23,2	23,2	23,2	23,2
	100	85,0	**27,1**	27,7	27,5	27,4	**27,1**	**27,1**	**27,1**
8	20	2,4	**5,8**	**5,8**	**5,8**	**5,8**	**5,8**	**5,8**	**5,8**
	40	12,0	**11,3**	11,4	11,4	**11,3**	**11,3**	**11,3**	**11,3**
	60	28,2	**16,1**	16,2	16,3	16,2	16,2	**16,1**	**16,1**
	80	53,3	**22,4**	22,6	22,5	22,6	**22,4**	**22,4**	**22,4**
	100	84,5	**27,7**	28,4	28,1	28,0	27,9	27,8	27,8
9	20	2,6	**14,3**	**14,3**	**14,3**	**14,3**	**14,3**	**14,3**	**14,3**
	40	11,8	**27,8**	**27,8**	**27,8**	**27,8**	**27,8**	**27,8**	**27,8**
	60	29,1	**43,7**	43,8	**43,7**	**43,7**	**43,7**	**43,7**	**43,7**
	80	51,8	**57,7**	**57,7**	**57,7**	**57,7**	**57,7**	**57,7**	**57,7**
	100	90,4	**69,5**	**69,5**	**69,5**	**69,5**	**69,5**	**69,5**	**69,5**
10	20	3,3	**4,2**	4,3	**4,2**	4,3	**4,2**	**4,2**	**4,2**
	40	13,4	**7,4**	7,5	**7,4**	**7,4**	**7,4**	**7,4**	**7,4**
	60	30,9	**10,0**	10,4	10,2	10,2	10,1	**10,0**	**10,0**
	80	60,2	**12,8**	13,0	13,0	13,0	**12,8**	12,9	**12,8**
	100	97,3	**15,8**	16,6	16,2	16,2	15,9	15,9	**15,8**
(1 – 10)			7230	7360	7284	7275	7243	7241	7234

*Valores em negrito correspondem às melhores soluções de cada conjunto de instâncias

Tabela 35 – Resultados do BRKGA-VCD e do algoritmo comparado para o 3BP com rotação. O critério de parada do método proposto foi atingir 200 gerações.

Cl	n	$200g(s)$	Z	BRKGA
1	50	6,8	**11,7**	11,8
	100	18,5	**23,0**	**23,0**
	150	59,8	**31,7**	**31,7**
	200	110,2	**43,1**	43,4
2	50	4,6	**11,7**	11,8
	100	19,1	**22,4**	22,5
	150	58,2	**31,5**	**31,5**
	200	109,0	**42,4**	42,5
3	50	6,7	**11,6**	11,6
	100	18,8	**22,6**	**22,6**
	150	59,5	**32,0**	32,4
	200	111,5	**42,3**	**42,3**
4	50	3,5	**28,9**	**28,9**
	100	19,5	**58,4**	**58,4**
	150	42,7	**86,4**	**86,4**
	200	79,9	**118,3**	**118,3**
5	50	11,4	**7,5**	**7,5**
	100	34,6	**13,7**	**13,7**
	150	135,4	**18,6**	**18,6**
	200	229,4	**25,3**	**25,3**
6	50	2,9	**8,9**	9,4
	100	12,7	**17,9**	18,9
	150	24,2	**27,5**	28,2
	200	47,0	**33,3**	**33,3**
7	50	8,5	**6,4**	**6,4**
	100	25,7	**10,8**	11,3
	150	74,7	**13,7**	14,6
	200	147,4	**20,3**	**20,3**
8	50	7,0	**8,3**	9,2
	100	26,0	**17,5**	18,2
	150	77,7	**22,0**	22,1
	200	159,0	**24,8**	**24,8**
(1, 4 – 8)			6780	6837
(1 – 8)			8945	9009

*Valores em negrito correspondem às melhores soluções de cada conjunto de instâncias.

Tabela 36 – Resultados do BRKGA-VCD e do algoritmo comparado para o 2BP com rotação. O critério de parada do método proposto foi atingir 200 gerações.

Cl	n	$200g(s)$	Z	BRKGA
1	20	0,2	6,6	6,6
	40	0,8	12,8	12,8
	60	2,1	19,5	19,5
	80	3,7	27,0	27,0
	100	5,9	31,3	31,3
2	20	0,7	1,0	1,0
	40	2,2	1,9	1,9
	60	4,8	2,5	2,5
	80	8,5	3,1	3,1
	100	12,8	3,9	3,9
3	20	0,3	4,7	4,7
	40	1,1	9,2	9,2
	60	2,6	13,4	13,4
	80	5,3	18,2	18,2
	100	8,8	**21,9**	22,0
4	20	0,9	1,0	1,0
	40	2,9	1,9	1,9
	60	7,2	2,3	2,3
	80	11,7	3,1	3,1
	100	18,1	3,7	3,7
5	20	0,3	5,9	5,9
	40	1,2	11,4	11,4
	60	3,0	17,2	17,2
	80	5,6	23,9	23,9
	100	8,5	27,7	27,7
6	20	0,9	1,0	1,0
	40	3,4	1,6	1,6
	60	8,1	2,1	2,1
	80	14,3	3,0	3,0
	100	27,3	3,2	3,2
7	20	0,3	5,2	5,2
	40	1,2	10,2	10,2
	60	2,6	14,6	14,6
	80	5,2	20,8	20,8
	100	8,0	25,0	25,0
8	20	0,3	5,3	5,3
	40	1,1	10,3	10,3
	60	2,5	14,7	14,7
	80	5,3	20,4	20,4
	100	9,1	25,2	25,2
9	20	0,3	14,3	14,3
	40	1,3	27,5	27,5
	60	3,2	43,5	43,5
	80	5,8	57,3	57,3
	100	10,1	69,3	69,3
10	20	0,4	4,1	4,1
	40	1,5	7,2	7,2
	60	3,4	9,9	9,9
	80	6,4	12,5	12,5
	100	10,1	15,4	15,4
(1 – 10)			6987	6988

*O valor em negrito corresponde ao único resultado melhorado.

CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho propôs três heurísticas para os problemas clássicos de empacotamento tridimensional e bidimensional. Os algoritmos são baseados nas meta-heurísticas GRASP, Busca Tabu e BRKGA. Cada algoritmo proposto foi continuamente desenvolvido ao longo do ano de 2017. Inicialmente, o trabalho (ZUDIO et al., 2017) introduziu o algoritmo GRASP-VND, e foi apresentado no SBPO 2017. Esta dissertação estendeu este algoritmo com novos componentes, por exemplo, o DFTRC e o FOA. Posteriormente, o trabalho (ZUDIO et al., 2018), apresentado no IC-VNS 2017, propôs o algoritmo BRKGA-VCD como uma variante do BRKGA. Este método também foi estendido neste documento, onde melhores resultados foram obtidos.

Os métodos propostos são baseados em uma heurística construtiva que gera soluções a partir de uma sequência de itens. O empacotamento dos itens é feito através de espaços maximais, os quais modelam o espaço vazio remanescente de cada caixa aberta. A escolha de um espaço maximal de uma determinada caixa provê a localização em que o item é empacotado. Esta dissertação apresenta três diferentes critérios de avaliação para espaços maximais. Os critérios são utilizados pela heurística construtiva para avaliar os espaços maximais. O algoritmo de empacotamento foi estendido para atender as variantes dos problemas em que há a possibilidade de rotação dos itens. Além disso, o trabalho também apresenta análises de complexidade de cada componente proposto, cujo objetivo é detalhar o comportamento prático dos algoritmos propostos e seu processo de implementação.

Diferentes estratégias e abordagens foram apresentadas para cada fase e componente das heurísticas GRASP, Busca Tabu e BRKGA, de maneira que é possível combiná-las para formar diferentes implementações. Tais implementações foram testadas em diversos experimentos computacionais preliminares sem rotação, a fim de selecionar as melhores configurações de cada heurística. Foi possível determinar a melhor disposição de cada abordagem para um conjunto de 820 instâncias, as quais compõem a base de teste padrão para vários trabalhos encontrados na literatura. Posteriormente, os parâmetros calibráveis das implementações GRASP e Busca Tabu com as estratégias selecionadas foram ajustados empiricamente através de uma ferramenta computacional. Este trabalho exibiu uma comparação entre as três heurísticas, executando a implementação selecionada de cada uma no mesmo ambiente computacional e com o mesmo critério de parada para as 820 instâncias padrões. Os experimentos mostraram que o BRKGA proposto aliado com a nova abordagem VCD obtém sistematicamente os melhores resultados.

Para estabelecer uma comparação com abordagens de outros autores, o BRKGA-VCD foi executado novamente em dois experimentos computacionais com as mesmas instâncias, onde o primeiro deles considerou itens com orientação fixa, e o segundo permitiu a possibilidade de rotacionar os itens. Os resultados obtidos foram comparados com outras abordagens no estado da arte encontradas na literatura.

O primeiro experimento mostrou que o BRKGA-VCD provê as melhores soluções para o problema de empacotamento tridimensional e bidimensional com orientação fixa, aprimorando o melhor resultado conhecido para diversas instâncias.

No segundo experimento, o BRKGA-VCD melhorou significativamente o melhor resultado conhecido para as instâncias tridimensionais com rotação: 64 caixas foram economizadas nas 320 instâncias testadas, sendo que o método proposto neste trabalho obteve melhores resultados para a maioria dos conjuntos testados comparados às demais abordagens. Os resultados obtidos para as instâncias bidimensionais com rotação foram equivalentes à melhor abordagem pré-existente, o que sugere serem estes resultados prováveis ótimos globais.

Evitamos utilizar tempo de execução em CPU nas comparações com as outras abordagens da literatura, pois cada algoritmo comprado foi executado em um ambiente computacional diferente. Finalmente, os dois experimentos também mostram que o BRKGA-VCD é capaz de obter resultados de alta qualidade para instâncias de larga escala em pouco tempo de execução em CPU, quando executados em um notebook padrão munido com um processador *single core* de 2,5GHz.

Como trabalhos futuros, pretende-se estender a base de testes padrão com instâncias que utilizam uma quantidade maior de itens. Outra meta é melhorar a implementação dos algoritmos através de estruturas ou estratégias que efetuam a interseção de paralelepípedos ou retângulos com um menor tempo de execução em CPU, acelerando a construção parcial ou completa das soluções. Objetiva-se estudar outras variantes do 3BP e do 2BP, por exemplo, considerar itens com peso ou caixas não homogêneas. Outro objetivo futuro é estudar os limites inferiores para determinar se os resultados encontrados são ótimos globais. Além disso, pretende-se estender as abordagens propostas para a execução em ambientes heterogêneos CPU-GPU.

REFERÊNCIAS

ARMENTANO, V. A.; ARAÚJO, O. C. B. Heurística de múltiplos inícios para o problema de empacotamento tridimensional em faixa. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, XXIX, 2007, Fortaleza. *Anais do SBPO 2007*. Rio de Janeiro: SOBRAPO, 2007. p. 1530–1541.

BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on computing*, INFORMS, v. 6, n. 2, p. 154–160, jan. 1994.

BERKEY, J. O.; WANG, P. Y. Two-dimensional finite bin-packing algorithms. *Journal of the operational research society*, Springer, Great Britain, v. 38, n. 5, p. 423–429, maio 1987.

BOEF, E. D. et al. Erratum to "the three-dimensional bin packing problem": Robot-packable and orthogonal variants of packing problems. *Operations Research*, INFORMS, Maryland, v. 53, n. 4, p. 735–736, jul./ago. 2005.

BOSCHETTI, M. A. New lower bounds for the three-dimensional finite bin packing problem. *Discrete Applied Mathematics*, Elsevier, v. 140, n. 1–3, p. 241–258, maio 2004.

BOSCHETTI, M. A.; MINGOZZI, A. The two-dimensional finite bin packing problem. part i: New lower bounds for the oriented case. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, Springer, v. 1, n. 1, p. 27–42, 2003.

BOSCHETTI, M. A.; MINGOZZI, A. The two-dimensional finite bin packing problem. part ii: New lower and upper bounds. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, Springer, v. 1, n. 2, p. 135–147, 2003.

BOYAR, J. et al. Online bin packing with advice. *Algorithmica*, Springer, New York, v. 74, n. 1, p. 507–527, jan. 2016.

BRITO, J. A. M.; MACULAN, N.; BRITO, L. R. Algoritmo grasp aplicado ao problema de planejamento do tratamento por radiocirurgia. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, XLVII, 2015, Ipojuca. *Anais do SBPO 2015*. Rio de Janeiro: SOBRAPO, 2015. p. 1723–1734.

CAPOROSSI, G.; HANSEN, P.; MLADENOVIĆ, N. Variable neighborhood search. In: SIARRY, P. (Ed.). *Metaheuristics*. 1. ed. Cham, Switzerland: Springer, 2016. p. 77–98.

CHRISTENSEN, H. et al. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, Elsevier, v. 24, n. 1, p. 63–79, maio 2017.

COELHO, I. M. et al. Optframe: a computational framework for combinatorial optimization problems. In: WORKSHOP ON APPLIED COMBINATORIAL OPTIMIZATION, VII, 2011, Porto. *Proceedings of VII ALIO/EURO*. Porto: ALIO-EURO, 2011. p. 51–54.

COFFMAN JR., E. G.; CSIRIK; LEUNG, J. Y.-T. Variable-sized bin packing and bin covering. In: GONZALEZ, T. F. (Ed.). *Handbook of Approximation Algorithms and Metaheuristics*. 1. ed. New York: Chapman & Hall/CRC, 2006. cap. 34, p. 1–10.

COFFMAN JR., E. G.; CSIRIK; LEUNG, J. Y.-T. Variants of classical one-dimensional bin packing. In: GONZALEZ, T. F. (Ed.). *Handbook of Approximation Algorithms and Metaheuristics*. 1. ed. New York: Chapman & Hall/CRC, 2006. cap. 33, p. 77–98.

COFFMAN JR., E. G.; CSIRIK, J. A classification scheme for bin packing theory. *Acta Cybernetica*, ACM, v. 18, n. 1, p. 47–60, 2007.

COFFMAN JR., E. G. et al. Bin packing approximation algorithms: survey and classification. In: PARDALOS, M. P.; DU, D.-Z.; GRAHAM, R. (Ed.). *Handbook of combinatorial optimization*. 1. ed. New York: Springer, 2013. p. 455–531.

COFFMAN JR., E. G.; GAREY M, R.; S., J. D. Approximation algorithms for bin-packing - an updated survey. In: AUSILLO, G.; LUCERTINI, M.; SERAFINI, P. (Ed.). *Algorithm Design for Computer System Design*. New York: Springer, 1984. v. 284, p. 49–106.

COFFMAN JR., E. G.; GAREY M, R.; S., J. D. Approximation algorithms for bin packing: A survey. In: HOCHBAUM, D. (Ed.). *Approximation Algorithms for NP-Hard Problems*. 1. ed. Boston: PWS, 1996. cap. 2, p. 1–53.

CÔTÉ, J.-F.; GENDREAU, M.; POTVIN, J.-Y. An exact algorithm for the two-dimensional orthogonal packing problem with unloading constraints. *Operations Research*, INFORMS, v. 62, n. 5, p. 1126–1141, out. 2014.

CRAINIC, T. G.; PERBOLI, G.; TADEI, R. Ts2 pack: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research*, Elsevier, v. 195, n. 3, p. 744–760, jun. 2009.

CRAINIC, T. G.; PERBOLI, G.; TADEI, R. *A greedy adaptive search procedure for multi-dimensional multi-container packing problems*. Quebec: CIRRELT, 2012. 1–19 p. Disponível em: <http://porto.polito.it/2495925/1/CIRRELT_2012_10.pdf>. Acesso em: 14 de novembro de 2017.

DELL'AMICO, M.; MARTELLO, S.; VIGO, D. A lower bound for the non-oriented two-dimensional bin packing problem. *Discrete Applied Mathematicsh*, Elsevier, v. 118, n. 1, p. 13–24, abril 2002.

DYCKHOFF, H. A typology of cutting and packing problems. *European Journal of Operational Research*, Elsevier, v. 44, n. 2, p. 145–159, jan. 1990.

EPSTEIN, L.; STEE, R. v. Multidimensional packing problems. In: GONZALEZ, T. (Ed.). *Handbook of Approximation Algorithms and Metaheuristics*. 1. ed. New York: Chapman & Hall/CRC, 2007. cap. 35, p. 1–15.

FALKENAUER, E. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, Springer, v. 2, n. 1, p. 5–30, jun. 1996.

FAROE, O.; PISINGER, D.; ZACHARIASEN, M. Guided local search for the three-dimensional bin-packing problem. *Journal on computing*, INFORMS, v. 15, n. 3, p. 267–283, ago. 2003.

FEKETE, S. P.; SCHEPERS, J. New classes of fast lower bounds for bin packing problems. *Mathematical programming*, Springer, v. 91, n. 1, p. 11–31, out. 2001.

FEKETE, S. P.; SCHEPERS, J. A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research*, INFORMS, v. 29, n. 9, p. 353–368, maio 2004.

FEKETE, S. P.; SCHEPERS, J. A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, Springer, v. 60, n. 2, p. 311–329, out. 2004.

FEKETE, S. P.; SCHEPERS, J.; VEEN, J. C. Van der. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, INFORMS, v. 55, n. 3, p. 569–587, jun. 2007.

FEKETE, S. P.; VEEN, J. C. Van der. Packlib 2: An integrated library of multi-dimensional packing problems. *European Journal of Operational Research*, Elsevier, v. 183, n. 3, p. 1131–1135, dez. 2007.

GAREY, M. R.; GRAHAM, R. L.; ULLMAN, J. D. Worst-case analysis of memory allocation algorithms. In: ACM SYMPOSIUM ON THEORY OF COMPUTING, 4, 1972, Colorado. *Proc. of the fourth ann. ACM sym. on The. of comp.* New York: ACM, 1972. p. 143–150.

GAREY, M. R.; JOHNSON, D. S. "strong"np-completeness results: Motivation, examples, and implications. *Journal of the ACM*, ACM, v. 25, n. 3, p. 499–508, jul. 1978.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1. ed. New York: W. H. Freeman & Co., 1979. ISBN 0716710447.

GAREY, M. R.; JOHNSON, D. S. Approximation algorithms for bin packing problems: A survey. In: AUSIELLO, G.; LUCERTINI, M. (Ed.). *Analysis and design of algorithms in combinatorial optimization*. 1. ed. New York: Springer, 1981. p. 147–172.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, Elsevier, v. 13, n. 5, p. 533–549, maio 1986.

GLOVER, F.; LAGUNA, M. Tabu search*. In: PARDALOS, M. P.; DU, D.-Z.; GRAHAM, R. (Ed.). *Handbook of Combinatorial Optimization*. 1. ed. New York: Springer, 2013. p. 3261–3362.

GLOVER, F. W.; KOCHENBERGER, G. A. *Handbook of metaheuristics*. New York: Springer, 2006. v. 57.

GONÇALVES, J. F.; RESENDE, M. G. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, Springer, v. 17, n. 5, p. 487–525, out. 2011.

GONÇALVES, J. F.; RESENDE, M. G. A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, Elsevier, v. 145, n. 2, p. 500–510, 2013.

HIFI, M. et al. A linear programming approach for the three-dimensional bin-packing problem. *Electronic Notes in Discrete Mathematics*, Elsevier, v. 36, n. 5, p. 993–1000, ago. 2010.

HIFI, M.; NEGRE, S.; WU, L. Hybrid greedy heuristics based on linear programming for the three-dimensional single bin-size bin packing problem. *International Transactions in Operational Research*, Wiley Online Library, v. 21, n. 1, p. 59–79, dez. 2014.

HOCHBA, D. S. Approximation algorithms for np-hard problems. *International Transactions in Operational Research*, ACM, New York, v. 28, n. 2, p. 40–52, jun. 1997.

HOLLAND, J. H. *Adaptation in natural and artificial systems: An introductory analysis with application to biology, control, and artificial intelligence*. 1. ed. Cambridge: MIT PRESS, 1975.

JOHNSON, D. S. Fast allocation algorithms. In: ANNUAL SYMPOSIUM ON SWITCHING AND AUTOMATA THEORY, 13, 1972, Washington. *Record of 13th Annual Sym. on Swit. and Aut. Theory*. Piscataway: IEEE, 1972. p. 144–154.

JOHNSON, D. S. *Near-optimal bin packing algorithms*. 401 p. Tese (Doutorado) — Massachusetts Institute of Technology, Dept. of Mathematics, Cambridge, 1973.

JOHNSON, D. S. et al. Worst-case performance bounds for simple one-dimensional packing algorithms. *Journal on Computing*, SIAM, Philadelphia, v. 3, n. 4, p. 299–325, Jul. 1974.

KANG, K.; MOON, I.; WANG, H. A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Applied Mathematics and Computation*, Elsevier, Philadelphia, v. 219, n. 3, p. 1287–1299, Out. 2012.

KARP, R. M. Reducibility among combinatorial problems. In: MILLER, R. E.; THATCHER, J. W.; BOHLINGER, J. D. (Ed.). *Complexity of computer computations*. 1. ed. New York: Springer, 1972. p. 85–103.

LI, X.; ZHAO, Z.; ZHANG, K. A genetic algorithm for the three-dimensional bin packing problem with heterogeneous bins. In: INDUSTRIAL AND SYSTEMS ENGINEERING RESEARCH CONFERENCE, 2014, Montréal. *Proc. of the 2014 Indus. and Sys. Eng. Research Conference*. Peachtree Corners: IISE, 2014. p. 2039–2048.

LIAO, C.-S.; HSU, C.-H. New lower bounds for the three-dimensional orthogonal bin packing problem. *European Journal of Operational Research*, Elsevier, v. 225, n. 2, p. 244–252, março 2013.

LODI, A.; MARTELLO, S.; MONACI, M. Two-dimensional packing problems: A survey. *European journal of operational research*, Elsevier, v. 141, n. 2, p. 241–252, set. 2002.

LODI, A.; MARTELLO, S.; VIGO, D. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, Elsevier, v. 112, n. 1, p. 158–166, jan. 1999.

LODI, A.; MARTELLO, S.; VIGO, D. Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, Elsevier, v. 141, n. 2, p. 410–420, set. 2002.

LODI, A.; MARTELLO, S.; VIGO, D. Tspack: a unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research*, Springer, New York, v. 131, n. 1-4, p. 203–213, out. 2004.

LÓPEZ-IBÁÑEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, Elsevier, v. 3, p. 43–58, set. 2016.

MACK, D.; BORTFELDT, A. A heuristic for solving large bin packing problems in two and three dimensions. *Central European Journal of Operations Research*, Springer, v. 20, n. 2, p. 337–354, jun. 2012.

MARTELLO, S.; PISINGER, D.; VIGO, D. The three-dimensional bin packing problem. *Operations Research*, INFORMS, v. 48, n. 2, p. 256–267, abril 2000.

MARTELLO, S. et al. Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. *ACM Transactions on Mathematical Software*, ACM, New York, v. 33, n. 1, p. 7, março 2007.

MARTELLO, S.; TOTH, P. Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics*, Elsevier, v. 28, n. 1, p. 59–70, jul. 1990.

MARTELLO, S.; VIGO, D. Exact solution of the two-dimensional finite bin packing problem. *Management science*, INFORMS, v. 44, n. 3, p. 388–399, março 1998.

MONACI, M.; TOTH, P. A set-covering-based heuristic approach for bin-packing problems. *Journal on Computing*, INFORMS, v. 18, n. 1, p. 71–85, fev. 2006.

MOURA, A.; BORTFELDT, A. A two-stage packing problem procedure. *International Transactions in Operational Research*, Wiley Online Library, v. 24, n. 1-2, p. 43–58, jan. 2017.

PARREÑO, F. et al. A hybrid grasp/vnd algorithm for two-and three-dimensional bin packing. *Annals of Operations Research*, Springer, New York, v. 179, n. 1, p. 203–220, set. 2010.

PARREÑO, F. et al. Neighborhood structures for the container loading problem: a vns implementation. *Journal of Heuristics*, Springer, New York, v. 16, n. 1, p. 1–22, maio 2010.

PARREÑO, F. et al. A maximal-space algorithm for the container loading problem. *INFORMS Journal on Computing*, INFORMS, v. 20, n. 3, p. 412–422, 2008.

RESENDE, M. G.; RIBEIRO, C. C. *Optimization by GRASP*. 1. ed. New York: Springer, 2016.

RIOS, E. et al. Exploring parallel multi-gpu local search strategies in a metaheuristic framework. *Journal of Parallel and Distributed Computing*, Elsevier, v. 111, n. 1, p. 39–55, jan. 2018.

SARAIVA, R. D.; NEPOMUCENO, N.; PINHEIRO, P. R. A layer-building algorithm for the three-dimensional multiple bin packing problem: a case study in an automotive company. *IFAC-PapersOnLine*, Elsevier, v. 48, n. 3, p. 490–495, jan. 2015.

SILVA, J. C.; SOMA, N.; MACULAN, N. A greedy search for the three-dimensional bin packing problem: the packing static stability case. *International Transactions in Operational Research*, Wiley Online Library, v. 10, n. 2, p. 141–153, abril 2003.

SIMCHI-LEVI, D. et al. New worst-case results for the bin-packing problem. *Naval Research Logistics*, Wiley & Office of Naval Research, New York, v. 41, n. 4, p. 579–586, jun. 1994.

SZWARCFITER, J. L. *Teoria Computacional dos Grafos: Os Algoritmos*. 1. ed. Amsterdam: Elsevier, 2018.

SZWARCFITER, J. L.; MARKENZON, L. *Estruturas de Dados e seus Algoritmos*. 3. ed. Rio de Janeiro, RJ: LTC Editora, 2010.

WÄSCHER, G.; HAUSSNER, H.; SCHUMANN, H. An improved typology of cutting and packing problems. *European journal of operational research*, Elsevier, v. 183, n. 3, p. 1109–1130, dez. 2007.

ZUDIO, A. et al. Algoritmo grasp/vnd para o problema clássico de empacotamento tridimensional e bidimensional. In: SIMPóSIO BRASILEIRO DE PESQUISA OPERACIONAL, XLIX, 2017, Blumenau. *Anais do SBPO 2017*. Rio de Janeiro: SOBRAPO, 2017. p. 1831–1842.

ZUDIO, A. et al. Brkga/vnd hybrid algorithm for the classic three-dimensional bin packing problem. *Electronic Notes in Discrete Mathematics*, Elsevier, New York, v. 66, p. 175–182, abril 2018.