



Universidade do Estado do Rio de Janeiro  
Centro de Tecnologia e Ciências  
Instituto de Matemática e Estatística  
Programa de Pós-Graduação em Ciências Computacionais

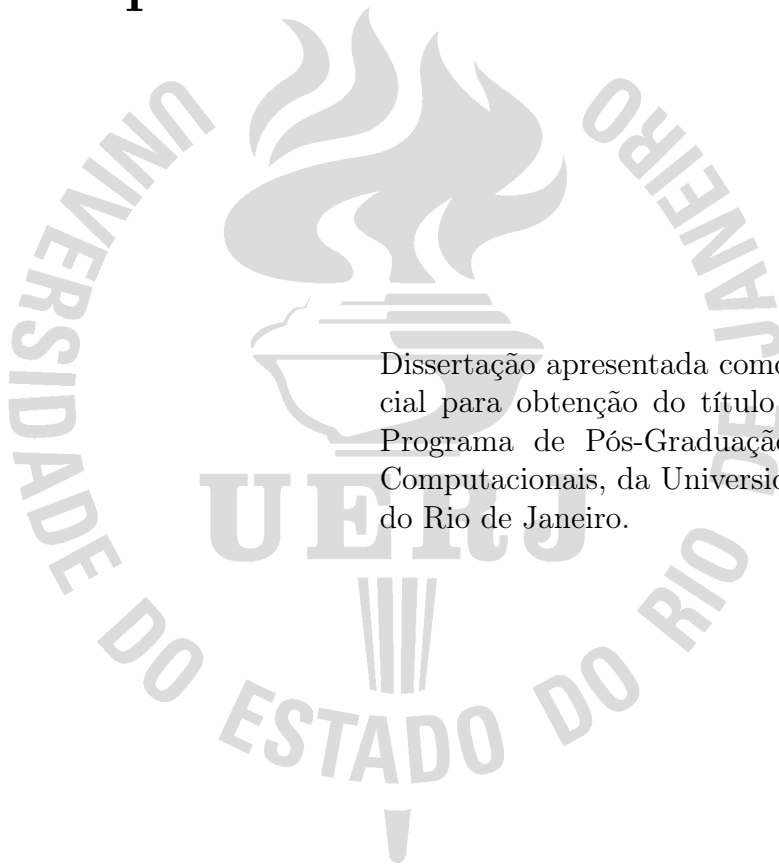
Anderson Zudio de Moraes

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

Rio de Janeiro  
2017

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  
Orientador: Prof. Dr. Igor Machado Coelho

Rio de Janeiro  
2017

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 13 de Novembro de 2017

Banca Examinadora:

---

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

---

Prof. Dr. Igor Machado Coelho (Orientador)  
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

Rio de Janeiro  
2017

# Resumo

ZUDIO, Anderson. *Meta-Heurísticas para o Problema de Empacotamento 2D e 3D*.

2017. 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. Este trabalho propõe como tema de dissertação o estudo da 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. Portanto, várias heurísticas combinadas com meta-heurísticas são propostas para resolver instâncias grandes 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 preliminares demonstram que os métodos propostos neste trabalho obtêm resultados de qualidade superior ou equivalente aos algoritmos estado da arte encontrados na literatura.

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

# Abstract

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 work studies the classic versions of two and three-dimensional case of bin packing with fixed and non-fixed orientated boxes as a dissertation theme. The bin packing problem has many industrial applications and relates to other complex problems as cutting, repartitioning and scheduling. Therefore, this work proposes heuristics combined with metaheuristics to devise good quality solutions for large scale instances. Extensive computational tests with 820 standard instances demonstrate that the methods proposed improves quality solution of state-of-art algorithms found in the literature.

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

# Sumário

<b>1</b>	<b>Introdução</b>	<b>10</b>
1.1	Motivação . . . . .	12
1.2	Estrutura deste documento . . . . .	13
<b>2</b>	<b>Conceitos Teóricos</b>	<b>15</b>
2.1	Complexidade Computacional . . . . .	15
2.1.1	Notação O-grande . . . . .	15
2.1.2	Notação Theta . . . . .	16
2.1.3	Problema de Decisão . . . . .	16
2.1.4	Problema de Otimização . . . . .	17
2.1.5	Classes de Problemas . . . . .	18
2.2	Heurística . . . . .	18
2.2.1	Algoritmos Gulosos e Gulosos Adaptativos . . . . .	19
2.2.2	Busca Local . . . . .	20
2.2.3	Variable Neighborhood Descent (VND) . . . . .	23
2.2.4	Representação da Solução . . . . .	25
2.3	Meta-heurística . . . . .	26
2.3.1	Greedy Randomized Adaptative Search Procedure . . . . .	26
2.3.2	Busca Tabu . . . . .	28
2.3.3	Algoritmo Evolucionário . . . . .	31
2.3.4	Biased Random-Keys Genetic Algorithm . . . . .	34
<b>3</b>	<b>Literatura</b>	<b>38</b>
3.1	Modelo de Programação Linear Inteira Mista . . . . .	38
3.2	Revisão da Literatura . . . . .	41
<b>4</b>	<b>Heurísticas para o 3BP e 2BP</b>	<b>44</b>
4.1	Representação da Solução . . . . .	44
4.2	Função Objetivo . . . . .	45
4.3	Espaço Maximal . . . . .	45
4.4	Algoritmo de Empacotamento . . . . .	47
4.4.1	Complexidade . . . . .	48
4.5	Heurística Construtiva . . . . .	50
4.5.1	Best Fit e Best EM . . . . .	52
4.5.2	DFTRC . . . . .	54
4.5.3	Complexidade . . . . .	56
4.6	Busca Local . . . . .	58
4.6.1	Complexidade . . . . .	61
4.7	Variable Neighborhood Descent . . . . .	64

4.7.1	Complexidade . . . . .	66
4.8	Greedy Randomized Adaptative Search Procedure . . . . .	68
4.8.1	Fase de construção . . . . .	68
4.8.2	Fase de Melhora . . . . .	69
4.8.3	Complexidade . . . . .	70
4.9	Busca Tabu . . . . .	71
4.9.1	Complexidade . . . . .	75
4.10	Biased Random-Keys Genetic Algorithm . . . . .	79
4.10.1	Variable Cross Descent . . . . .	83
4.10.2	Complexidade . . . . .	88
<b>5</b>	<b>Experimentos Computacionais Preliminares</b>	<b>94</b>
5.1	Instâncias . . . . .	94
5.1.1	Tridimensional . . . . .	94
5.1.2	Bidimensional . . . . .	95
5.2	Implementação e Ambiente Computacional . . . . .	96
5.3	Algoritmos Comparados . . . . .	96
5.4	Resultados Computacionais Preliminares . . . . .	97
5.4.1	Greedy Randomized Adaptative Search Procedure . . . . .	98
5.4.2	Busca Tabu . . . . .	102
5.4.3	Biased Random-Key Genetic Algorithm . . . . .	106
<b>6</b>	<b>Conclusão</b>	<b>111</b>
	<b>Referências Bibliográficas</b>	<b>113</b>

# Lista de figuras

1.1	Exemplos de empacotamento tridimensional e bidimensional. [Fonte: Autoria Própria]	11
2.1	Representação gráfica de um subespaço de soluções com duas vizinhanças distintas de uma mesma solução inicial.	24
2.2	Representação gráfica do vetor de candidatos e do RCL delimitado pelo $\alpha$ .	28
2.3	Exemplo gráfico da execução de uma Busca Tabu com duas iterações arbitrárias representadas em um subespaço de soluções.	30
2.4	Exemplo gráfico do operador de reprodução <i>one-point</i> .	34
2.5	Exemplo gráfico da estratégia da roleta.	34
2.6	Construção de uma geração do RKGA ou BRKGA.	36
2.7	Exemplo de reprodução no BRKGA. A probabilidade de escolha do gene elite é dado pelo parâmetro $\rho$ .	36
3.1	Exemplo de uma instância do 3BP sem rotação solucionada através do <i>First Fit</i> apresentado no Algoritmo 3.1. O empacotamento utiliza o ponto inferior esquerdo frontal como referência. A ordem de empacotamento dos itens é dada pelo tipo: a primeira iteração aloca um de tipo 1, na segunda um de tipo 2 e assim por diante.	40
4.1	Exemplos de EM modelando o espaço vazio em duas caixas.	47
4.2	Exemplo de corte em um EM tridimensional.	49
4.3	Exemplo de corte em um EM bidimensional.	49
4.4	Exemplos que ilustram as possíveis orientações de um item.	51
4.5	Exemplo bidimensional do critério de escolha Best Fit e Best EM.	54
4.6	Exemplo bidimensional da seleção do CI de dois novos EM com o Best Fit ou Best EM.	54
4.7	Exemplo bidimensional com rotação do critério de escolha DFTRC.	55
4.8	Exemplo do processo de ordenação efetuado pelo decodificador para $n = 6$ .	82
4.9	Exemplo do processo de codificação de uma permutação para $n = 6$ .	86



# Lista de tabelas

4.1	Complexidade dos procedimentos que compõem o Algoritmo 4.1. . . . .	50
4.2	Sumário da complexidade de tempo do Algoritmo 4.2. . . . .	58
4.3	Sumário da complexidade de tempo do Algoritmo 4.3. . . . .	63
4.4	Relação entre a vizinhança $V_k$ e seu método no VND proposto. . . . .	66
4.5	Sumário da complexidade de tempo de cada passo do Algoritmo 4.4 em conjunto com a complexidade final das vizinhanças $V_1, V_2, V_3$ e $V_4$ . . . . .	68
4.6	Sumário da complexidade dos componentes de cada GRASP proposto em conjunto com a complexidade final de uma iteração. . . . .	71
4.7	Sumário da complexidade dos componentes de cada busca tabu proposta. . . . .	79
4.8	Sumário da complexidade dos componentes do BRKGA e do BRKGA/VCD. . . . .	93
5.1	Tipos de itens utilizados nas classes 1 até 5 das instâncias 3D . . . . .	95
5.2	Tipos de itens utilizados nas classes 7 até 10 das instâncias 2D . . . . .	96
5.3	Métodos literatura utilizados na comparação preliminar. . . . .	97
5.4	Configuração utilizada no GRASP/VND. . . . .	98
5.5	Resultado preliminar do GRASP/VND para o 3BP sem rotação. . . . .	100
5.6	Resultado preliminar do GRASP/VND para o 2BP sem rotação. . . . .	101
5.7	Configuração dos componentes do BT/VN. . . . .	102
5.8	Resultado preliminar do BT/VN para o 3BP sem rotação. . . . .	103
5.9	Resultado preliminar do BT/VN para o 2BP sem rotação. . . . .	105
5.10	Configuração dos componentes do BRKGA/VCD. . . . .	106
5.11	Acumulado de caixas obtidas em todas as instâncias do 3BP e do 2BP sem rotação através das gerações. . . . .	106
5.12	Resultado preliminar do BRKGA/VCD para o 3BP sem rotação. . . . .	108
5.13	Resultado preliminar do BRKGA/VCD para o 2BP sem rotação. . . . .	109

# Lista de algoritmos

2.1	Construtivo Guloso . . . . .	19
2.2	Construtivo Guloso Adaptativo . . . . .	20
2.3	First Improvement . . . . .	22
2.4	Best Improvement . . . . .	22
2.5	Busca Local . . . . .	22
2.6	Variable Neighborhood Descent . . . . .	25
2.7	Semi-Guloso ou Guloso Randomizado . . . . .	27
2.8	Greedy Randomized Adaptive Search Procedure . . . . .	28
2.9	Busca Tabu . . . . .	30
2.10	Algoritmo Genético . . . . .	35
2.11	Biased Random-Keys Genetic Algorithm . . . . .	37
3.1	First Fit . . . . .	40
4.1	Algoritmo de Empacotamento . . . . .	48
4.2	Heurística construtiva de empacotamento . . . . .	52
4.3	Busca Local para o 3BP/2BP com <i>Best Improvement</i> . . . . .	60
4.4	Vinhança para $k \in \{2, 3, 4\}$ . . . . .	66
4.5	Fase de construção GRASP para o 3BP e o 2BP. . . . .	69
4.6	Busca Tabu BT para o 3BP e o 2BP . . . . .	74
4.7	Busca Tabu BT/VN para o 3BP e o 2BP . . . . .	76
4.8	Heurística construtiva de empacotamento para o BRKGA . . . . .	84
4.9	Decodificador utilizado no BRKGA . . . . .	84
4.10	Variable Cross Descent . . . . .	89

# Capítulo 1

## 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  $d_i$ ,  $i \in \{1, 2, \dots, n\}$ , 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 1.1 mostra exemplos de empacotamentos do 3BP e 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

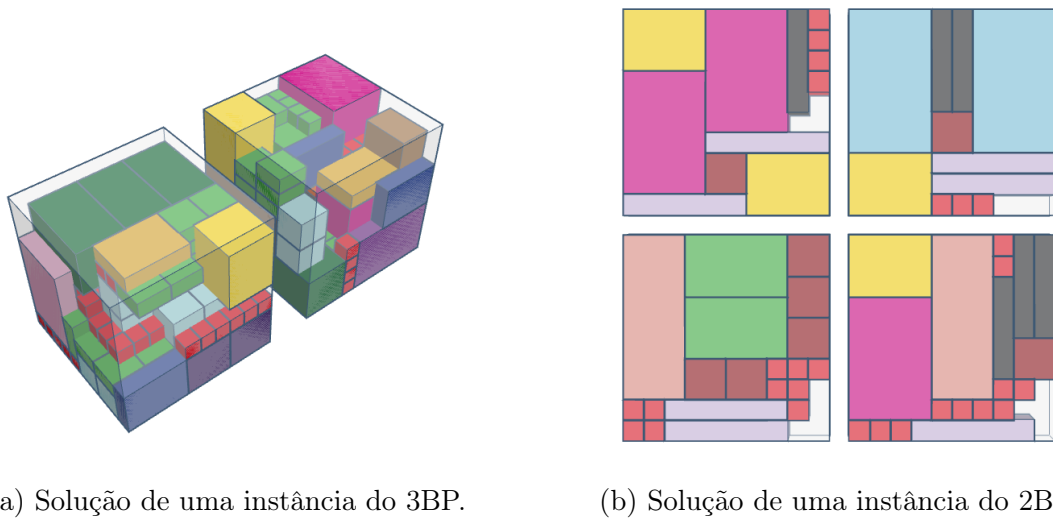


Figura 1.1: Exemplos de empacotamento tridimensional e bidimensional.

[Fonte: Autoria Própria]

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, este trabalho preliminar apresenta uma proposta de dissertação, descrevendo heurísticas gulosas que realizam o empacotamento dos itens combinadas com meta-heurísticas para resolver o 3BP e o 2BP. 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 a serem testadas e comparadas entre si. 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.

Como todas as possíveis versões de cada algoritmo proposto não foram completamente testadas, a melhor versão conhecida atualmente de cada algoritmo foi implementada e submetida a um extensivo teste computacional preliminar composto por 820 instâncias com itens de orientação fixa. Tais instâncias são a base de teste para diversos trabalhos da literatura. Os resultados computacionais preliminares mostram que os métodos propostos obtêm soluções de qualidade superior ou equivalente a outros algoritmos no estado da arte encontrados na literatura. Além disso, os dados demonstram que um dos métodos propostos, chamado BRKGA/VCD, fornece soluções que são provavelmente ótimas, melhorando o melhor resultado reportado na literatura para este conjunto de instâncias.

Como trabalho final de dissertação, objetiva-se determinar a melhor versão de cada algoritmo através de testes empíricos no mesmo ambiente computacional utilizado neste trabalho, pois diversos componentes descritos neste documento não foram extensivamente

testados com cada método. Desta maneira, as versões com as melhores configurações serão testadas novamente com a base de teste padrão da literatura. Os novos resultados computacionais serão utilizados para estender a análise comparativa apresentada neste documento com os algoritmos da literatura. Finalmente, a dissertação expandirá estes resultados para o caso que permite a rotação de itens.

## 1.1 Motivação

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 com 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, tal que 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 comparação de vários trabalhos científicos caso o gerador tenha fornecido instâncias com itens diferentes entre os ambientes computacionais utilizados. Este pode ter sido o caso do trabalho recente (HIFI; NEGRE; WU, 2014), que alega que alguns trabalhos reportam resultados melhores que os obtidos pelo algoritmo exato para alguns grupos de instâncias do 3BP.

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 provavelmente a solução ótima.

Desta maneira, este trabalho apresenta uma proposta de dissertação, onde novos algoritmos para o 3BP e o 2BP são testados preliminarmente com a base de teste padrão da literatura composta por 300 e 500 instâncias para os casos bi e tridimensional respectivamente. Objetiva-se obter novos resultados referentes as versões bi e tridimensional do problema, considerando itens de orientação fixa ou permitindo sua rotação. Os melhores resultados obtidos para cada instância serão disponibilizados em conjunto com a própria instância separadamente, de maneira que qualquer leitor interessado possa sintetizar a configuração das caixas com seus respectivos itens. A implementação do melhor método estabelecido na dissertação também será disponibilizado.

## 1.2 Estrutura deste documento

O restante do trabalho é organizado da seguinte forma:

- Capítulo 2: 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. Pretende-se estender este capítulo apresentando a prova de que o problema abordado é NP-Difícil.
- Capítulo 3: apresenta a revisão bibliográfica dos problemas de empacotamento bidimensional e tridimensional, destacando os principais trabalhos da literatura que propõem algoritmos para resolução da versão clássica deste problema. Além disso, este capítulo exibe e descreve um modelo de programação linear inteira para a versão tridimensional sem rotação. Objetiva-se estender este capítulo detalhando também o modelo para a versão tridimensional com rotação.
- Capítulo 4: 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 2. Além disso, o capítulo também fornece uma análise de complexidade para cada algoritmo.

- Capítulo 5: mostra os resultados computacionais preliminares do 3BP e do 2BP sem rotação através de uma versão promissora de cada método proposto no Capítulo 4. Esta versão usa a melhor configuração de componentes conhecida no momento da escrita deste trabalho. Além disso, o capítulo também descreve o ambiente computacional utilizado e as instâncias. Tais instâncias são a base de testes para diversos trabalhos da literatura. Portanto, os resultados preliminares são utilizados para comparar os algoritmos propostos neste trabalho com outros métodos estado da arte da literatura. Pretende-se determinar a melhor versão de cada algoritmo através de testes empíricos adicionais. Finalmente, a dissertação estenderá os resultados computacionais para os casos em que há rotação de itens.
- Capítulo 6: apresenta conclusões parciais para o trabalho e perspectivas futuras.

# Capítulo 2

## 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 os conceitos fundamentais de teoria da complexidade computacional, que são utilizados para analisar teoricamente a performance dos métodos propostos.

### 2.1 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 a quantidade de instruções primitivas da máquina que um algoritmo requer para sua execução completa. Assim, a complexidade de espaço ou memória é a análise da quantidade de células de memória que são utilizadas simultaneamente durante a execução do algoritmo, ou seja, a quantidade de espaço utilizada para o processamento do algoritmo em qualquer determinado ponto de execução. Os conceitos desta seção seguem como referência os livros (SZWARCFITER; MARKENZON, 2010; RESENDE; RIBEIRO, 2016).

#### 2.1.1 Notação O-grande

Existem diversas notações e maneiras de classificar a complexidade de um determinado recurso. Uma das classificações fundamentais é o 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:

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

Assim, temos uma família de notações que são amplamente utilizadas para evidenciar a ordem de grandeza deste termo de maior grau da função  $f(n)$ . Esta dissertação utiliza a notação **big-O** (O-grande) para analisar a complexidade de pior caso dos métodos propostos.

**Definição 2.1.1** Dizemos que  $f(n) = \mathcal{O}(g(n))$  se, para alguma constante  $C$  e um valor inicial  $n_0, \forall n \geq n_0$  temos  $f(n) \leq C \times g(n)$ .

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  $A, B$  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 expressa como um polinômio em  $n$  e *intratáveis* em caso contrário.

## 2.1.2 Notação Theta

Outra maneira de classificar a complexidade é através de uma função que define seu limite inferior e superior, cujo o objetivo é medir tanto a menor utilização possível do recurso em questão quanto a máxima. Assim, temos a seguinte definição:

**Definição 2.1.2** Dizemos que  $f(n) = \Theta(g(n))$  se, para duas constantes  $C_1, C_2$  e um valor inicial  $n_0, \forall n \geq n_0$ , temos  $f(n) \geq C_1 \times g(n)$  e  $f(n) \leq C_2 \times g(n)$ .

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

## 2.1.3 Problema de Decisão

O problema  $P$  é um problema de decisão quando o conjunto solução  $S$  é composto somente pelos elementos *Sim* e *Não*. Ou seja, os problemas de decisão consistem em responder *Sim* ou *Não*. Por exemplo:

- Seja o número  $A$  e um conjunto  $S$  de números. Existe algum número  $B \in S$  maior que  $A$ ?

- Seja  $G$  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 distância menor que  $B$  começando pela cidade  $A$ ?
- Seja o conjunto  $I$  de paralelepípedos e um conjunto finito  $C$  de cubos homogêneos. É possível empacotar ortogonalmente todos os paralelepípedos de  $I$  nos cubos do conjunto  $C$  sem sobreposição?

### 2.1.4 Problema de Otimização

Seja um problema  $P$ , o conjunto  $S = \{s_1, s_2, \dots, s_n\}$  das soluções viáveis e a função objetivo  $f : S \rightarrow \mathbb{R}$  que associa cada solução  $s_i \in S$ ,  $i = 1, 2, \dots, n$ , com um número real. O problema  $P$  é um problema de otimização de minimização se ele consiste em determinar uma solução  $s^* \in S \mid f(s^*) \leq f(s), \forall s \in S$ . Analogamente,  $P$  é um problema de otimização de maximização se ele procura determinar  $s^* \in S \mid f(s^*) \geq f(s), \forall s \in S$ . 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. Em outras palavras, encontrar a melhor solução de todas.

Considere o conjunto  $X = \{x_1, x_2, \dots, x_n\}$  das 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 de valores.
- Otimização combinatória: são os problemas onde toda variável  $x \in X$  é discreta, ou seja, pode assumir um número finito ou infinito contável de valores.

Desta maneira, problemas de otimização combinatória tem um número finito de soluções viáveis. Todo o 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  $A \in S$ .
- Seja  $G$  um mapa de rotas entre cidades e  $A \in G$  uma cidade, encontre a distância da menor rota que visita todas as cidades de  $G$  começando em  $A$ .
- Seja o conjunto  $I$  de paralelepípedos e um conjunto  $C$  de cubos homogêneos, determine o menor número possível de cubos em  $C$  que empacotam ortogonalmente todos os paralelepípedos de  $I$  sem sobreposição.

### 2.1.5 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 os resolve. Ou seja, são os problemas de decisão em que há uma máquina de *Turing* determinística que resolve o problema em tempo polinomial.
- 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. Ou seja, 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 tempo polinomial.
- Classe NP-Difícil: é a classe que consiste dos problemas  $\pi$  tal que todo problema  $\pi_L \in NP$  pode ser reduzido em tempo polinomial para  $\pi$ . Ou seja, são os problemas para os quais existe uma máquina de *Turing* determinística que transforma o problema  $\pi$  no problema  $\pi_L$  em tempo polinomial, desta maneira, resolver  $\pi_L$  é equivalente a resolver  $\pi$ .
- Classe NP-Completo: é a classe que consiste dos problemas de decisão que pertencem às classes NP e NP-Difícil.

## 2.2 Heurística

**Heurística** é um algoritmo que resolve um determinado problema sem garantias sobre a qualidade da solução encontrada ou sobre sua performance. Geralmente, heurísticas trocam 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 ótimas ou perfeitas.

### 2.2.1 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. 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 2.1 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 2.1 Construtivo Guloso

---

```

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

```

---

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 2.2 descreve o pseudocódigo de um guloso adaptativo, onde  $g$  é a função que avalia o potencial de qualidade de um candidato. (RESENDE; RIBEIRO, 2016)

Nos problemas de empacotamento, as heurísticas para efetuar o empacotamento dos itens são gulosas, isto é, 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

---

**Algoritmo 2.2** Construtivo Guloso Adaptativo

---

```

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

```

---

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. Estes processos de empacotamento são chamados de heurísticas construtivas gulosas.

### 2.2.2 Busca Local

Seja 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  $k \in \{1, 2, \dots, n\}$  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. Ou seja, 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.

### 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  $s^k$ ,  $k \in \{1, 2, \dots, n\}$ . Portanto, a vizinhança  $V(s) = \{s^1, s^2, \dots, s^n\}$  sobre a estrutura de vizinhança  $V$  com  $n$  movimentos é obtida aplicando cada movimento  $M_k$  na solução  $s$ . As soluções  $s^1, s^2, \dots, s^n \in V(s)$  são chamados 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. Como o 3BP e o 2BP não precisam de tais técnicas, pois sempre são trabalhados com soluções viáveis, no restante desta dissertação este detalhe não será mais discutido.

### 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 2.3 descreve o procedimento realizado pelo First Improvement para um problema de minimização.

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 2.4 detalha este procedimento para um problema de minimização.

Não há nenhuma garantia de que o Best Improvement encontre soluções melhores que

---

**Algoritmo 2.3** First Improvement
 

---

```

1: função FIRSTIMPROVEMENT(Solução:  $s$ ; Estrutura de vizinhança:  $V$ )
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  $s$ 
4:       retorna  $s_k$                                               ▷ Seleciona o primeiro vizinho que melhora  $s$ 
5:     fim se
6:   fim para
7:   retorna  $s$                                                     ▷ Se um vizinho melhor  $s$  não existir, retorna o próprio.
8: fim função

```

---



---

**Algoritmo 2.4** Best Improvement
 

---

```

1: função BESTIMPROVEMENT(Solução:  $s$ ; Estrutura de vizinhança:  $V$ )
2:    $s^* \leftarrow s$                                               ▷ Melhor solução  $s^*$  recebe solução  $s$ 
3:   para cada  $s_k \in V(s)$  faça                                    ▷ Para cada vizinho de  $s$ 
4:     se  $f(s_k) < f(s^*)$  então  ▷ Se  $s_k$  for melhor que a melhor solução encontrada
5:        $s^* \leftarrow s^k$                                           ▷ Mantém a melhor solução em  $s^*$ 
6:     fim se
7:   fim para
8:   retorna  $s^*$                                                   ▷ Retorna a melhor solução encontrada
9: fim função

```

---

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 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 2.5 apresenta a busca local, onde o procedimento *Improvement* pode ser o First Improvement ou o Best Improvement.

---

**Algoritmo 2.5** Busca Local
 

---

```

1: função BUSCALOCAL(Solução inicial:  $s_0$ ; Estrutura de vizinhança:  $V$ )
2:    $s^* \leftarrow s_0$                                               ▷ Melhor solução  $s^*$  recebe solução inicial
3:   faça
4:      $s^* \leftarrow \text{IMPROVEMENT}(s^*, V)$   ▷ Best Improvement ou First Improvement
5:   enquanto Houver melhora em  $s^*$ 
6:     retorna  $s^*$                                                   ▷ Retorna a melhor solução encontrada
7: fim função

```

---

## Ótimo Local vs. Ótimo Global

Seja 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:

- $\nexists s' \in S \mid f(s') > f(s^*)$  caso o problema  $P$  seja de maximização
- $\nexists s' \in S \mid f(s') < f(s^*)$  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  $V$ . Ou seja, 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  $V(c) \subset S$ .

### 2.2.3 Variable Neighborhood Descent (VND)

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 número de soluções. Este conceito é apresentado graficamente na Figura 2.1, onde soluções são dispostas em um plano. Duas vizinhanças, representadas respectivamente pelas cores vermelho e azul, são compostas por diferentes soluções. A primeira vizinhança é composta por soluções na diagonal da solução inicial, e a segunda por soluções nos eixos horizontal e vertical. Pela figura podemos observar que ambas as vizinhanças percorrem soluções distintas no espaço em direções diferentes. Assim, a ideia do VND é **combinar diferentes estruturas de vizinhança** para buscar soluções através de um método que utiliza mais de uma busca local.

O VND é uma heurística que percorre o espaço de soluções em diferentes direções. Percorrer o espaço de soluções com estruturas de vizinhanças que realizam vários movimentos distintos pode atingir soluções muito boas, porém o tempo de execução aumenta devido ao maior número de soluções avaliadas. Analogamente, uma vantagem do VND



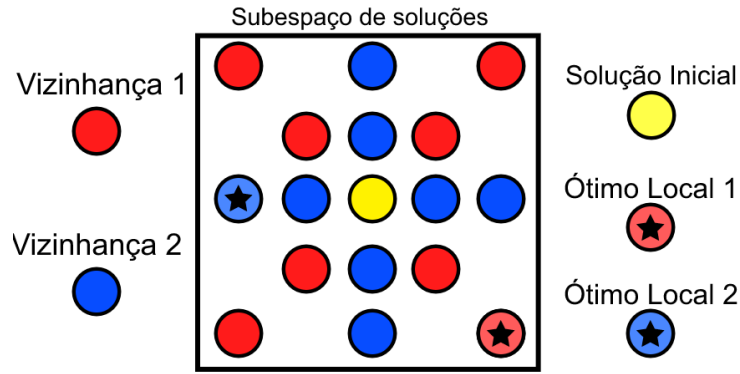


Figura 2.1: Representação gráfica de um subespaço de soluções com duas vizinhanças distintas de uma mesma solução inicial.

em relação à busca local é o seu potencial em percorrer um maior número de soluções em busca de múltiplos ótimos locais. Uma desvantagem é o maior custo computacional em termos de tempo de execução devido ao maior número de soluções para avaliar.

O VND recebe uma solução inicial  $s_0$  e  $n$  estruturas de vizinhança como parâmetro de entrada. Começando pela estrutura de vizinhança  $V_1$ , o procedimento realiza uma busca local com cada  $V_k$  sequencialmente,  $k \in \{1, 2, \dots, n\}$ . 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, ou seja,  $s$  é ótimo local em relação a todas  $V_k(s)$  vizinhanças.

A performance do VND está diretamente ligada com a 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 aumentam conforme 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 2.6 detalha o VND para um problema de otimização combinatória, onde a função *BuscaLocal* é o Algoritmo 2.5. (CAPOROSI; HANSEN; MLADENOVIĆ, 2016)

---

**Algoritmo 2.6** Variable Neighborhood Descent

---

```

1: função VND(Solução inicial:  $s_0$ , Vetor de estruturas de vizinhança:  $V$ )
2:    $s^* \leftarrow s_0$  ▷ Melhor solução.
3:    $k \leftarrow 1$  ▷ O processo começa em  $V_1$ 
4:   enquanto  $k < \text{Tamanho de } V$  faça
5:      $s^* \leftarrow \text{BUSCALOCAL}(s^*, V_k)$  ▷ Busca um ótimo local conforme  $V_k(s^*)$ 
6:     se Houver melhora em  $s^*$  então
7:        $k \leftarrow 1$ 
8:     senão
9:        $k \leftarrow k + 1$ 
10:    fim se
11:  fim enquanto
12:  retorna  $s^*$  ▷ Retorna a melhor solução
13: fim função

```

---

### 2.2.4 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 busca local.

Por exemplo, considere o problema clássico da mochila que é NP-Difícil: dada uma mochila de capacidade  $B$  e um conjunto de  $n$  itens, cada um caracterizado por seu peso  $w_i > 0$  e seu valor  $v_i > 0$ ,  $i \in \{1, 2, \dots, n\}$ , determinar o subconjunto de itens que podem ser carregados pela mochila tal que a soma de valores seja máxima e de modo que a soma dos pesos não ultrasse a capacidade  $B$ , sabendo que  $\sum_{i=1}^n w_i > B$ . 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 pode ser representada pelo conjunto de tuplas:  $\{i, j\} \mid i \neq j \text{ e } n \geq i, j \geq 1$ .

Nos problemas de empacotamento, diversas representações de soluções são utilizadas. Uma delas utiliza soluções codificadas. Representações de soluções codificadas precisam de um decodificador determinístico para obter uma solução do problema 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 heu-

rísticas 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 2.3.4.

## 2.3 Meta-heurística

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, 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 3 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.

### 2.3.1 Greedy Randomized Adaptive Search Procedure

O GRASP (*Greedy Randomized Adaptive Search Procedure*) é uma meta-heurística composta de duas fases no seu processo iterativo, onde o método recomeça de uma nova solução construída desde o princípio, a cada iteração. Uma referência atual para o GRASP e suas

aplicações é o livro (RESENDE; RIBEIRO, 2016), no qual esta seção se baseia.

### Fase de construção

A primeira fase do GRASP, chamada de fase de construção, recebe como parâmetro de entrada um número real  $1 \geq \alpha \geq 0$  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  $\alpha$  para introduzir aleatoriedade ao guloso adaptativo. Assim, o algoritmo semi-guloso deve sempre considerar uma lista com os  $100 \cdot \alpha\%$  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 2.2. 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  $\alpha$  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  $\alpha$  seja 1, o processo se torna completamente aleatório, pois a RCL será composta por todos os candidatos. O Algoritmo 2.7 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 2.7 Semi-Guloso ou Guloso Randomizado

---

**Requerimento:** Parâmetro  $\alpha$  no intervalo  $[0, 1]$

```

1: função SEMI-GULOSO(Número real:  $\alpha$ )
2:    $s \leftarrow \emptyset$  ▷ Solução construtiva começa desdo princípio.
3:   enquanto  $s$  não está completo faça
4:     Lista  $\leftarrow$  Vazio ▷ Lista de candidatos
5:     para cada possível candidato  $c$  faça ▷ Cada passo seleciona um  $c$  para  $s$ 
6:       Lista  $\leftarrow \{c, g(c)\}$  ▷ Adiciona  $c$  pela ordem de seu potencial  $g(c)$  na lista
7:     fim para
8:     RCL  $\leftarrow$   $100 \cdot \alpha\%$  melhores candidatos da Lista
9:      $c' \leftarrow$  Candidato aleatório da RCL
10:     $s \leftarrow s \cup \{c'\}$  ▷ Adiciona o candidato em  $s$ 
11:  fim enquanto
12:  retorna  $s$  ▷ Retorna a solução construída
13: fim função

```

---

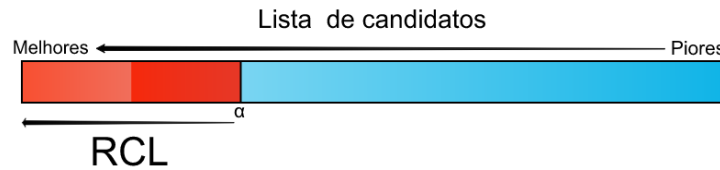


Figura 2.2: Representação gráfica do vetor de candidatos e do RCL delimitado pelo  $\alpha$ .

### 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 2.8 apresenta o pseudocódigo do GRASP para um problema de minimização, onde  $f$  é a função objetivo associado ao problema de otimização combinatória que está sendo resolvido, a função *Semi-Guloso* é o processo semi-guloso do Algoritmo 2.7 e a *BuscaLocal* é o Algoritmo 2.5 com Best Improvement ou First Improvement.

---

#### Algoritmo 2.8 Greedy Randomized Adaptive Search Procedure

---

**Requerimento:** Parâmetro  $\alpha$  no intervalo  $[0, 1]$

```

1: função GRASP(Número Real:  $\alpha$ )
2:   enquanto Critério de parada não satisfeito faça
3:      $s \leftarrow \text{SEMI-GULOSO}(\alpha)$            ▷ Constrói uma solução gulosa aleatória.
4:      $s \leftarrow \text{BUSCALOCAL}(s)$              ▷ Best ou First
5:     se  $f(s) < f(s^*)$  então
6:        $s^* \leftarrow s$                        ▷ Mantém a melhor solução
7:     fim se
8:   fim enquanto
9:   retorna  $s^*$                                ▷ Retorna a melhor solução encontrada
10: fim função

```

---

### 2.3.2 Busca Tabu

A **Busca Tabu** 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 segue 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. Este critério é uma regra que cria 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 mecanismos que envolvem sintetizam memória dinâmica que aceita muitas soluções, 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.

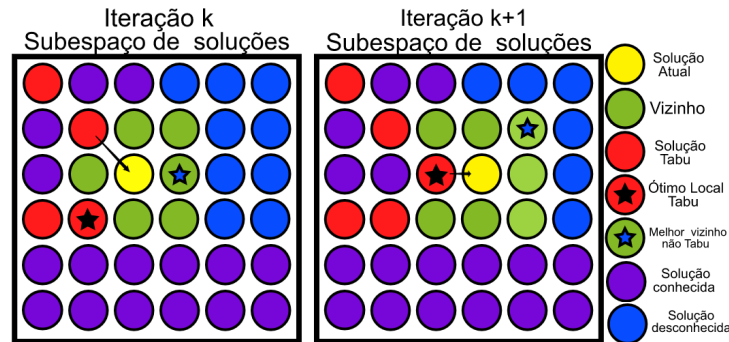


Figura 2.3: Exemplo gráfico da execução de uma Busca Tabu com duas iterações arbitrárias representadas em um subespaço de soluções.

---

**Algoritmo 2.9** Busca Tabu

---

```

1: função BUSCATABU(Solução inicial:  $s_0$ ; Estrutura de vizinhança:  $V$ ; Tama-
   nho máximo do Tabu:  $T_{max}$ )
2:    $s^*, s \leftarrow s_0$            ▷ Melhor solução  $s^*$  e solução corrente  $s$  recebem solução inicial
3:    $T \leftarrow \emptyset$            ▷ Lista tabu começa vazia
4:   enquanto critério de parada não satisfeito faça
5:      $s' \leftarrow \text{Melhor } V(s)$            ▷ Melhor vizinho da solução corrente
6:     se  $s' \in T$  então           ▷ Se  $s'$  é uma solução tabu ou o movimento é tabu
7:       se  $s'$  atende o critério de aspiração então
8:         ATUALIZA( $T$ )           ▷ Remove  $s'$  ou o movimento da lista tabu
9:       senão
10:        Descarta  $s'$  de  $V(s)$  e volte para a linha 5   ▷ Ache o próximo melhor
11:      fim se
12:    fim se
13:    se  $s'$  é melhor que  $s^*$  então
14:       $s^* \leftarrow s'$            ▷ Nova melhor solução.
15:    fim se
16:     $s \leftarrow s'$            ▷ Solução corrente recebe o melhor vizinho não tabu
17:    ATUALIZA( $T, T_{max}, s'$ )       ▷ Atualiza o tabu de acordo com  $T_{max}$  e  $s'$ 
18:  fim enquanto
19:  retorna  $s^*$            ▷ Retorna a melhor solução encontrada
20: fim função

```

---

A Figura 2.3 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 2.9 apresenta a busca tabu para um problema de otimização de minimização, onde *Atualiza* é o procedimento que realiza o gerenciamento da lista tabu.

### 2.3.3 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 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).

#### Algoritmo Genético

O algoritmo genético 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  $f : S \rightarrow R$  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 variáveis 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  $\Sigma$ . Os elementos de  $\Sigma$  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 o mesmo cromossomo 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  $I_1$  e  $I_2$  são denominados de *parentes*, 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 2.4 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 2.5 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  $\omega$  deve ser um número pequeno para afetar somente alguns poucos indivíduos. A mutação é um operador  $\oplus$  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*.

Vale a pena enfatizar que os mecanismos descritos nas fases acima podem 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

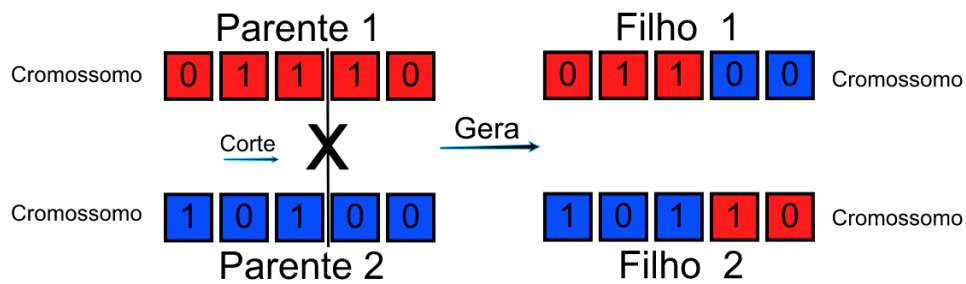


Figura 2.4: Exemplo gráfico do operador de reprodução *one-point*.

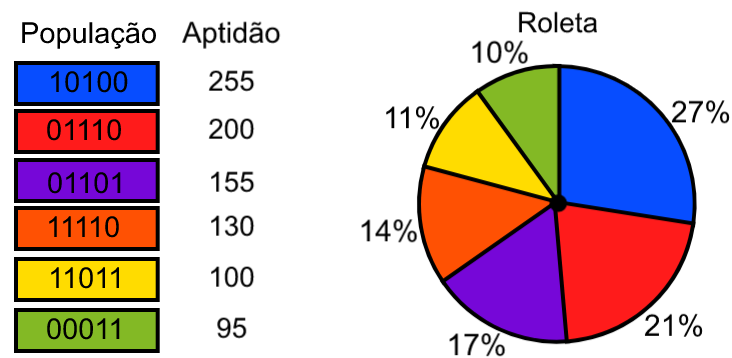


Figura 2.5: Exemplo gráfico da estratégia da roleta.

de entrada calibrável. O Algoritmo 2.10 mostra um *framework* para o algoritmo genético através das fases descritas acima, onde os parâmetros de entrada pode variar de acordo com a estratégia aplicada em cada etapa.

### 2.3.4 Biased Random-Keys Genetic Algorithm

O BRKGA (*Biased Random-Keys Genetic Algorithm*) é uma meta-heurística classificada como um algoritmo evolucionário. Portanto, os conceitos descritos na Seção 2.3.3 também valem são utilizados nesta seção. O método é uma variação do RKGA (*Random-Keys Genetic Algorithm*), que por sua vez 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, entretanto, 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 *Random-Keys*, na forma de vetores de números reais no intervalo  $[0, 1]$ . A implementação desse processo requer um componente chamado de-

---

**Algoritmo 2.10** Algoritmo Genético
 

---

```

1: função AG(Quantidade de indivíduos:  $P_{max}$ ; Taxa de seleção:  $\alpha$ ; Operador
   de reprodução:  $\oplus$ ; Operador de mutação  $\uplus$ ; Taxa de mutação:  $\omega$ )
2:    $P \leftarrow \text{POPULAÇÃOINICIAL}(P_{max})$   $\triangleright$  População inicial com  $P_{max}$  indivíduos
   aleatórios
3:   AVALIA( $P$ )  $\triangleright$  Mantém os indivíduos ranqueados por aptidão
4:   enquanto critério de parada não satisfeito faça
5:      $P \leftarrow \text{SELEÇÃO}(P, \alpha)$   $\triangleright$  Seleciona os  $100\alpha\%$  melhores indivíduos
6:      $\text{REPRODUÇÃO}(P, \oplus, P_{max})$   $\triangleright$  Preenche a população  $P$  com novos indivíduos
7:      $\text{MUTAÇÃO}(P, \uplus, \omega)$   $\triangleright$  Introduz  $100\omega\%$  mutantes na população
8:     AVALIA( $P$ )  $\triangleright$  Mantém os indivíduos ranqueados por aptidão
9:   fim enquanto
10:   $I^* \leftarrow$  Melhor indivíduo de  $P$ 
11:  retorna  $I^*$   $\triangleright$  Retorna o melhor indivíduo
12: fim função

```

---

codificador. 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  $P_\epsilon \subset P \mid |P_\epsilon| < n - |P_\epsilon|$ . A cada geração, uma nova população  $P'$  é construída com  $P_{max}$  indivíduos. Os indivíduos de  $P_\epsilon$  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  $P_\omega$  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 2.6 apresenta graficamente a construção da nova população a cada geração, destacando a população elite e mutante dado 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.

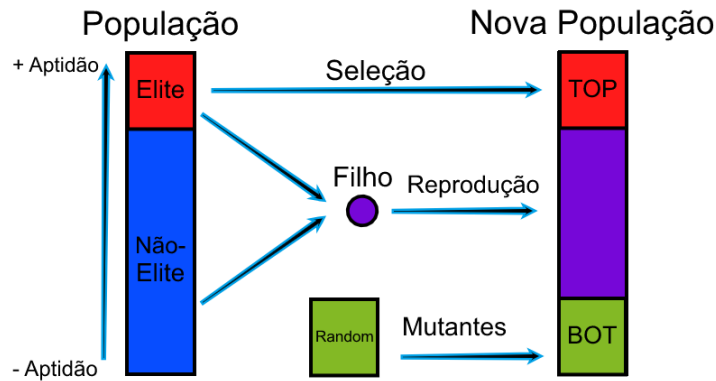


Figura 2.6: Construção de uma geração do RKGA ou BRKGA.

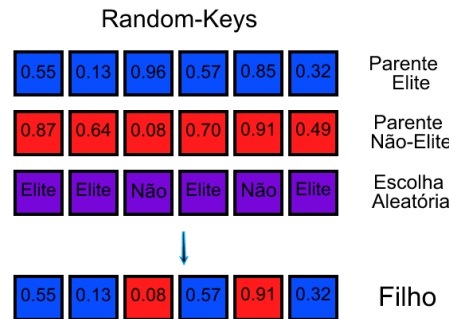


Figura 2.7: Exemplo de reprodução no BRKGA. A probabilidade de escolha do gene elite é dado pelo parâmetro  $\rho$ .

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 2.7. 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  $\rho$ . 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 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 2.11 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 2.11** Biased Random-Keys Genetic Algorithm
 

---

```

1: função BRKGA(Quantidade de indivíduos:  $P_{max}$ ; TOP:  $\epsilon$ ; BOT:  $\omega$ ; Taxa da
   reprodução:  $\rho$ )
2:    $P \leftarrow \text{POPULAÇÃOINICIAL}(P_{max})$   $\triangleright$  População inicial com  $n$  indivíduos aleatórios
3:    $\text{DECODIFICA\_AVALIA}(P)$   $\triangleright$  Mantém os indivíduos ranqueados por aptidão
   através do decodificador
4:   enquanto critério de parada não satisfeito faça
5:      $P' \leftarrow \text{SELEÇÃO}(P, \epsilon)$   $\triangleright$  Copia os elites para  $P'$ 
6:      $\text{MUTAÇÃO}(P', \omega)$   $\triangleright$  Introduz  $100\omega\%$  indivíduos aleatórios em  $P'$ 
7:      $\text{REPRODUÇÃO}(P', \rho, P_{max})$   $\triangleright$  Preenche  $P'$  com novos indivíduos
8:      $\text{DECODIFICA\_AVALIA}(P')$   $\triangleright$  Mantém os indivíduos ranqueados por aptidão
   através do decodificador
9:      $P \leftarrow P'$   $\triangleright$  Finaliza a geração atual
10:  fim enquanto
11:   $I^* \leftarrow$  Melhor indivíduo de  $P$ 
12:  retorna  $I^*$   $\triangleright$  Retorna o melhor indivíduo
13: fim função

```

---

# Capítulo 3

## Literatura

Este capítulo revisa os principais trabalhos da literatura que estabelecem um limite inferior ou propõem algoritmos para o 3BP e o 2BP. Além disso, este capítulo também apresenta um modelo de programação linear inteira mista para o 3BP sem rotação. Recentemente, os métodos propostos que resolvem o 3BP também solucionam o 2BP. Geralmente, os algoritmos propostos na literatura são submetidos a um teste empírico utilizando um conjunto padrão de instâncias, a fim de realizar uma comparação computacional, em termos de qualidade da solução obtida, entre os diversos algoritmos em distintos ambientes computacionais.

### 3.1 Modelo de Programação Linear Inteira Mista

Como o espaço de soluções viáveis não é mais convexo após o empacotamento de um item, é difícil construir um modelo simples de programação linear inteira para o 3BP, assim, esta seção descreve o modelo de programação linear inteira mista (MPLIM) proposto por (HIFI et al., 2010).

Dado um conjunto  $I = \{a_1, a_2, \dots, a_n\}$  de  $n$  itens com orientação fixa 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 de caixas  $Q = \{C_1, C_2, \dots\}$  homogêneas com largura  $W$ , altura  $H$  e profundidade  $D$  inteiros positivos. O problema de empacotamento tridimensional clássico consiste em determinar um número inteiro positivo  $\lambda$  e uma partição associada  $S = S_1 \cup S_2 \cup \dots \cup S_\lambda = \{a_1, a_2, \dots, a_n\}$ , tal que  $S_j \cap S_k = \emptyset$ ,  $1 \leq j \neq k \leq \lambda$ , onde cada item  $a_i \in S_j$  é empacotado ortogonalmente sem sobreposição na caixa  $C_j \in Q$ . A solução  $\lambda$  é ótima se o seu  $S$  é mínimo.

Começando por 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 representa a própria caixa. O objetivo do modelo é minimizar o número  $\lambda = \max(\{\lambda_1, \lambda_2, \dots, \lambda_n\})$  de caixas utilizadas.

O modelo de programação linear inteira mista, denominado MPLIM, pode ser formulado da seguinte maneira:

$$\begin{aligned}
& \text{Minimizar } \lambda & (\text{MPLIM}) \\
& \text{Sujeito a} \\
& l_{ij} + l_{ji} + u_{ij} + u_{ji} + b_{ij} + b_{ji} + c_{ij} + c_{ji} = 1, & i < j = 1, 2, \dots, n \quad (1) \\
& x_i - x_j + W(l_{ij} - c_{ij} - c_{ji}) \leq W - w_i, & i \neq j = 1, 2, \dots, n \quad (2) \\
& y_i - y_j + H(u_{ij} - c_{ij} - c_{ji}) \leq H - h_i, & i \neq j = 1, 2, \dots, n \quad (3) \\
& z_i - z_j + D(b_{ij} - c_{ij} - c_{ji}) \leq D - d_i, & i \neq j = 1, 2, \dots, n \quad (4) \\
& (\hat{\lambda} - 1)(l_{ij} + l_{ji} + u_{ij} + u_{ji} + b_{ij} + b_{ji}) + \lambda_i - \lambda_j + \hat{\lambda}c_{ij} \leq \hat{\lambda} - 1, & i \neq j = 1, 2, \dots, n \quad (5) \\
& l_{ij}, u_{ij}, b_{ij}, c_{ij} \in \{0, 1\}, & i \neq j = 1, 2, \dots, n \\
& 0 \leq x_i \leq W - w_i, & i = 1, 2, \dots, n \\
& 0 \leq y_i \leq H - h_i, & i = 1, 2, \dots, n \\
& 0 \leq z_i \leq D - d_i, & i = 1, 2, \dots, n \\
& 0 \leq \lambda_i \leq \lambda \leq \hat{\lambda}, & i = 1, 2, \dots, n
\end{aligned}$$

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 dois itens não se sobrepõem é imposta pelas restrições (1, 2, 3, 4). A restrição 5 implica que quando  $c_{ij} = 1$  ou  $c_{ji} = 1$ , então os itens  $a_i$  e  $a_j$  estão localizados em diferentes caixas. Assim, 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  $\lambda$ . 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  $\Theta(n)$  que utiliza no máximo  $n$  caixas. O Algoritmo 3.1 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 3.1 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 3.1** First Fit
 

---

```

1: função FIRSTFIT(Lista de Itens:  $I$ ; Lista de caixas:  $Q$ )
2:    $\hat{\lambda} \leftarrow 0$  ▷ Quantidade de caixas utilizadas
3:   para  $a_i \in I$  faça ▷ Para cada item  $a_i$ ,  $1 \leq i \leq n$ , no conjunto  $I$ 
4:      $j \leftarrow 1$ 
5:     enquanto  $j \leq \hat{\lambda}$  faça ▷ Para cada caixa aberta
6:       se é possível empacotar  $a_i$  em  $C_j \in Q$  então
7:         EMPACOTA( $a_i, C_j$ ) ▷ Empacota o item em  $C_j$ 
8:         Finaliza o loop enquanto
9:       fim se
10:      se  $a_i$  não foi empacotado então
11:         $\hat{\lambda} \leftarrow$  Abre uma nova caixa
12:        EMPACOTA( $a_i, C_{\hat{\lambda}}$ ) ▷ Empacota na nova caixa
13:      fim se
14:    fim enquanto
15:  fim para
16:  retorna  $\hat{\lambda}$  ▷ Upper Bound para o MPLIM
17: fim função

```

---

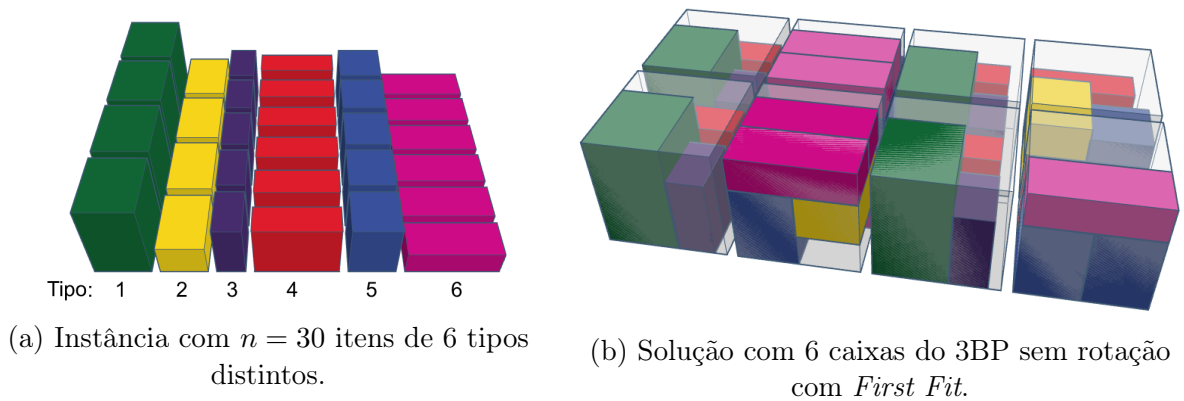


Figura 3.1: Exemplo de uma instância do 3BP sem rotação solucionada através do *First Fit* apresentado no Algoritmo 3.1. O empacotamento utiliza o ponto inferior esquerdo frontal como referência. A ordem de empacotamento dos itens é dada pelo tipo: a primeira iteração aloca um de tipo 1, na segunda um de tipo 2 e assim por diante.

## 3.2 Revisão da Literatura

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 é uma solução parcial do 3BP. Porém, Boef et al. (BOEF et al., 2005) demonstram que o algoritmo exato só é 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. Este procedimento é utilizado em (MARTELLO; PISINGER; VIGO, 2000) para demonstrar um limite inferior para o problema tridimensional.

Em (FEKETE; SCHEPERS, 2004a) lida-se com o problema em sua versão multi-dimensional, modelando soluções através de uma representação com grafo. 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. Deste modo, em (FEKETE; SCHEPERS, 2004b; FEKETE; SCHEPERS; VEEN, 2007; FEKETE; VEEN, 2007) é proposto um *framework Branch & Bound* para resolver instâncias grandes do problema clássico de empacotamento com dimensão arbitrária (nBP). O modelo de programação linear inteira mista apresentado na Seção 3.1 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 (HIFI; NEGRE; WU, 2014).

Os trabalhos citados anteriormente, (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). (FEKETE; SCHEPERS, 2001) provê um novo limite inferior para ambos os problemas que domina o limite demonstrado por (MARTELLO; PISINGER; VIGO, 2000). (BOSCHETTI; MINGOZZI, 2003a, 2003b; BOSCHETTI, 2004) combinam os resultados dos trabalhos anteriores com heurísticas para melhorar os limites propostos. Atualmente, os autores de (LIAO; HSU, 2013) revisam um limite inferior do 1BP para estabelecer uma extensão para o 3BP sem rotação dominante 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) apresenta um limite inferior que domina o limite do trabalho anterior para o 3BP com rotação.

Vários algoritmos construtivos e heurísticas foram propostos para resolver problemas de empacotamento. 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. Assim, 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. Então, 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 designado para resolver o problema de carregamento de contêiner proposto em (PARREÑO et al., 2008). (CRAINIC; PERBOLI; TADEI, 2012) combina um algoritmo guloso adaptativo com mecanismos de aprendizado e evolução para sintetizar um *framework* para os problemas de empacotamento tridimensional com caixas homogêneas ou heterogêneas. Este método pode ser aplicado em diversas variações do problema de empacotamento com itens orientados ou não.

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. 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 decrescente. 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.

## Capítulo 4

# Heurísticas para o 3BP e 2BP

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 de grande porte, 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.

### 4.1 Representação da Solução

No caso tridimensional, a representação computacional de um item  $a_i$ ,  $1 \leq i \leq n$ , é dado pelo vetor  $(h_i, w_i, d_i, x_i, y_i, z_i, \lambda_i)$  de variáveis inteiras positivas. Para cada item,  $h_i$  é a sua altura,  $w_i$  sua largura,  $d_i$  sua profundidade,  $(x_i, y_i, z_i)$  as coordenadas cartesianas que representam sua localização dentro de sua caixa e  $\lambda_i$  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 a posição de seu respectivo canto. Neste 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  $S = \{C_1, C_2, \dots, C_\lambda\}$  com  $\lambda$  caixas e as variáveis  $(H, W, D)$  inteiras positivas

para a altura, largura e profundidade das caixas. Cada caixa  $C_j \in S$ ,  $1 \leq j$  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  $\lambda$  em conjunto com os valores  $\lambda_i$  que ditam em qual caixa o item  $a_i$  está presente.

## 4.2 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, (GONÇALVES; RESENDE, 2013) propôs 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. Desta maneira, a Equação 4.1 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.

$$FOA(s) = FOT(s) + \frac{MC(s)}{C} \quad (4.1)$$

Por exemplo, considere duas soluções  $s_1$  e  $s_2$  distintas que utilizam 3 caixas cada, ou seja,  $N(s_1) = FOT(s_2) = 3$ . 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 à  $s_2$ .

## 4.3 Espaço Maximal

Esta seção apresenta um componente fundamental para o funcionamento do algoritmo de empacotamento da Seção 4.4, que, por sua vez, é utilizado na heurística construtiva, presente na Seção 4.5, 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 caixa aberta, isto é, uma caixa com pelo menos um item já empacotado. Esta modelagem também é utilizado 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, 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 4.1, 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 4.5, 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 a 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 4.1a e 4.1b 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 associado 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 4.4.

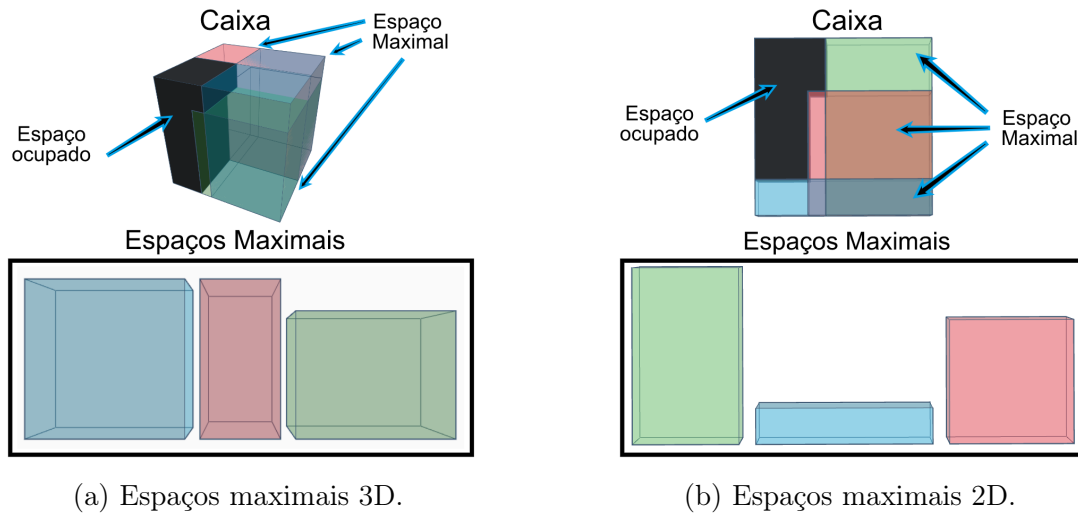


Figura 4.1: Exemplos de EM modelando o espaço vazio em duas caixas.

## 4.4 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 4.5, 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 4.5.

O algoritmo de empacotamento deve formar novos EM quando um item é empacotado em uma caixa aberta. Como descrito na Seção 4.3, 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 4.2 e 4.3 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. Este caso pode ser identificado com uma simples verificação. 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 depende do critério de avaliação de um EM feito pela heurística construtiva. Tais critérios são introduzidos nas Seções 4.5.1 e 4.5.2.



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 considerar 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 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 a performance das aplicações em aproximadamente 33%. O Algoritmo 4.1 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 4.1** 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.

- 1: **procedimento** EMPACOTA(**Item:**  $a$ ; **Caixa:**  $C$ ; **Espaço Maximal:**  $E$ )
  - 2:     REMOVE( $E, C$ ) ▷ Remove o espaço maximal da caixa
  - 3:     CORTE( $E, a, C$ ) ▷ Corta o antigo EM baseado no item e insere os novos na caixa
  - 4:     ATUALIZA( $C, a$ ) ▷ Filtra e atualiza o restante dos EM em  $C$
  - 5: **fim procedimento**
- 

#### 4.4.1 Complexidade

Esta seção apresenta complexidades de tempo, estimadas pela contagem de instruções elementares, do método de empacotamento presente no Algoritmo 4.1.

De acordo com as representações computacionais de um item e um EM, descritas

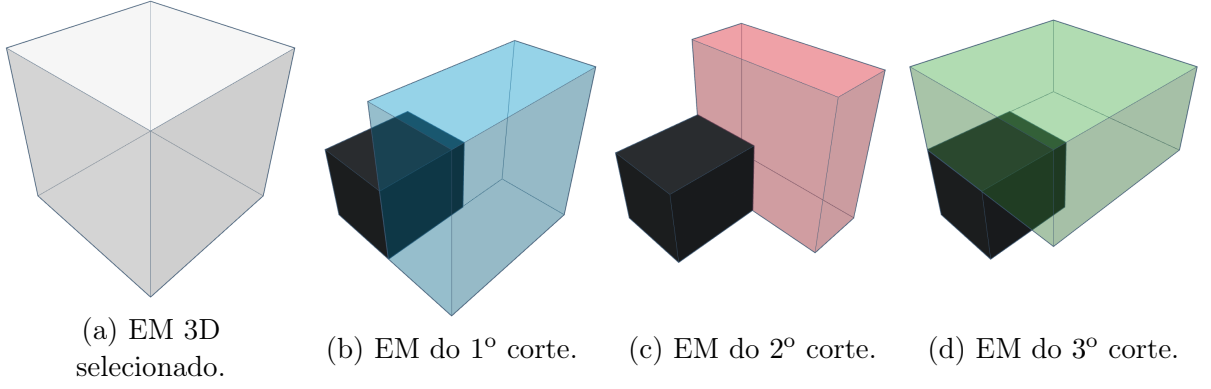


Figura 4.2: Exemplo de corte em um EM tridimensional.

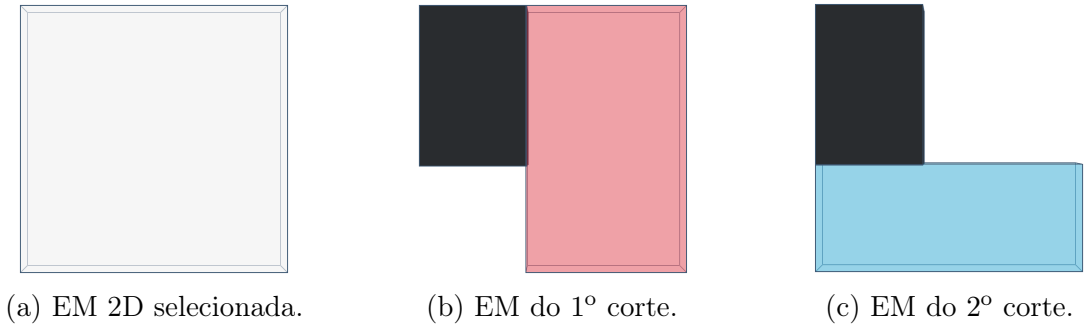


Figura 4.3: Exemplo de corte em um EM bidimensional.

respectivamente nas Seções 4.1 e 4.3, 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. Ou seja, o acesso e a inserção são efetuadas em tempo constante  $\Theta(1)$  e a remoção em complexidade linear  $\mathcal{O}(m)$ , tal que o parâmetro  $m$  majora o número de EM presentes em qualquer caixa aberta utilizada durante toda a execução da heurística construtiva ou da meta-heurística. O algoritmo de corte, presente na linha 3 do Algoritmo 4.1, que forma novos paralelepípedos e retângulos e os introduz na caixa tem complexidade constante  $\Theta(1)$ , pois o corte do EM pode ser implementado através de operações com números inteiros em um número constante de passos, e a inserção também tem complexidade constante. Observe que o filtro adicional para espaços redundantes, especificado na Seção 4.3, não afeta a complexidade do corte, pois se trata de algumas cláusulas condicionais.

A estrutura dinâmica que armazena cada EM sofre diversas alterações pelo procedimento presente na linha 4 do Algoritmo 4.1 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 é verificar a interseção entre um paralelepípedo ou retângulo com no máximo  $m$  outros no espaço contido pela caixa. Assim, é necessário ve-

rificar cada EM no máximo uma vez. Este passo deve sintetizar  $k \leq m$  novos EM através do mesmo processo de corte da linha 3, porém em cada EM que apresenta interseção com o item. Além disso, o EM de interseção 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 com a representação computacional especificada nesta seção para o item e o EM, a interseção pode ser verificada com duas cláusulas condicionais. O processo de atualização prossegue 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 menor que  $m$ . A Tabela 4.1 sumariza as complexidades de tempo associadas aos procedimentos que sintetizam o Algoritmo 4.1 em conjunto com a própria complexidade final do método. A tabela mostra que a complexidade final do procedimento 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 4.3 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 ou meta-heurística 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 4.1: Complexidade dos procedimentos que compõem o Algoritmo 4.1.

Procedimento	Passo	Complexidade	Observação
Remove	2	$\mathcal{O}(m)$	Utilizando um vetor de acesso direto como estrutura auxiliar.
Corte	3	$\Theta(1)$	
Atualiza	4	$\mathcal{O}(m^2)$	
Algoritmo 4.1	Final	$\mathcal{O}(m^2)$	

## 4.5 Heurística Construtiva

A qualidade das soluções encontradas nas meta-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 meta-heurísticas 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.

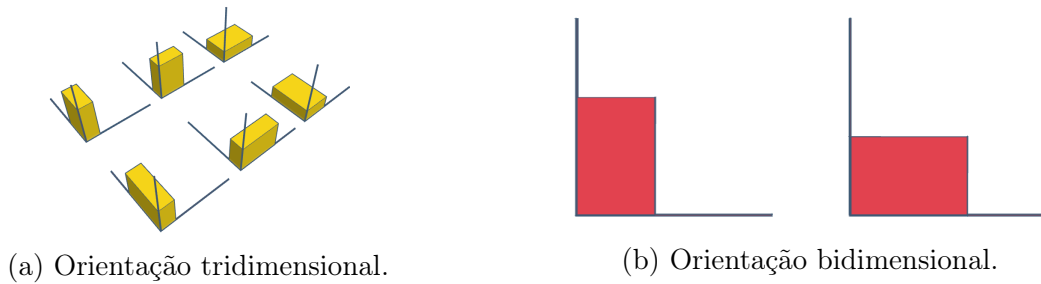


Figura 4.4: Exemplos que ilustram as possíveis orientações de um item.

Considere o conjunto de  $n$  itens  $I = \{a_1, a_2, \dots, a_n\}$ . 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 um empacotamento final com todos os itens seja sintetizado. 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 são dados,  $1 \leq i \leq n$ , o respectivo item  $a_i$  é submetido ao processo de empacotamento descrito na Seção 4.4. 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  $S = \{C_1, C_2, \dots, C_\lambda\}$  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  $\lambda$  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 4.4 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 4.5.1 e 4.5.2. Quando o processo verifica todas as caixas de  $S$  e seus respectivos EM, um conjunto de EM candidatos  $L$  é obtido com suas respectivas avaliações. Então, o item é empacotado no melhor EM candidato através do Algoritmo 4.1 da Seção 4.4. Porém, se  $L = \emptyset$ , ou seja, 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  $\lambda$  deve ser incrementado em 1. Para o caso com rotação, após a seleção do melhor EM, a orientação do item é escolhida aleatoriamente dentre as possíveis orientações que este pode assumir para o melhor EM.

O Algoritmo 4.2 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 4.5.1 e 4.5.2, e *Empacota* é o Algoritmo 4.1 da Seção 4.4.

---

**Algoritmo 4.2** Heurística construtiva de empacotamento

---

```

1: procedimento CONSTRUTIVA(Permutação:  $P$ ; Itens:  $I$ ; Dimensões:  $(W, H, D)$ )
2:    $S \leftarrow \emptyset$                                 ▷ Solução construída começa vazia
3:    $\lambda \leftarrow 0$                               ▷ Número de caixas abertas começa com 0
4:   para cada  $i \in P$  faça                          ▷ Para cada item na ordem da permutação
5:      $a_i \leftarrow i$ -ésimo item de  $I$                 ▷ Item  $a_i$  será empacotado
6:      $L \leftarrow \emptyset$                           ▷ Lista de EM candidatos
7:     para cada  $C \in S$  faça                          ▷ Para cada caixa aberta
8:       se VAZIO( $C$ )  $\geq$  ESPAÇO( $a_i$ ) então          ▷ Se o espaço vazio de  $C$  pode conter  $a_i$ 
9:         para cada EM  $\in C$  faça
10:          se VERIFICA( $a_i$ , EM) então                ▷ Se o EM pode conter  $a_i$ 
11:             $L \leftarrow \{EM, C, f(EM)\}$           ▷ EM de  $C$  é válido com valor  $f(EM)$ 
12:          fim se
13:        fim para
14:      fim se
15:    fim para
16:    se  $L \neq \emptyset$  então                          ▷ Se existe EM candidato
17:      EM  $\leftarrow$  Melhor candidato em  $L$             ▷ Melhor EM de acordo com  $f$ 
18:       $C \leftarrow$  Caixa do melhor candidato          ▷ Referência para o empacotamento
19:    senão                                              ▷ Não há candidatos em  $L$ 
20:       $C \leftarrow$  CAIXA( $W, H, D$ )                    ▷ Uma nova caixa é aberta
21:       $S \leftarrow C$                                   ▷ Adiciona a nova caixa em  $S$ 
22:       $\lambda \leftarrow \lambda + 1$                     ▷ Atualiza o número de caixas abertas
23:      EM  $\leftarrow$  ESPAÇO( $C, W, H, D$ )                ▷ EM recebe o único EM da nova caixa
24:    fim se
25:    ORIENTA( $a_i$ , EM)  ▷ Se há rotação, escolhe uma orientação aleatória para  $a_i$ .
26:    EMPACOTA( $a_i, C, EM$ )  ▷ Empacota  $a_i$  através do Algoritmo 4.1
27:  fim para
28:  retorna  $S, \lambda$                                 ▷ Retorna a solução  $S$  e o número de caixas abertas
29: fim procedimento

```

---

### 4.5.1 Best Fit e Best EM

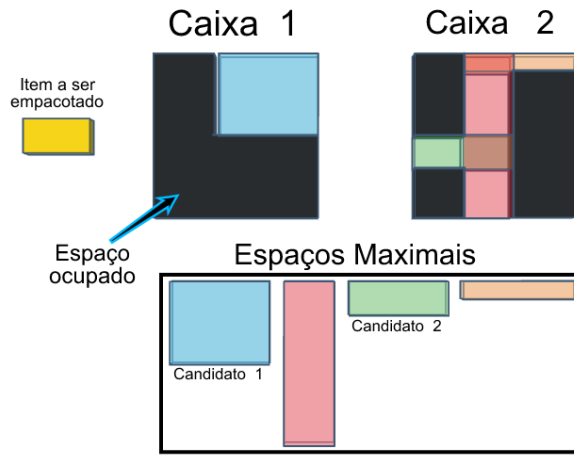
O **Best Fit** e o **Best EM** são critérios utilizados para avaliar o potencial de um EM candidato para receber o item sendo empacotado na iteração corrente da heurística construtiva. Além disso, quando um EM é construído através do processo de corte ou atuali-

zação apresentado na Seção 4.4, 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).

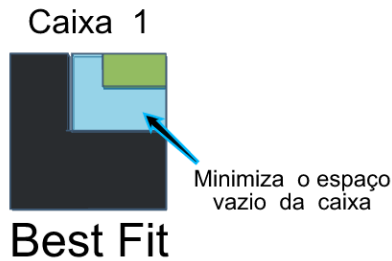
O Best Fit avalia o EM de acordo com a ocupação de sua respectiva caixa. Isto é, o valor do EM é baseado na quantidade de volume ocupado da caixa. Esta estratégia depende dos itens que já estão empacotados. O melhor EM deve ser aquele que está associado à caixa com maior volume ocupado, ou seja, o item será empacotado na caixa que resultará na minimização de espaço vazio da respectiva caixa. 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 será aquele que minimiza o volume total dos novos EM formados pelo procedimento de corte. Ou seja, o melhor EM 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 4.5 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 minimiza o volume do espaço vazio após o empacotamento. 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.

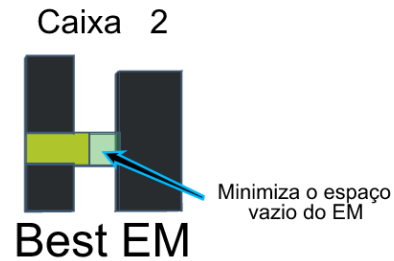
Como especificado na Seção 4.3, o CI é o canto onde o item será posicionado quando empacotado. Desta maneira, quando uma heurística ou meta-heurística utiliza um destes dois critérios, o CI deve ser selecionado de maneira a minimizar a distância entre o canto do item e o respectivo canto da caixa. Ou seja, o CI deve ser o mais próximo do respectivo canto da caixa. Caso exista empate, então um dos cantos candidatos deve ser escolhido aleatoriamente. A Figura 4.6 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. Portanto, 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 checka que há um empate entre dois cantos: o superior esquerdo e o inferior esquerdo. Portanto, 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 AM é realizar o empacotamento dos itens começando pelos cantos, em seguida preenchendo os lados e finalizando no centro da caixa.



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



(b) EM selecionado pelo Best Fit.



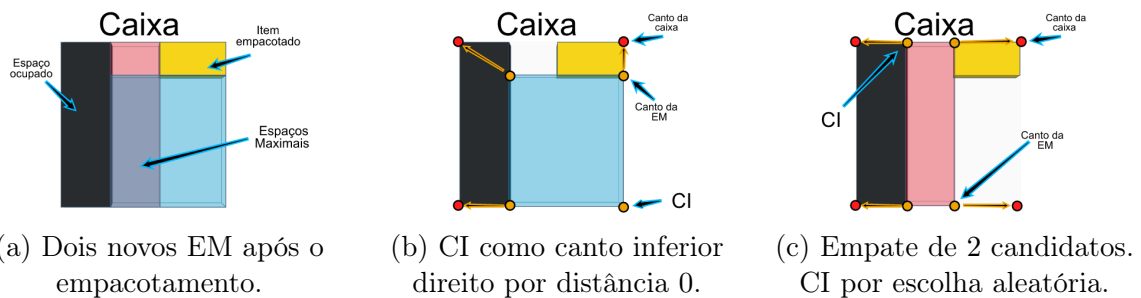
(c) EM selecionado pelo Best EM.

Figura 4.5: Exemplo bidimensional do critério de escolha Best Fit e Best EM.

### 4.5.2 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 momento 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



(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.

Figura 4.6: Exemplo bidimensional da seleção do CI de dois novos EM com o Best Fit ou Best EM.

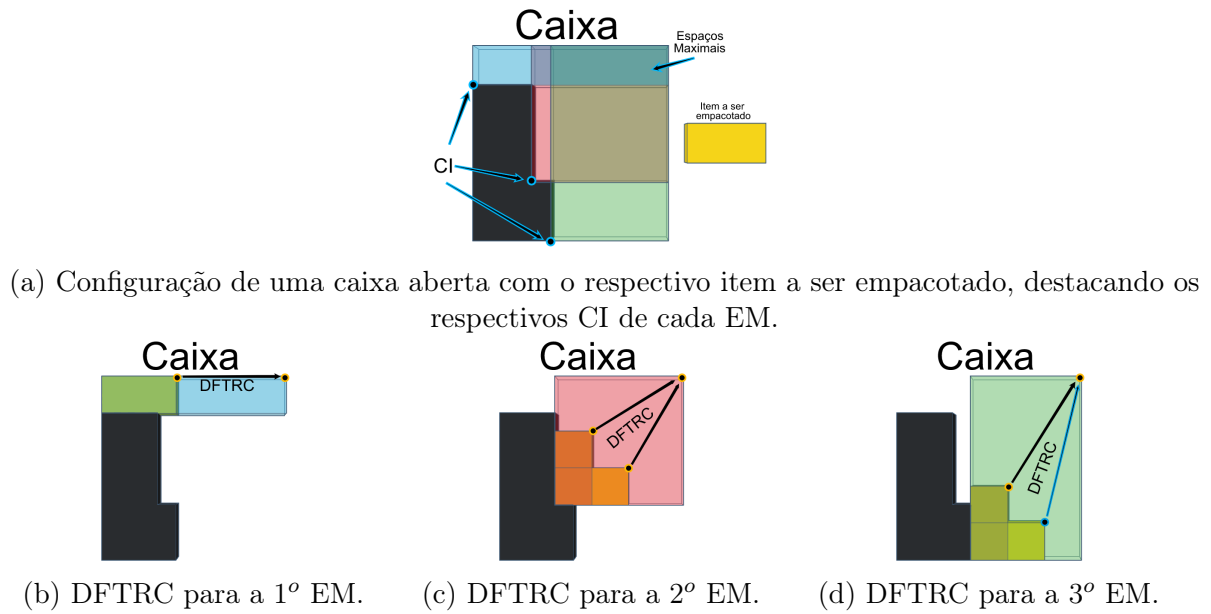


Figura 4.7: Exemplo bidimensional com rotação do critério de escolha DFTRC.

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. Isto é, 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 4.7 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.

O DFTRC também é capaz de selecionar uma orientação para o item, escolhendo a orientação que maximiza o critério entre as possíveis orientações do próprio. Um exemplo pode ser visto na Figura 4.7d, 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 5.1 deste trabalho. O critério de seleção utiliza a melhor orientação do item para 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 4.1, é escolhida



aleatoriamente após a seleção do melhor EM assim como detalhado no Algoritmo 4.2.

### 4.5.3 Complexidade

Esta seção apresenta a complexidade de tempo, determinada através da avaliação da quantidade de operações primitivas, da heurística construtiva detalhada pelo Algoritmo 4.2.

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 4 do Algoritmo 4.2 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 7 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. Ou seja, no pior caso temos uma caixa aberta a cada iteração, e a cláusula de repetição 7 deve percorrer até  $n$  caixas.

O conjunto de passos presente nas linhas 8 até 14 consiste de uma verificação para avaliar se na caixa há uma localização válida para o item. A cláusula condicional da linha 8 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 a dimensão dos itens e da caixa são inteiros positivos. A cláusula de repetição da linha 9 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  $\Theta(1)$ . Portanto, a verificação da linha 10 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 4.4).

Caso o EM seja um candidato válido para o empacotamento, o passo 11 efetua a inserção deste EM na lista auxiliar  $L$  de candidatos, que pode ser implementada como uma lista de prioridade *heap*. A inserção em uma estrutura de dados *heap* tem complexidade logarítmica em relação ao tamanho da entrada. Neste caso, a complexidade para a inserção em  $L$  seria  $\mathcal{O}(\log m)$ , pois  $m$  é um valor que também majora o número de possíveis candidatos para empacotar o item. Além disso, o passo da linha 11 também utiliza um dos critérios de avaliação detalhados nas Seções 4.5.2 e 4.5.1. 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 se trata de 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). 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 em comparações entre números inteiros positivos (6 para o 3BP e 2 para o 2BP). Desta maneira, a possibilidade de rotação não afeta a complexidade do DFTRC.

Os passos 16 até 24 armazenam os parâmetros de entrada para o Algoritmo 4.1 presente no passo 26. A seleção do melhor EM candidato da linha 17 tem complexidade  $\Theta(1)$  devido ao acesso na *heap*  $L$ . O método *Orienta* da linha 25 escolhe aleatoriamente uma orientação dentre as possíveis para o item baseado no melhor EM. Portanto, pode ser implementado com um número constante de operações e comparações devido a 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 26, é detalhada na Seção 4.4.1, sendo  $\mathcal{O}(m^2)$ .

Então, a complexidade de tempo do Algoritmo 4.2 é  $\mathcal{O}(n \times \text{máximo}(n \times m \log m, m^2))$ . O termo  $n \times m \log m$  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 4.2 sumariza a complexidade de tempo dos passos descritos nos parágrafos anteriores que compõem o Algoritmo 4.2 em conjunto com a própria complexidade final para a grande maioria das instâncias. O restante dos passos não mencionados, inclusive os passos 2 e 3, têm complexidade constante.

Tabela 4.2: Sumário da complexidade de tempo do Algoritmo 4.2.

Instrução	Passo	Complexidade	Observação
1º <i>Loop</i>	4	$\mathcal{O}(n)$	Utilizando um vetor de acesso direto como auxiliar para o armazenamento dos EM de uma caixa aberta e um <i>heap</i> para $L$ .
2º <i>Loop</i>	7	$\mathcal{O}(n)$	
1º Condição	8	$\Theta(1)$	
3º <i>Loop</i>	9	$\mathcal{O}(m)$	
2º Condição	10	$\Theta(1)$	
Inserção em $L$	11	$\mathcal{O}(\log m)$	
Best Fit ou EM/DFTRC	11	$\Theta(1)$	
Melhor elemento de $L$	17	$\Theta(1)$	
Orienta	25	$\Theta(1)$	
Empacota	26	$\mathcal{O}(m^2)$	
Algoritmo 4.2	Final	$\mathcal{O}(n \times m^2)^*$	

\*Complexidade para a grande maioria das instâncias.

## 4.6 Busca Local

A heurística de empacotamento apresentada na Seção 4.5 pode ser utilizada na fase de construção de meta-heurísticas para a resolução do 3BP ou do 2BP. Porém, algumas meta-heurísticas, como o GRASP e a Busca Tabu, necessitam de mecanismos para a busca de soluções viáveis no espaço de soluções do problema, a fim de sintetizar uma fase de melhora e encontrar ótimos locais. Esta seção apresenta uma heurística que pode ser utilizada na fase de melhora, recebendo uma solução já construída como parâmetro de entrada. O foco principal é propor um método de busca local, dentre os introduzidos na Seção 2.2.2, para servir como um mecanismo que é 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 movê-lo para um novo canto de inserção. A própria heurística construtiva proposta na Seção 4.5 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 bom 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. 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 4.5 para realizar o reempacotamento dos itens. Para isso, um novo parâmetro de entrada deve ser introduzido no Algoritmo 4.5, 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 nos passos 1, 2 e 3: a solução parcial  $S$  seria introduzida como parâmetro de entrada na linha 1, o passo 2 seria removido e o passo 3 seria carregar a variável  $\lambda$  com o número de caixas abertas de  $S$ . Assim, a permutação  $P$  de entrada agora representa a ordem de empacotamento dos  $\hat{n} \leq n$  itens presentes no conjunto  $I$  de entrada. 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  $P$  utilizada.

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 4.2, 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 a qualidade de diferentes

soluções.

O Algoritmo 4.3 detalha o procedimento de busca local proposto com *Best Improvement*, tal que a função *Reempacota* é a heurística construtiva presente no Algoritmo 4.2 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 apresentado nas Seções 4.5.1 e 4.5.2. 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 utilizada são as soluções geradas com sucesso pelo método *Reempacota*.

---

**Algoritmo 4.3** 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.

```

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

```

---

### 4.6.1 Complexidade

Esta seção apresenta a complexidade de tempo do Algoritmo 4.3, que realiza uma busca local através do reempacotamento de itens com a heurística construtiva detalhada no Algoritmo 4.2.

O passo 3 consiste de uma cláusula de repetição que reinicializa o processo de busca até encontrar um ótimo local. A análise do número de repetições máxima não corresponde ao comportamento prático deste algoritmo. 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 com uma única caixa aberta. Desta maneira, no pior caso, o passo 3 realizaria  $n - 1$  repetições. Este é um caso especial que raramente acontece na prática. Além disso, as iterações da busca local, que são efetuadas dentro da cláusula de repetição do passo 3, ficam cada vez mais rápidas de serem executadas, uma vez que melhores soluções são encontradas e um número maior de desistências no reempacotamento acontece. Portanto, esta análise utilizará a complexidade média para o passo 3, a fim de melhor corresponder ao comportamento prático do algoritmo.

As soluções iniciais utilizadas são geradas pela heurística construtiva proposta na Seção 4.5, 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. Desta maneira, 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. Então, no caso médio, o número de repetições do passo 3 é pequeno quando comparado com a magnitude do parâmetro de entrada  $n$ , e as iterações exigem cada vez menos tempo de processamento para averiguar se há melhora. Desta maneira, este trabalho considera que a complexidade média do passo 3 é  $\Theta(1)$ .

Os passos 2, 4, 16 e 25 efetuam atribuições com variáveis cujo tamanho 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 4.1. 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 consistem de variáveis positivas inteiras. Como uma solução viável apresenta todos os itens da instância, o conjunto total de caixas abertas possuem os  $n$  itens empacotados. Então a complexidade da cópia é independente do número de caixas da solução. Portanto, 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. Desta maneira, a complexidade final da atribuição presente nos passos 2, 4, 16 e 25 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 das atribuições efetuadas nos passos 2, 4, 16 e 25.

As cláusulas de repetição presentes nos passos 6 e 11 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 6 e 11 efetuam no máximo  $n$  repetições.

O passo 17 deve esvaziar todos os itens de duas caixas da solução atual. No pior caso para esta função, a solução atual  $s$  pode ter duas caixas abertas, com todos os itens presentes. Assim, a função deverá iterar sobre todos os itens das instâncias para sintetizar o conjunto  $I$ . 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. Portanto, o passo da linha 17 realiza no máximo  $n$  iterações. O passo 18 realiza repetições baseadas no número de itens do conjunto  $I$ . Analogamente, este conjunto pode ter no máximo  $n$  itens. Então, o número de repetições máximo do passo 18 também é  $n$ .

O passo 21 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, a complexidade da remoção e inserção de um elemento no final é  $\Theta(1)$ . Desta maneira, as complexidades previamente apresentadas nas Seções 4.4.1 , 4.5.3 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 21 pode ser implementado com complexidade constante.

Finalmente, a função *Reempacota*, utilizada no passo 22, é uma reimplementação da heurística construtiva detalhada na Seção 4.2 pelo Algoritmo 4.2 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  $n$  no pior caso. Assim, 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 da busca local proposta no Algoritmo 2.5 é dada pelos passos 3, 6, 11 e 22, sendo  $\mathcal{O}(n^3 \times m^2)$ . O passo 4 não afeta a complexidade final devido à clausula de repetição presente no passo 6. Assim como os passos 16, 17, 18 e 21 não afetam a complexidade final devido à complexidade do método *Reempacota* presente no passo 22. Nos casos especiais detalhados na Seção 4.5.3, a heurística de empacotamento apresenta uma complexidade no pior caso diferente. Nestes casos especiais, a complexidade da busca local é  $\mathcal{O}(n^4 \times m \log m)$ . A análise de complexidade desta seção é sumarizada pela Tabela 4.3. A tabela relaciona a instrução com o passo e sua complexidade, apresentando também a complexidade final do Algoritmo 4.3. 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 6 e 11, ou seja, na iteração em que o método esvazia as caixas  $i = \lambda - 1$  e  $j = \lambda$ .

Tabela 4.3: Sumário da complexidade de tempo do Algoritmo 4.3.

Instrução	Passo	Complexidade	Observação
Cópia da solução	2, 4, 16 e 25	$\mathcal{O}(m)^*$	Utilizando uma lista duplamente encadeada como auxiliar para o armazenamento das caixas abertas presente em uma solução.
<i>Loop</i> externo	3	$\Theta(1)^{**}$	
1º <i>Loop</i>	6	$\mathcal{O}(n)$	
2º <i>Loop</i>	11	$\mathcal{O}(n)$	
Construção de $I$	17	$\mathcal{O}(n)$	
3º <i>Loop</i>	18	$\mathcal{O}(n)$	
Remoção de caixas	21	$\Theta(1)$	
Reempacota	22	$\mathcal{O}(n \times m^2)^*$	
Algoritmo 4.3	Final	$\mathcal{O}(n^3 \times m^2)^*$	

\*Complexidade para a grande maioria das instâncias.

\*\*Complexidade do caso médio.



## 4.7 Variable Neighborhood Descent

Como mencionado na Seção 4.6, algumas 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. O processo de busca pode ficar atrelado a um determinado local ou direção no espaço de soluções impossibilitando alguns ótimos locais de serem encontrados, que por sua vez podem ser um ótimo global para o problema. Portanto, 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 4.6.

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. Portanto, 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. Porém, 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 4.6 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 2.6 introduzido na Seção 2.2.3, que detalha toda a fundamentação teórica da heurística VND. Um total de quatro vizinhanças  $V_k$  são utilizadas, onde  $V_1$  é composto pelas soluções geradas com sucesso no passo 22 do Algoritmo 4.3, que detalha a busca local proposta neste trabalho. Portanto, a busca local da Seção 4.6 é utilizada 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 4.6: uma pequena modificação na heurística construtiva, presente no Algoritmo 4.2 e detalhada na Seção 4.5, 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 4.2.

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 4.2. Analogamente, o único vizinho contido em  $V_3$  é obtido da mesma maneira, porém 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 a função objetivo ser o FOT ou FOA. Assim como detalhado na Seção 4.2, 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, pois isto significa que a solução que será construída neste reempacotamento já é equivalente ou pior que a atual. Porém, se o FOA for utilizado, duas soluções com o mesmo número de caixas abertas podem ter avaliações diferentes. Aqui, 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 4.4 detalha o processo das vizinhanças  $V_2$ ,  $V_3$  e  $V_4$ . A Tabela 4.4 sumariza as quatro vizinhanças, relacionando-as com o método utilizado. Assim, o VND proposto utiliza essas vizinhanças e executa de acordo com o Algoritmo 2.6 apresentado no Capítulo 2.

Tabela 4.4: Relação entre a vizinhança  $V_k$  e seu método no VND proposto.

Vizinhança	Método
$V_1$	Busca Local da Seção 4.6 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

**Algoritmo 4.4** Vinhança para  $k \in \{2, 3, 4\}$ **Requerimento:** Solução  $s$  deve ter pelo menos 2 caixas abertas.

```

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

```

**4.7.1 Complexidade**

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 2.6, 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 4 do VND possui o mesmo problema que o passo 3 da busca local detalhada pelo Algoritmo 2.5: a análise da quantidade máxima de repetições deste passo não corresponde ao comportamento prático do algoritmo. O número máximo de repetições acontece em um caso especial análogo ao detalhado na Seção 4.6.1 na suposição de que uma solução trivial de má qualidade seja passada e seja possível responder a 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 raramente deve acontecer fora do âmbito de testes controlados. O objetivo desta seção é determinar a complexidade de tempo de cada busca local que utiliza as vizinhanças relacionadas com os métodos da Tabela 4.4. A primeira busca local é a própria heurística proposta na Seção 4.6 detalhada no Algoritmo 4.6, e as três últimas vizinhanças são

detalhadas no Algoritmo 4.4 compostas por uma única solução.

A complexidade da primeira busca local é detalhada na Seção 4.6.1, 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 dos três últimos métodos de busca local é dada diretamente pela análise do pior caso do Algoritmo 4.4, pois trata-se de vizinhanças que geram uma solução.

Os passos 2 e 13 de cada vizinhança são atribuições entre variáveis cujo tamanho depende da instância. Como descrito na Seção 4.6.1, a complexidade da cópia de uma solução com a representação computacional detalhada na Seção 4.1 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 2 e 13 é  $\mathcal{O}(m)$ .

O passo 3 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* utilizando o último elemento como pivô, a lista duplamente encadeada pode ser ordenada com  $\mathcal{O}(n \log n)$  instruções. Assim, a complexidade do passo 3 é  $\mathcal{O}(n \log n)$ . Para  $V_4$ , o passo 3 algoritmo realiza um embaralhamento das caixas abertas, que pode ser implementado por uma ou mais completas iterações sequenciais em toda a lista, ou seja, por um método  $\mathcal{O}(n)$ .

O passo 4 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 4 tem complexidade  $\mathcal{O}(n)$ .

O passo 5 armazena  $q$  itens presentes nas caixas selecionadas dentro do conjunto  $I$  e realiza  $q$  iterações. O valor  $q$  é menor que  $n$ , porém, no pior caso, este valor é próximo de  $n$ . Desta maneira, o passo 5 tem complexidade  $\mathcal{O}(n)$ . O passo 6 realiza uma iteração em cada item  $a_i \in I$  para obter a sua respectiva ordem de empacotamento da solução original. Analogamente, o número de itens presente é majorado por  $n$ . Portanto, o passo 6 também tem complexidade  $\mathcal{O}(n)$ .

O passo 9 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 relacionadas em  $C$  é no máximo  $\frac{n}{2}$ . Então, o passo 9 é um método que realiza até  $\frac{n}{2}$  repetições. Como descrito na Seção 4.6.1, 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 9 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 10, é a reimplementação da heurística construtiva presente no Algoritmo 4.2 para modificar seus parâmetros de entrada.

Esta modificação é detalhada na Seção 4.6. A reimplementação do método tem a mesma complexidade da heurística construtiva, que por sua vez é analisada na Seção 4.5.3. Portanto, a complexidade do passo 10 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 4.5 sumariza a complexidade de cada passo do Algoritmo 4.4 e apresenta a complexidade final das vizinhanças  $V_k, k \in \{1, 2, 3, 4\}$ .

Tabela 4.5: Sumário da complexidade de tempo de cada passo do Algoritmo 4.4 em conjunto com a complexidade final das vizinhanças  $V_1, V_2, V_3$  e  $V_4$ .

Instrução	Passo	Complexidade	Observação
Cópia da solução	2 e 13	$\mathcal{O}(m)^*$	Utilizando uma lista duplamente encadeada como auxiliar para o armazenamento das caixas abertas presente em uma solução.
Ordenação/Embaralhamento	3	$\mathcal{O}(n \log n) / \mathcal{O}(n)$	
Construção de $C$	4	$\mathcal{O}(n)$	
Construção de $I$	5	$\mathcal{O}(n)$	
Construção de $P$	6	$\mathcal{O}(n)$	
Remoção de caixas	9	$\mathcal{O}(n)$	
Reempacota	10	$\mathcal{O}(n \times m^2)^*$	
$V_1$ (Algoritmo 4.3)	Final	$\mathcal{O}(n^3 \times m^2)^*$	
$V_2, V_3$ e $V_4$ (Algoritmo 4.4)	Final	$\mathcal{O}(n \times m^2)^*$	

\*Complexidade para a grande maioria das instâncias.

## 4.8 Greedy Randomized Adaptative Search Procedure

A heurística construtiva proposta na Seção 4.5 é capaz de resolver os problemas de empacotamento. Porém, 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 boa alternativa para a resolução do 3BP e 2BP 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 Adaptative Search Procedure*) proposto no trabalho (ZUDIO et al., 2017a), utilizando como base a fundamentação teórica da Seção 2.3.1.

### 4.8.1 Fase de construção

A fase de construção do GRASP proposto é composta pela heurística adaptativa gulosa detalhada no Algoritmo 4.2. O algoritmo deve ser modificado, introduzindo um parâmetro de entrada adicional: o valor  $\alpha$ , com o objetivo de diversificar as soluções iniciais obtidas a cada iteração e escapar de ótimos locais. A heurística construtiva modificada torna-se um algoritmo semi-guloso. Os EM candidatos permanecem armazenados na lista

de candidatos  $L$ , mas a seleção de um candidato não é mais determinística: ao invés de selecionar o melhor candidato, o semi-guloso deve selecionar um dos  $\alpha\%$  melhores candidatos aleatoriamente. Os  $\alpha\%$  melhores EM candidatos compõem o RCL (*Restricted Candidate List*). O Algoritmo 4.5 detalha a fase de construção do GRASP, com a modificação no parâmetro de entrada e na seleção do Algoritmo 4.2. O critério de escolha para determinar a qualidade de cada EM pode ser o Best Fit, Best EM ou DFTRC detalhados nas Seções 4.5.1 e 4.5.2.

---

**Algoritmo 4.5** Fase de construção GRASP para o 3BP e o 2BP.

---

**Requerimento:** Parâmetro  $\alpha$  no intervalo  $[0, 1]$

```

1: procedimento SEMI-GULOSO(Permutação:  $P$ ; Itens:  $I$ ; Dimensões:  $(W, H, D)$ ;
   Número Real:  $\alpha$ )
2:    $S \leftarrow \emptyset$                                 ▷ Solução construída começa vazia
3:    $\lambda \leftarrow 0$                                 ▷ Número de caixas abertas começa com 0
4:   para cada  $i \in P$  faça                                ▷ Para cada item na ordem da permutação
5:      $a_i \leftarrow i$ -ésimo item de  $I$                                 ▷ Item  $a_i$  será empacotado
6:      $L \leftarrow \emptyset$                                 ▷ Lista de EM candidatos
7:     para cada  $C \in S$  faça                                ▷ Para cada caixa aberta
8:       se  $\text{VAZIO}(C) \geq \text{ESPAÇO}(a_i)$  então                                ▷ Se o espaço vazio de  $C$  pode conter  $a_i$ 
9:         para cada  $EM \in C$  faça
10:          se  $\text{VERIFICA}(a_i, EM)$  então                                ▷ Se o EM pode conter  $a_i$ 
11:             $L \leftarrow \{EM, C, f(EM)\}$                                 ▷ EM de  $C$  é válida com valor  $f(EM)$ 
12:          fim se
13:        fim para
14:      fim se
15:    fim para
16:    se  $L \neq \emptyset$  então                                ▷ Se existe EM candidato
17:       $EM \leftarrow$  Um dos  $100\alpha\%$  melhores de  $L$                                 ▷ Um candidato aleatório da RCL
18:       $C \leftarrow$  Caixa do candidato                                ▷ Referência para o empacotamento
19:    senão                                ▷ Não há candidatos em  $L$ 
20:       $C \leftarrow \text{CAIXA}(W, H, D)$                                 ▷ Uma nova caixa é aberta
21:       $S \leftarrow C$                                 ▷ Adiciona a nova caixa em  $S$ 
22:       $\lambda \leftarrow \lambda + 1$                                 ▷ Atualiza o número de caixas abertas
23:       $EM \leftarrow \text{ESPAÇO}(C, W, H, D)$                                 ▷ EM recebe o único EM da nova caixa
24:    fim se
25:     $\text{ORIENTA}(a_i, EM)$                                 ▷ Se há rotação, escolhe uma orientação aleatória para  $a_i$ .
26:     $\text{EMPACOTA}(a_i, C, EM)$                                 ▷ Empacota  $a_i$  através do Algoritmo 4.1
27:  fim para
28:  retorna  $S, \lambda$                                 ▷ Retorna a solução  $S$  e o número de caixas abertas
29: fim procedimento

```

---

## 4.8.2 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 algo-

ritmo proposto, podemos seguir a ideia original do GRASP sintetizando uma heurística chamada de **GRASP Puro**. O GRASP Puro é o próprio Algoritmo 2.8 utilizando a busca local proposta na Seção 4.6 para os problemas de empacotamento. Desta maneira, a fase de melhora do GRASP Puro é o Algoritmo 4.3, onde a busca local pode ser implementada com o First Improvement ou Best Improvement. Como mencionado na Seção 4.7, 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  $\alpha$  pequeno. Para contornar isso, um segundo método foi desenvolvido, chamado *GRASP/VND*, utilizando o VND proposto na Seção 4.7.

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 4.4 na fase de melhora. Este segundo método aplica os mecanismos detalhados na Seção 4.7 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, a fim 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 de uma meta-heurística 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).

### 4.8.3 Complexidade

Esta seção estuda o comportamento prático do algoritmo GRASP proposto através da análise de sua complexidade. A complexidade depende do critério de parada utilizado, que por sua vez é, geralmente, determinado tempo de execução. Esta seção também analisa a complexidade dos componentes que sintetizam cada GRASP proposto, a fim de detalhar o comportamento prático de uma iteração de cada método. A análise é feita

através da contagem de instruções básicas no pior caso.

O primeiro componente é uma modificação na heurística construtiva descrita na Seção 4.5 e detalhada no Algoritmo 4.5. Este componente compõe a fase de construção de todas as versões propostas do GRASP para os problemas de empacotamento. O Algoritmo 4.5 tem a mesma complexidade do Algoritmo 4.2, que por sua vez é sumarizada pela Tabela 4.2. A modificação altera somente a complexidade do passo 17: no pior caso, o candidato selecionado exige a remoção de até  $m$  candidatos de  $L$ , que pode ser implementado como um *Heap*. Portanto, a complexidade do passo 17 passa a ser  $\mathcal{O}(m)$  ao invés de  $\Theta(1)$  como no algoritmo original, onde  $m$  é o valor que majora o número de EM presentes em qualquer caixa aberta durante toda a execução da meta-heurística. Como a complexidade dos passos 7 até 15 e 26 é superior ao passo 17, então, a complexidade final permanece a mesma para qualquer instância. A Tabela 4.2 mostra que, para a maior parte das instâncias, a complexidade desta fase de construção é  $\mathcal{O}(n \times m^2)$ .

A fase de melhora é composta pela busca local ou o VND proposto nas Seções 4.6 e 4.7 respectivamente. A complexidade da busca local é detalhada na Seção 4.6.1 e sumarizada pela Tabela 4.3. A tabela mostra que a complexidade final da busca local para a grande maioria das instâncias é  $\mathcal{O}(n^3 \times m^2)$ . O comportamento prático e a complexidade do VND é detalhado na Seção 4.7.1, onde a Tabela 4.5 sumariza a complexidade de cada vizinhança proposta. Desta maneira, a complexidade de uma iteração GRASP é dada por esta fase, pois a complexidade da busca local e do VND supera a complexidade da heurística construtiva. A Tabela 4.6 sumariza a complexidade de cada componente que compõe as 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.

Tabela 4.6: Sumário da complexidade dos componentes de cada GRASP proposto em conjunto com a complexidade final de uma iteração.

Componente	Fase	Complexidade
Heurística construtiva (Algoritmo 4.1)	Construção	$\mathcal{O}(n \times m^2)^*$
Busca Local (Algoritmo 4.3)	Melhora	$\mathcal{O}(n^3 \times m^2)^*$
VND (Algoritmo 4.7)	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.

## 4.9 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** é uma meta-heurística



que não utiliza um mecanismo de recomeço, e sim constrói uma solução para melhorá-la durante suas iterações até o critério de parada ser atingido. As iterações da busca tabu aplicam um mecanismo para escapar de ótimos locais independente de qualquer gerador de números pseudoaleatórios, isto é, é uma estratégia determinística. 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, a busca tabu proposta 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 busca tabu 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 2.3.2.

A busca tabu proposta, chamada BT, para os problemas de empacotamento segue a mesma ideia da busca local detalhada no Algoritmo 4.6, 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 4.5 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 4.5.1 e 4.5.2. O segundo parâmetro de entrada na busca tabu original, detalhada no Algoritmo 2.9, 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 4.6: o Algoritmo 4.2 é parcialmente modificado para receber uma solução que já possui caixas abertas e um subconjunto 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 a busca tabu 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  $\lambda_{max}$  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  $\lambda_{max}$ .

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 utiliza 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, então ela é removida da lista e selecionada como final da iteração atual. Assim, 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 4.6 detalha a busca tabu proposta, 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 4.6. Como descrito na Seção 4.7, 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 4.7 e propostas no VND apresentado na Seção 4.7, a fim de intensificar e diversificar o processo de busca. O método utiliza cada uma das três vizinhanças no início da iteração atual antes de começar o processo de esvaziar caixas par a par, ou seja,  $V_2, V_3$  e  $V_4$  são executadas antes de  $V_1$ . Desta maneira, cada iteração realiza três reempacotamentos 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

---

**Algoritmo 4.6** Busca Tabu BT para o 3BP e o 2BP
 

---

**Observação:** a estrutura de vinhança  $V$  constitui-se das soluções geradas pelo método *Reempacota*, portanto, não é passada como parâmetro de entrada.

```

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

```

---

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 4.7 detalha o BT/VN de acordo com as modificações descritas.

### 4.9.1 Complexidade

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 dependem do critério de parada utilizado. Geralmente, este critério é atingir um determinado tempo de execução. Esta seção descreve a complexidade de tempo dos componentes que sintetizam os Algoritmos 4.6 e 4.7, 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 heurística construtiva da Seção 4.5. Desta maneira, o Algoritmo 4.2 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 4.5.3, 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 da meta-heurística em qualquer caixa aberta.

Seguindo o Algoritmo 4.6 como referência, os passos 2, 18, 28, 42 e 44 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 4.6.1, 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 8 é uma cláusula de repetição que itera nas caixas abertas da solução corrente assim como o passo 13. 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  $n$ . Desta maneira, a quantidade de repetições que ambos os passos realizam é majorado pelo parâmetro  $n$ . Portanto, os passos 8 e 13 têm complexidade  $\mathcal{O}(n)$ . As análises de complexidade anteriores assumem que a estrutura que armazena as caixas abertas de uma solução é uma lista duplamente encadeada. Os

---

**Algoritmo 4.7** Busca Tabu BT/VN para o 3BP e o 2BP

---

```

1: função BT-VN(Solução inicial:  $s_0$ ; Tamanho máximo do Tabu:  $T_{max}$ )
2:    $s^*, s \leftarrow s_0$                                 ▷ Melhor solução  $s^*$  e solução corrente  $s$  recebem solução inicial
3:    $T \leftarrow \emptyset$                                 ▷ Lista tabu começa vazia
4:   enquanto critério de parada não satisfeito faça
5:      $s_V^* \leftarrow \emptyset$                                 ▷ Melhor vizinho não tabu
6:      $\lambda_{max} \leftarrow \infty$                         ▷  $\lambda_{max}$  guarda a avaliação de  $s_V^*$ 
7:     para  $k \in \{2, 3, 4\}$  faça
8:        $s \leftarrow V_k(s)$                                 ▷ Aplica o Algoritmo 4.4 mantendo o melhor
9:     fim para
10:     $\lambda \leftarrow$  Número de caixas de  $s$                 ▷ Caixas abertas da solução atual
11:    para  $i$  de 1 até  $\lambda - 1$  faça
12:       $C \leftarrow$  Caixa  $C_i$  de  $s$                         ▷  $i$ -ésima caixa da solução atual
13:      se  $C$  estiver com 100% de carga então
14:        Avançar o loop                                ▷ Ignora caixas promissoras
15:      fim se
16:      para  $j$  de  $i + 1$  até  $\lambda$  faça
17:         $C' \leftarrow$  Caixa  $C_j$  de  $s$                     ▷  $j$ -ésima caixa da solução atual
18:        se  $C'$  estiver com 100% de carga então
19:          Avançar o loop                                ▷ Ignora caixas promissoras
20:        fim se
21:         $s' \leftarrow s$                                 ▷ Solução parcial recebe solução atual
22:         $I \leftarrow$  Todos os itens de  $C$  e  $C'$             ▷ Conjunto de itens para reempacotar
23:        para  $a_k \in I$  faça
24:           $P \leftarrow k$ -ésimo termo da permutação de  $s^*$   ▷ Mantém a ordem original
25:        fim para
26:        Apagar as caixas  $C_i$  e  $C_j$  da solução  $s'$ 
27:         $s' \leftarrow \text{REEMPACOTA}(s', P, I, (W, H, D), \lambda_{max})$   ▷ Retorna  $\emptyset$  se desistir
28:        se  $s' \neq \emptyset$  então
29:          se  $s' \notin T$  então                                ▷ Se  $s'$  não é uma solução tabu
30:            se  $s'$  é melhor que  $s_V^*$  então
31:               $s_V^* \leftarrow s'$                                 ▷ Novo melhor vizinho
32:               $\lambda_{max} \leftarrow f(s_V^*)$                 ▷ Avaliação de  $s_V^*$ .
33:            fim se
34:          senão
35:            Marca  $s'$  em  $T$                                 ▷ Sinaliza que este tabu foi encontrado nesta iteração
36:          fim se
37:        fim se
38:      fim para
39:    fim para
40:    se  $s_V^* = \emptyset$  então                                ▷ Se não há melhor vizinho
41:       $s_V^* \leftarrow$  Melhor tabu em  $T$  marcado nesta iteração  ▷ Critério de aspiração
42:       $s_V^* \leftarrow V_4(s_V^*)$                             ▷ Sempre aceita a solução gerada e nunca desiste
43:    fim se
44:    se  $s_V^*$  é melhor que  $s^*$  então
45:       $s^* \leftarrow s_V^*$                                 ▷ Nova melhor solução
46:    fim se
47:     $s \leftarrow s_V^*$                                 ▷ Solução corrente recebe o melhor vizinho não tabu
48:    ATUALIZA( $T, T_{max}, s_V^*$ )                            ▷ Atualiza o tabu de acordo com  $T_{max}$  e  $s_V^*$ 
49:  fim enquanto
50:  retorna  $s^*$                                 ▷ Retorna a melhor solução encontrada
51: fim função

```

---

passos 9 e 14 podem ser implementados com variáveis que recebem uma referência para as caixas abertas  $C_i$  e  $C_j$ . Desta maneira, a complexidade dos passos 9 e 14 é constante. Analogamente, a verificação feita nos passos 10 e 15 pode ser implementada também em tempo constante através de uma variável auxiliar que armazena a carga da caixa em sua representação computacional.

O passo 19 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 17 do Algoritmo 4.3, que tem sua complexidade sumarizada na Tabela 4.3. A complexidade para a construção de  $I$  é  $\mathcal{O}(n)$ . Desta maneira, o passo 20 é uma cláusula de repetição que realiza no máximo  $n$  iterações, pois este passo itera sobre os itens em  $I$ . O passo 23 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 23 também é constante. O passo 24 é uma reimplementação do Algoritmo 4.2 para modificar os parâmetros de entrada, a fim de realizar o reempacotamento com uma solução parcialmente construída e com um alvo máximo de qualidade para a desistência. Como descrito na Seção 4.6, esta modificação não altera a complexidade original da heurística construtiva, portanto, o passo 24 tem complexidade  $\mathcal{O}(n \times m^2)$  para a maioria das instâncias.

O passo 26 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 4.1. Este passo pode ser feito através de uma segunda cláusula de repetição sobre o vetor que armazena cada  $\lambda_i$ , 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 29 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 32 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 32 pode ser implementado em conjunto com a iteração realizada no passo 26. Portanto, este passo tem complexidade constante. O passo 38 é executado quando nenhum vizinho foi gerado

com sucesso, isto é, todos os vizinhos são tabu. O passo 38 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 32. Então, o passo 38 realiza no máximo  $T_{max}$  repetições. O passo 39 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 38. O passo 45 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 45 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 4.7, a maior parte dos passos são os mesmos que os realizados pelo BT, sendo de mesma complexidade. A principal modificação está presente nos passos 7 e 42. O passo 7 utiliza as vizinhanças  $V_2$ ,  $V_3$  e  $V_4$  em sequência, onde o Algoritmo 4.4 detalha o processo de cada uma delas. A complexidade destes métodos é detalhada na Seção 4.7.1 e resumida na Tabela 4.5, sendo todas  $\mathcal{O}(n \times m^2)$  para a maioria das instâncias. Finalmente, o passo 42 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 8, 13 e 24 para o BT e 11, 16 e 27 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 4.7 resume 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 4.7: Sumário da complexidade dos componentes de cada busca tabu proposta.

Instrução	Passo		Complexidade	Observação
	BT	BT/VN		
Solução inicial (Algoritmo 4.2)	Externo		$\mathcal{O}(n \times m^2)^*$	Utilizando uma lista duplamente encadeada como estrutura auxiliar para armazenar as soluções tabu.
Cópia da solução	2, 18, 28, 42 e 44	2, 21, 31, 45 e 47	$\mathcal{O}(m)^*$	
1º e 2º <i>Loop</i>	8 e 13	11 e 16	$\mathcal{O}(n)$	
Armazenar $C_i$ e $C_j$	9 e 14	12 e 17	$\Theta(1)$	
1º e 2º verificação	10 e 15	13 e 18	$\Theta(1)$	
Construir $I$	19	22	$\mathcal{O}(n)$	
3º <i>Loop</i>	20	23	$\mathcal{O}(n)$	
Apagar par de caixas	23	26	$\Theta(1)$	
Reempacota	24	27	$\mathcal{O}(n \times m^2)^*$	
Verificar tabu	26	29	$\mathcal{O}(T_{max} \times n^2)$	
Marcar solução	32	35	$\Theta(1)$	
Selecionar tabu	38	41	$\mathcal{O}(T_{max})$	
Remover tabu	39		$\Theta(1)$	
Atualizar $T$	45	48	$\Theta(1)$	
$V_2, V_3$ e $V_4$		7 e 42	$\mathcal{O}(n \times m^2)^*$	
1 iteração	Final	Final	$\mathcal{O}(n^3 \times m^2)^*$	

\*Complexidade para a grande maioria das instâncias.

## 4.10 Biased Random-Keys Genetic Algorithm

Como descrito na Seção 4.6, estratégias de busca local para o 3BP e o 2BP que envolvem a movimentação de itens, seja entre diferentes caixas ou dentro de si própria, 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 4.5. Uma alternativa é utilizar um processo que esvazia caixas, parcialmente 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 de uma meta-heurística. Por exemplo, ficar atrelada em um determinado ponto do espaço de soluções com um alto tempo de execução sem melhora.

Esta seção apresenta um método onde a 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 4.2. O procedimento independe de qualquer fase de melhora que necessita de algum tipo de busca local. O método proposto 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.

O algoritmo proposto, denominado BRKGA/VCD, é uma extensão do método apresentado no trabalho (ZUDIO et al., 2017b), 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 da meta-heurística BRKGA (*Biased Random-Keys Genetic Algorithm*) 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 2.3.3 e 2.3.4 para algoritmos evolucionários e o próprio BRKGA.

O BRKGA e a variação BRKGA/VCD para o 3BP e o 2BP é detalhada pelo Algoritmo 2.11, 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  $TOP$   $\epsilon$ : 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  $BOT$   $\omega$ : 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  $\rho$ : 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 *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 5 do algoritmo. Este processo permanece inalterado, isto é, a seleção consiste em copiar os  $100\epsilon$  indivíduos de  $P$  para  $P'$ , ou seja, manter os Elites na geração atual. 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 8, 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 depende 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 4.2 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 em uma permutação. Para exemplificar este processo, a Figura 4.8 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.

Como mencionado anteriormente, o empacotamento dos itens é realizado através de uma variante da heurística construtiva proposta na Seção 4.5 detalhada pelo Algoritmo 4.2. A finalidade desta variante é construir soluções utilizando um tempo menor de execução comparado à heurística construtiva original de empacotamento, 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 empacotar 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 de um item.

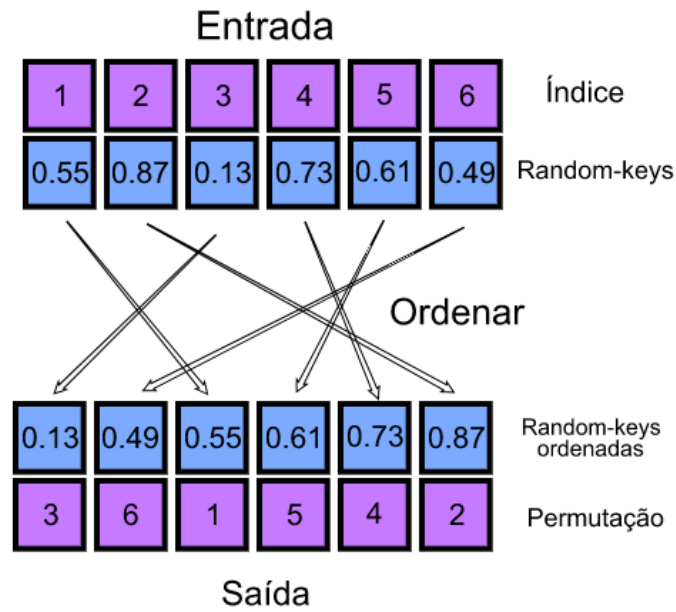


Figura 4.8: Exemplo do processo de ordenação efetuado pelo decodificador para  $n = 6$ .

Desta forma, a estratégia da heurística 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 variação 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  $A$  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. Ou seja, 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 4.2, 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.

$$A_i \in A \quad | \quad i = \{k \times r\} \quad (4.2)$$

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 heurística 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 4.2 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 4.8 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 4.9 apresenta o decodificador utilizado no BRKGA e no BRKGA/VCD, onde a função *Empacota* é o Algoritmo 4.8, podendo utilizar o Best Fit, Best EM ou o DFTRC como critério de avaliação para as EM.

#### 4.10.1 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 do BRKGA é sintetizar estratégias algorítmicas que são 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

---

**Algoritmo 4.8** 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.

```

1: procedimento CONSTRUTIVA(Permutação:  $P$ ; Itens:  $I$ ; Dimensões:  $(W, H, D)$ ;
   Lista de random-keys:  $L_1, L_2$ )
2:    $S \leftarrow \emptyset$                                 ▷ Solução construída começa vazia
3:    $\lambda \leftarrow 0$                                 ▷ Número de caixas abertas começa com 0
4:   para cada  $i \in P$  faça                                ▷ Para cada item na ordem da permutação
5:      $a_i \leftarrow i$ -ésimo item de  $I$                     ▷ Item  $a_i$  será empacotado
6:      $L \leftarrow \emptyset$                                 ▷ Lista de EM candidatos
7:     para cada  $C \in S$  faça                                ▷ Para cada caixa aberta
8:       se VAZIO( $C$ )  $\geq$  ESPAÇO( $a_i$ ) então    ▷ Se o espaço vazio de  $C$  pode conter  $a_i$ 
9:         para cada EM  $\in C$  faça
10:          se VERIFICA( $a_i$ , EM) então    ▷ Se o EM pode conter  $a_i$ 
11:             $L \leftarrow \{EM, C, f(EM)\}$     ▷ EM de  $C$  é válido com valor  $f(EM)$ 
12:          fim se
13:        fim para
14:        se  $L \neq \emptyset$  então    ▷ Se existe pelo menos um EM candidato
15:          Parar o loop                                ▷ Não tenta encontrar nas próximas caixas
16:        fim se
17:      fim se
18:    fim para
19:    se  $L \neq \emptyset$  então    ▷ Se existe EM candidato
20:      EM  $\leftarrow$  Melhor candidato em  $L$     ▷ Melhor EM de acordo com  $f$ 
21:       $C \leftarrow$  Caixa do melhor candidato    ▷ Referência para o empacotamento
22:      senão                                ▷ Não há candidatos em  $L$ 
23:         $C \leftarrow$  CAIXA( $W, H, D$ )    ▷ Uma nova caixa é aberta
24:         $S \leftarrow C$                                 ▷ Adiciona a nova caixa em  $S$ 
25:         $\lambda \leftarrow \lambda + 1$     ▷ Atualiza o número de caixas abertas
26:        EM  $\leftarrow$  ESPAÇO( $C, W, H, D$ )    ▷ EM recebe o único EM da nova caixa
27:      fim se
28:       $r_1 \leftarrow$  i-ésima random-key de  $L_1$     ▷ Random-key para o critério de escolha
29:       $r_2 \leftarrow$  i-ésima random-key de  $L_2$     ▷ Random-key para a orientação do item
30:      ORIENTA( $a_i$ , EM,  $r_2$ )    ▷ Se há rotação, determina a orientação de  $a_i$  através de  $r_2$ 
31:      EMPACOTA( $a_i, C$ , EM,  $r_1$ )    ▷ Empacota  $a_i$  através do Algoritmo 4.1 utilizando  $r_1$ 
   caso Best Fit ou Best EM seja o critério de escolha
32:    fim para
33:    retorna  $S, \lambda$     ▷ Retorna a solução  $S$  e o número de caixas abertas
34: fim procedimento

```

---

**Algoritmo 4.9** Decodificador utilizado no BRKGA
 

---

```

1: procedimento DECODIFICA(Indivíduo:  $i$ ; Itens:  $I$ ; Dimensões:  $(W, H, D)$ )
2:    $P \leftarrow$  ORDENA( $i$ )    ▷ Obtém a permutação através de ordenação de random-keys
3:    $L_1 \leftarrow$  random-keys referente ao CI    ▷ Caso o critério seja Best Fit ou Best EM
4:    $L_2 \leftarrow$  random-keys referente à orientação    ▷ Caso seja possível rotacionar
5:    $s \leftarrow$  EMPACOTA( $P, I, (W, H, D), L_1, L_2$ )    ▷ Empacotamento com o Algoritmo 4.8
6:   retorna  $s$ 
7: fim procedimento

```

---

com um método completamente aleatório. Desta maneira, 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 completamente aleatório.

No BRKGA original, a população inicial e os mutantes são indivíduos completamente 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 4.5, 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. Ou seja, 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. Portanto, 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. Desta maneira, 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 sequência 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 4.8 para empacotar os itens com cada permutação gerada como entrada. Assim, obtemos 4 ou 3 soluções com uma alta probabilidade de serem ótimas. Então, o procedimento deve codificar estas soluções em indivíduos chamados **Universais** através da Equação 4.3, 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 passados como parâmetro de entrada para a heurística

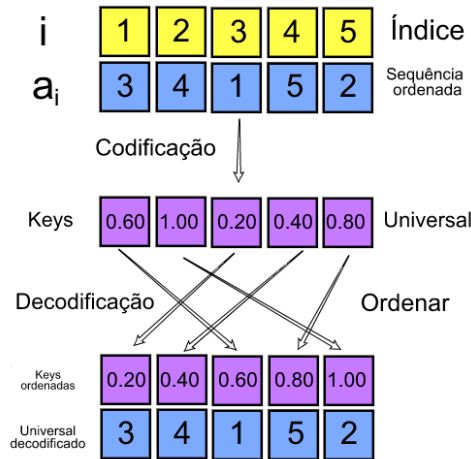


Figura 4.9: Exemplo do processo de codificação de uma permutação para  $n = 6$ .

construtiva em conjunto com a permutação ordenada. Finalmente, os universais devem ser mantidos em uma variável auxiliar que os armazena ordenados do melhor para o pior durante todo o processo da meta-heurística e devem ser introduzidos na população inicial.

$$r_i = p_i \times \frac{1}{n}, 1 \leq i \leq n \quad (4.3)$$

A Figura 4.9 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 4.3 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.

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 empacotados, 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 itens a serem empacotados. 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 método 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  $\omega$  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 4.9 e o parâmetro  $\rho$  que é utilizado na reprodução. A reprodução realizada durante o VCD é a mesma do BRKGA original, onde  $\rho$  é o mesmo parâmetro de entrada calibrável da meta-heurística cujo o objetivo é determinar a probabilidade de escolha Elitista, ou seja, a chance de escolher o gene do respectivo parente Elite ao invés do gene do parente não-Elite para o filho. 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  $\rho$  tanto na reprodução do processo iterativo do BRKGA quanto no VCD. Finalmente, o último parâmetro de entrada é um novo valor  $\alpha$  no intervalo  $]0, 1[$  que será detalhado a seguir e deve ser calibrado.

O VCD realiza seu processo iterativo até que a população de entrada atinja o tamanho do alvo de entrada. Desta maneira, 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 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 pode ser o próprio completamente aleatório gerado no primeiro passo ou um novo vetor de *random-keys* que não é decodificado e nem introduzido na população de entrada. Esta escolha depende do novo parâmetro de entrada  $\alpha$ , o método deve utilizar o indivíduo do primeiro passo com  $100\alpha\%$  de probabilidade, caso contrário, um novo vetor de *random-keys*.

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 mantêm 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  $\rho$ , 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. Assim, 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. Portanto, a vantagem de manter o parâmetro  $\alpha$  elevado é viciar ainda mais as chaves durante as iterações da meta-heurística, pois utilizamos o mesmo indivíduo completamente aleatório diversas vezes no VCD. A desvantagem é que este indivíduo pode codificar uma solução muito ruim, e viciar a chave dele pode tornar o processo evolutivo mais lento. Portanto,  $\alpha$  é um parâmetro que deve ser devidamente calibrado para balancear a frequência com o que as mesmas *keys* aparecem durante as gerações. O Algoritmo 4.10 detalha o VCD proposto, tal que o método *Reproduzir* é o processo de reprodução do BRKGA e a função *Decodifica* é o Algoritmo 4.9 utilizando a heurística construtiva detalhada pelo Algoritmo 4.8. Assim, o BRKGA/VCD é o BRKGA original utilizando o Algoritmo 4.10 para gerar a população inicial e mutantes aliado do procedimento preliminar descrito para introduzir os Universais.

#### 4.10.2 Complexidade

Esta seção descreve a complexidade de tempo no pior caso dos componentes do BRKGA e do BRKGA/VCD proposto para o 2BP e o 3BP através da contagem de instruções básicas, com o objetivo de descrever o comportamento prático de uma iteração do método. O processo iterativo destes algoritmos depende de um critério de parada que, geralmente, é uma quantidade máxima de iterações ou tempo de execução.

---

**Algoritmo 4.10** Variable Cross Descent
 

---

```

1: função VCD(População:  $P, U$ ; Tamanho:  $q$ ; Taxa de escolha:  $\alpha$ ; Taxa da
   reprodução:  $\rho$ ; Itens:  $I$ ; Dimensões:  $(W, H, D)$ )
2:   enquanto  $|P| < q$  faça                                ▷ Enquanto a população não estiver preenchida
3:      $i \leftarrow$  Novo indivíduo completamente aleatório
4:      $s \leftarrow$  DECODIFICA( $i, I, (W, H, D)$ )                ▷ Algoritmo 4.9
5:     Aptidão de  $i$ ,  $A \leftarrow$  Avaliação de  $s$ 
6:      $P \leftarrow i$                                           ▷ Introduz  $i$  na população já com avaliação
7:      $k \leftarrow 1$                                           ▷ Procedimento inspirado por VND
8:     enquanto  $k \leq |U|$  e tamanho de  $P < q$  faça        ▷  $U$  armazena os Universais
9:        $i' \leftarrow i$  com 100 $\alpha\%$  de chance ou gere um novo indivíduo completamente aleatório
10:       $f \leftarrow$  REPRODUZIR( $U_k, i', \rho$ )                 ▷ Filho do universal  $k$  com  $i'$ 
11:       $s \leftarrow$  DECODIFICA( $f, I, (W, H, D)$ )             ▷ Algoritmo 4.9
12:      Aptidão de  $f \leftarrow$  Avaliação de  $s$ 
13:       $P \leftarrow f$                                           ▷ Introduz o filho em  $P$  já com avaliação
14:      se Aptidão de  $f < A$  então
15:         $A \leftarrow$  Aptidão de  $f$ 
16:         $k \leftarrow 0$ 
17:      senão  $k \leftarrow k + 1$ 
18:      fim se
19:    fim enquanto
20:  fim enquanto
21:  retorna  $P$ 
22: fim função

```

---

O decodificador utilizado em ambas as versões do BRKGA é detalhado pelo Algoritmo 4.9. O primeiro passo 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 da instância. 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 *Mergesort*. Então, este passo tem complexidade  $\mathcal{O}(n \log n)$ . O passo seguinte realiza o empacotamento dos itens através do Algoritmo 4.8, que é uma modificação da heurística construtiva proposta na Seção 4.5. A modificação altera os parâmetros de entrada para receber duas novas listas de *random-keys* para tornar determinística a escolha do CI caso o critério de avaliação de um EM seja Best Fit ou Best EM e a orientação de cada item. Esta modificação não altera a complexidade estabelecida na Seção 4.5.3. 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 4.2. 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 4.8 é a mesma da heurística construtiva original, sendo  $\mathcal{O}(n \times \text{máximo}(n \times m \log m, m^2))$ , 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 4.5.3, 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 grande maioria das instâncias.

De acordo com o Algoritmo 2.11, o passo 2 sintetiza a população inicial de acordo com a 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 cláusula 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 meta-heurística. 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  $\mathcal{O}(P_{max} \log P_{max})$  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 4.10 deve substituir a cláusula de repetição do processo preliminar da versão original, mantendo o passo de ordenação da população do melhor para o pior indivíduo. 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. Após cada ordenação, uma permutação que representa a sequências de itens a serem empacotados é gerada e passada como parâmetro de entrada para o Algoritmo 4.8 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 o 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 de indivíduos. Portanto, a complexidade do passo de geração dos Universais é dada pelo processo de empacotamento, sendo  $\mathcal{O}(n \times m^2)$  para a grande 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 para o processo preliminar recebe como parâmetro de entrada um número inteiro positivo  $q$  para preencher a população  $P$ , também passada como entrada, com esta quantidade de indivíduos. Ela deve receber também o vetor com os Universais e os parâmetros adicionais para a reprodução, o empacotamento e a taxa de escolha. O passo 2 do VCD é uma cláusula de repetição que realiza até  $|P| - q$  repetições. O passo 3 gera um vetor completamente aleatório com até  $3n$  *random-keys*, portanto, tem complexidade  $\Theta(n)$ . O passo 4 utiliza o decodificador para obter uma solução, então tem complexidade  $\mathcal{O}(n \times m^2)$  para a grande maioria das instâncias. O passo 6 introduz o indivíduo no final da população que deve ser implementada como um vetor de indivíduos. Assim, tem complexidade constante. O passo 9 é a escolha do segundo parente para a reprodução. Independente da escolha, o cromossomo deste parente não é alterado. Portanto, se o processo utilizar o indivíduo já gerado, a implementação pode utilizar uma referência para o último elemento de  $P$ . Caso contrário, temos o pior caso onde até  $3n$  *random-keys* devem ser geradas para sintetizar um novo parente que nunca será decodificado. Desta maneira, a complexidade do passo 9 é  $\Theta(n)$ . O passo 10 executa o processo de reprodução para gerar um filho. Este processo é uma cláusula de repetição que itera sobre os genes do filho. Portanto, realiza até  $3n$  repetições. Para cada gene, um número aleatório deve ser gerado e comparado através da taxa de elitismo  $\epsilon$ , 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 o indivíduo gerado no passo 9. Assim, a complexidade do processo de reprodução e do passo 10 é  $\Theta(n)$ . O passo 11 decodifica o filho gerado no passo 10. Então, tem complexidade  $\mathcal{O}(n \times m^2)$  para a grande maioria das instâncias. O restante dos passos não mencionados, exceto o passo 8, são comparações e atribuições entre referências ou variáveis básicas, portanto, têm complexidade constante. O passo 8 é uma cláusula de repetição controlada com a variável  $k$ , análoga ao VND. O número de repetições deste passo 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. Desta maneira, uma iteração do passo 8 diminui uma repetição que o passo 2 realiza, por causa da cardinalidade de  $P$  que aumenta em 1 a cada iteração. Ou seja, o número de repetições que os passos 2 e 8 realizam em conjunto é exatamente  $|P| - q$ . Observe também que a complexidade do conjunto de passos internos da cláusula de repetição

presente no passo 8 é a mesma do conjunto de passos internos da clausula presente no passo 2 (excluindo o próprio passo 8), sendo dada pelo decodificador como  $\mathcal{O}(n \times m^2)$  para a grande maioria das instâncias. Então, o conjunto de instruções internas que compõe as iterações dos passos 2 e 8 tem o mesmo objetivo: decodificar indivíduos e inseri-los em  $P$ . Portanto, 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. Então, a população que é passada como parâmetro de entrada para o VCD neste procedimento é o próprio vetor com os Universais já inseridos. 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  $P_{max}$ . Assim, a complexidade do VCD majora as dos outros passos efetuados durante o procedimento preliminar. Portanto, a complexidade do procedimento preliminar no BRKGA/VCD é  $\mathcal{O}(P_{max} \times n \times m^2)$ .

Os passos realizados durante o processo iterativo do BRKGA e do BRKGA/VCD é descrito pelo Algoritmo 2.11. Como mencionado anteriormente, considere que a população é um vetor de indivíduos. Portanto, qualquer inserção de um novo elemento pode ser efetuado no final deste vetor com complexidade constante. Desta maneira, o primeiro passo da iteração é o passo 5, que realiza o procedimento de seleção. A seleção consiste em copiar os 100 $\epsilon$ % melhores indivíduos da geração anterior para a geração atual, ou seja, a porção Elite de  $P_{max} \times \epsilon$  indivíduos. Como um indivíduo não é representado por uma variável básica de uma linguagem de programação, 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. Portanto, a complexidade da cópia de um cromossomo é  $\Theta(n)$ . Então, a complexidade do passo 5 é  $\Theta(P_{max} \times \epsilon \times n)$ . O passo 6 realiza o processo de mutação. No BRKGA, a mutação gera 100 $\omega$ % indivíduos completamente aleatórios e os insere na geração atual. Posteriormente no passo 8, os mutantes devem ser decodificados. Podemos decodificar os mutantes assim que eles são gerados no passo 6. Assim, 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 completamente aleatória 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. Portanto, a implementação do passo 6 no BRKGA em conjunto com as decodificações parciais do passo 8 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. Portanto, o número de indivíduos que o VCD deve gerar neste passo é  $P_{max} \times \omega$ . Então, a complexidade do passo 6 no BRKGA/VCD é  $\mathcal{O}(P_{max} \times \omega \times n \times m^2)$  para a grande maioria das instâncias. O passo 7 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*. Portanto, a reprodução tem complexidade  $\Theta(n)$ . Os filhos gerados devem ser decodificados no passo 8. Portanto, a implementação do passo 7 pode ser uma cláusula de repetição que executa a reprodução entre um parente Elite e um não-Elite aleatório e imediatamente decodifica o filho gerado. Assim, a quantidade de instruções em conjunto com as instruções necessárias para realizar parcialmente o passo 8 nos dá a complexidade do passo 7. 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 7 é  $\mathcal{O}(P_{max}(1 - \epsilon - \omega) \times n \times m^2)$ . Finalmente, o passo 8 decodifica e ordena a geração atual do melhor para o pior. Como as decodificações podem ser efetuadas durante os passos 6 e 7 no BRKGA e no BRKGA/VCD, o passo 8 consiste somente em ordenar a população com um algoritmo eficiente. Portanto, a complexidade deste passo último passo é  $P_{max} \log P_{max}$ . Assim, a complexidade final de uma iteração do BRKGA e do BRKGA/VCD é dada pelos passos 6 e 7, sendo  $\mathcal{O}(P_{max} \times n \times m^2 \times (1 - \text{máximo}(\epsilon, \omega)))$ .

A Tabela 4.8 sumariza as complexidades dos processos que compõem o BRKGA e o BRKGA/VCD em conjunto com a complexidade final de uma iteração. A tabela relaciona a instrução com o respectivo passo no Algoritmo 2.11 e sua complexidade.

Tabela 4.8: Sumário da complexidade dos componentes do BRKGA e do BRKGA/VCD.

Processo	Passo	Complexidade
Decodificador (Algoritmo 4.9)	3 e 8	$\mathcal{O}(n \times m^2)^*$
Preliminar (BRKGA)	2 e 3	$\mathcal{O}(P_{max} \times n \times m^2)^*$
Preliminar (BRKGA/VCD)	2 e 3	$\mathcal{O}(P_{max} \times n \times m^2)^*$
VCD (Algoritmo 4.10)	2, 3, 6 e 8	$\mathcal{O}(( P  - q) \times n \times m^2)^*$
Seleção	5	$\Theta(P_{max} \times \epsilon \times n)$
Mutação com decodificação (BRKGA)	6 e 8	$\mathcal{O}(P_{max} \times \omega \times n \times m^2)^*$
Mutação com decodificação (BRKGA/VCD)	6 e 8	$\mathcal{O}(P_{max} \times \omega \times n \times m^2)^*$
Reprodução com decodificação	7 e 8	$\Theta(n)$
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.

## Capítulo 5

# Experimentos Computacionais Preliminares

Este capítulo apresenta os resultados computacionais preliminares com uma versão promissora de cada algoritmo proposto no Capítulo 4 para o 3BP e o 2BP, com o objetivo de mostrar que esses métodos são capazes de produzir soluções de alta qualidade para instâncias grandes dos problemas de empacotamento, com baixo tempo de execução quando usados em um notebook padrão. Os resultados são referentes a um extensivo teste empírico efetuado com o conjunto de 820 instâncias detalhadas na Seção 5.1. Os algoritmos propostos são comparados com os métodos estado da arte encontrados na literatura. Esta comparação preliminar utiliza os resultados obtidos neste trabalho para contrastar com os dados reportados nos trabalhos da literatura em termos de qualidade encontrada na solução final do 3BP e do 2BP sem rotação.

### 5.1 Instâncias

#### 5.1.1 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 5.1. A

Tabela 5.1: Tipos de itens utilizados nas classes 1 até 5 das instâncias 3D

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	$[1, \frac{1}{2}W]$	$[1, \frac{1}{2}H]$	$[1, \frac{1}{2}D]$

tabela relaciona o tipo com o intervalo que cada dimensão  $(w_i, h_i, d_i)$  pode assumir,  $i \in \{1, \dots, n\}$ . 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.

- Classe 1 até 5:  $W = H = D = 10$ ; Para cada classe  $k \in \{1, 2, 3, 4, 5\}$ , o item é do tipo  $k$  com 60% de probabilidade e os outros 4 tipos com 10% cada de acordo com a Tabela 5.1.
- 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  $[1, 100]$ .

### 5.1.2 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) propôs um conjunto de instâncias com 6 classes. Assim, 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 5.2. A tabela relaciona o tipo do item com o intervalo que cada parâmetro pode assumir. 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  $[1, 35]$ .



Tabela 5.2: Tipos de itens utilizados nas classes 7 até 10 das instâncias 2D

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	$[1, \frac{1}{2}W]$	$[1, \frac{1}{2}H]$

- 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 5.2.

## 5.2 Implementação e Ambiente Computacional

A implementação de todos os componentes e de cada algoritmo detalhado no Capítulo 4 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. A implementação foi compilada através do *GCC* do *GNU Compiler Collection*<sup>1</sup> com a *flag* de otimização  $-O3$ . O ambiente computacional utilizado em todos os teste 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).

## 5.3 Algoritmos Comparados

Com o objetivo de avaliar a qualidade das soluções obtidas neste trabalho, os resultados preliminares são comparados com os dados reportados nos trabalhos da literatura relacionados pela Tabela 5.3, que relaciona os algoritmos estado da arte para o 3BP e o 2BP sem

<sup>1</sup>Disponível no seguinte sítio eletrônico: <<https://gcc.gnu.org/>>.

rotação. Estes trabalhos utilizam o mesmo conjunto de instâncias detalhados na Seção 5.1 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 apresentado na Seção 5.2. 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* do *GNU Compiler Collection* com a *flag* de otimização *-O3*.

Tabela 5.3: Métodos literatura utilizados na comparação preliminar.

Algoritmo	Referência	Estratégia
MVP	(MARTELLO; PISINGER; VIGO, 2000)	Branch-and-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)	Algoritmo Genético de Chaves Aleatórias Viciadas (3BP/2BP)
HPB	(BOSCHETTI; MINGOZZI, 2003b)	Heurística (2BP)
SCH	(MONACI; TOTH, 2006)	Heurística baseada em cobertura de conjunto (3BP)

## 5.4 Resultados Computacionais Preliminares

Esta seção apresenta os resultados preliminares obtidos por uma boa versão de cada método proposto. As versões foram testadas e calibradas manualmente com um subconjunto de instâncias do 3BP detalhadas na Seção 5.1, porém nem todas as possíveis configurações foram testadas na versão atual deste trabalho. Portanto, as próximas seções reportam o resultado obtido pela melhor configuração conhecida no momento sem garantir que elas são as melhores disposições de cada método proposto.

### 5.4.1 Greedy Randomized Adaptative Search Procedure

O algoritmo GRASP proposto na Seção 4.8 é uma extensão do método apresentado em (ZUDIO et al., 2017a). O trabalho citado reporta que o Best EM com  $\alpha = 0,2$  fornece os melhores resultados quando utilizado no empacotamento e reempacotamento dos itens durante as fases de construção e melhora respectivamente. Portanto, este é o critério de avaliação que será utilizado nesta seção, em conjunto com o parâmetro  $\alpha$  estabelecido. Além disso, os melhores resultados atualmente para este método são obtidos quando utiliza-se a estrutura VND com as quatro vizinhanças combinadas, tal que  $V_1$  deve ser implementada com *Best Improvement*. Portanto, a versão utilizada nesta seção é um GRASP/VND híbrido. Os resultados finais melhoram consideravelmente quando os itens de entrada são ordenados pelo maior volume comparado com ordenações aleatórias. As permutações passadas para a heurística construtiva na primeira fase que representam a sequência de itens a serem empacotados são sintetizadas de maneira a empacotar os itens começando pelo de maior volume. O critério de parada é 10 segundos de execução em CPU. A Tabela 5.4 sumariza a disposição utilizada no GRASP/VND desta seção, relacionando o componente ou o parâmetro de entrada com sua configuração.

Tabela 5.4: Configuração utilizada no GRASP/VND.

Componente	Configuração
Critério de Parada	10 segundos de execução em CPU
Função objetivo	FOT
Fase de construção/ Heurística construtiva	Empacotamento com Best EM $\alpha = 0,2$ Itens ordenados por maior volume
Fase de melhora/VND	Com as vizinhanças $V_1, V_2, V_3$ e $V_4$
$V_1, V_2, V_3$ e $V_4$	Reempacotamento com Best EM
$V_1$	Best Improvement

O GRASP/VND executou 10 vezes com sementes sequenciais no intervalo de  $[1, 10]$  para cada instância do conjunto estabelecido de 320 entradas do 3BP e 500 do 2BP, a fim de controlar o âmbito de execução do gerador de números pseudoaleatórios. A solução é o resultado médio dessas execuções. A Tabela 5.5 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  $TT$  é a média do tempo total para encontrar a melhor solução. As colunas restantes mostram os resultados dos algoritmos da literatura reportados nos trabalhos referenciados pela Tabela 5.3. A exceção é o algoritmo MPV, pois os valores reportados são as médias de 10 execuções no mesmo recurso computacional deste trabalho detalhado na Seção 5.2, com o tempo limite de execução em CPU de 60 segundos. As duas últimas linhas reportam o

número total acumulado de caixas utilizadas em todas as instâncias para a comparação entre os algoritmos.

A tabela mostra que vários conjuntos de soluções são equivalentes às melhores reportadas na literatura, particularmente o conjunto de 40 instâncias da classe 4. Para este conjunto, o GRASP/VND obtém o provável ótimo global em poucas iterações. Podemos observar também que as outras soluções se aproximam da melhor reportada, sendo que muitas vezes o GRASP/VND falha em determinar o provável ótimo global com uma diferença de 1 ou 2 caixas do acumulado de um conjunto de 10 instâncias. Por exemplo, os conjuntos de instâncias das classes 1 e 5 com  $n = \{100, 150, 200\}$  exemplificam resultados que são próximos do provável ótimo global. A tabela mostra que os resultados obtidos pelo GRASP/VND são sistematicamente melhores que o MVP em todas as instâncias. O MVP foi executado no mesmo ambiente computacional deste trabalho com o critério de parada sendo um tempo limite 6 vezes maior do que o GRASP/VND. Além disso, os dados mostram que o GRASP/VND obtém soluções melhores ou equivalentes do que o GASP e o EHG2; grande quantidade de soluções melhores ou equivalentes ao TS3 e o GLS com um acúmulo final de caixas melhor; e algumas soluções equivalentes ao GVND e BRKGA, que por sua vez obtém soluções melhores que o GRASP/VND. O teste de Wilcoxon pareado foi aplicado para decidir se as diferenças que são observadas na tabela podem ser estatisticamente significantes. O teste mostra que o GRASP/VND é significativamente melhor que o MVP, GASP e EHG2; equivalente ao TS3 e o GLS; e pior que o GVND e o BRKGA.

A Tabela 5.6 exhibe os resultados para as instâncias do 2BP sem rotação com a mesma configuração da tabela anterior: os valores reportados 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  $TT$  de tempo total para a melhor solução. As demais colunas reportam os valores obtidos pelos métodos da literatura.

Os resultados mostram que o GRASP/VND obtém soluções com qualidade equivalente ao melhor valor reportado da literatura com um tempo de execução inferior a 1 segundo no recurso computacional utilizado para a grande maioria das instâncias. As soluções encontradas são provavelmente o ótimo global. A tabela mostra que os resultados obtidos são equivalentes ou melhores que os resultados reportados pelos métodos TS3, GLS e HPB. Outra observação é que o acúmulo final de caixas para todas as instâncias mostra que o GRASP/VND obtém em média soluções de qualidade equivalente aos outros métodos: perdendo por uma margem de 0,3% para o BRKGA; 0,2% para GVND e SCH; e ganhando por uma margem igualmente pequena de TS3, GLS e HBP. O teste de Wilcoxon pareado mostra que o GRASP/VND é significantemente melhor que o TS3 e o GLS, enquanto que os demais dados não apresentam significância estatística.

Tabela 5.5: Resultado preliminar do GRASP/VND para o 3BP sem rotação.

$Cl$	$n$	$TT(s)$	$Z$	MVP	TS3	GLS	GASP	EHGH2	GVND	BRKGA
1	50	4,1	<b>13,4</b>	13,6	13,4	13,4	13,4	13,8	13,4	13,4
	100	3,2	26,7	27,4	26,6	26,7	26,9	27,6	26,6	26,6
	150	2,8	36,6	37,7	36,7	37,0	37,0	39,8	36,4	36,4
	200	3,1	51,0	52,0	51,2	51,2	51,6	50,6	50,9	50,8
2	50	0,8	<b>13,8</b>	13,9	13,8	—	—	—	13,8	13,8
	100	2,8	26,0	26,4	25,7	—	—	—	25,7	25,6
	150	1,0	37,0	38,1	37,2	—	—	—	36,9	36,6
	200	1,8	49,6	50,2	50,1	—	—	—	49,4	49,4
3	50	0,9	<b>13,3</b>	13,6	13,3	—	—	—	13,3	13,3
	100	3,8	26,2	27,1	26,0	—	—	—	26,0	25,9
	150	0,4	37,7	39,2	37,7	—	—	—	37,6	37,5
	200	2,7	50,5	51,0	50,5	—	—	—	50,0	49,8
4	50	0,0	<b>29,4</b>	29,4	29,4	29,4	29,4	29,4	29,4	29,4
	100	0,1	<b>59,0</b>	59,2	59,0	59,0	59,0	59,5	59,0	59,0
	150	0,1	<b>86,8</b>	87,5	86,8	86,8	86,8	90,4	86,8	86,8
	200	0,3	<b>118,8</b>	119,0	118,8	119,0	118,8	119,0	118,8	118,8
5	50	0,0	8,4	9,2	8,4	8,3	8,4	7,9	8,3	8,3
	100	0,0	15,1	17,1	15,0	15,1	15,1	14,6	15,0	15,0
	150	0,3	20,2	24,0	20,4	20,2	20,6	21,5	20,1	20,0
	200	0,3	27,2	28,2	27,6	27,2	27,7	29,6	27,1	27,1
6	50	1,4	9,8	9,8	9,9	9,8	9,9	11,8	9,8	9,7
	100	0,9	19,1	19,3	19,1	19,1	19,1	19,2	19,0	18,9
	150	6,0	29,3	30,3	29,4	29,4	29,5	29,8	29,2	29,0
	200	8,1	37,7	37,8	37,7	37,7	38,0	38,7	37,4	37,3
7	50	0,6	<b>7,4</b>	7,9	7,5	7,4	7,5	7,4	7,4	7,4
	100	0,3	12,5	15,5	12,5	12,3	12,7	13,5	12,5	12,2
	150	0,2	16,0	19,9	16,1	15,8	16,6	18,2	16,0	15,3
	200	1,4	23,5	24,2	23,9	23,5	24,2	24,1	23,5	23,4
8	50	0,1	<b>9,2</b>	9,5	9,3	9,2	9,3	9,4	9,2	9,2
	100	0,4	<b>18,9</b>	21,4	18,9	18,9	19,0	18,9	18,9	18,9
	150	1,0	24,2	27,5	24,1	23,9	24,8	26,0	24,1	23,6
	200	4,5	29,9	31,1	30,3	29,9	31,1	35,8	29,8	29,3
(1, 4 – 8)			7301	7585	7320	7302	7364	7565	7286	7258
(1 – 8)			9842	10180	9863				9813	9777

\*Valores em negrito correspondem a soluções equivalentes às melhores

Vale observar que o GRASP/VND é um método simples em relação aos algoritmos comparados da literatura. Por exemplo, o GVND utiliza um agrupamento de itens efetuado para compactar o empacotamento, tal que uma iteração GRASP encontra várias soluções inviáveis e somente finaliza quando os itens são empacotados em uma configuração melhor que a atual. O algoritmo de agrupamento consiste de um processo complexo baseado em uma heurística que resolve o problema de contêineres publicado em (PARRERO et al., 2008). Além disso, o GRASP/VND utiliza somente um parâmetro calibrável,

Tabela 5.6: Resultado preliminar do GRASP/VND para o 2BP sem rotação.

Cl	$n$	$TT(s)$	$Z$	TS3	GLS	HBP	SCH	GVND	BRKGA
1	20	0,0	<b>7,1</b>	7,1	7,1	7,1	7,1	7,1	7,1
	40	0,0	<b>13,4</b>	13,5	13,4	13,4	13,4	13,4	13,4
	60	1,7	<b>20,0</b>	20,1	20,1	20,1	20,0	20,0	20,0
	80	0,0	<b>27,5</b>	28,2	27,5	27,5	27,5	27,5	27,5
	100	0,0	31,8	32,6	32,1	31,8	31,7	31,7	31,7
2	20	0,0	<b>1,0</b>	1,0	1,0	1,0	1,0	1,0	1,0
	40	0,1	<b>1,9</b>	2,0	1,9	1,9	1,9	1,9	1,9
	60	0,0	<b>2,5</b>	2,7	2,5	2,5	2,5	2,5	2,5
	80	0,0	<b>3,1</b>	3,3	3,1	3,1	3,1	3,1	3,1
	100	0,0	<b>3,9</b>	4,0	3,9	3,9	3,9	3,9	3,9
3	20	0,9	<b>5,1</b>	5,5	5,1	5,1	5,1	5,1	5,1
	40	0,4	9,5	9,7	9,4	9,5	9,4	9,4	9,4
	60	0,0	14,0	14,0	14,0	14,0	13,9	13,9	13,9
	80	0,1	19,0	19,8	19,1	19,1	18,9	18,9	18,9
	100	2,9	<b>22,3</b>	23,6	22,6	22,6	22,3	22,3	22,3
4	20	0,0	<b>1,0</b>	1,0	1,0	1,0	1,0	1,0	1,0
	40	0,0	<b>1,9</b>	1,9	1,9	1,9	1,9	1,9	1,9
	60	0,0	<b>2,5</b>	2,6	2,5	2,5	2,5	2,5	2,5
	80	0,0	3,2	3,3	3,3	3,3	3,2	3,1	3,1
	100	0,0	3,8	4,0	3,8	3,8	3,8	3,8	3,7
5	20	0,0	<b>6,5</b>	6,6	6,5	6,5	6,5	6,5	6,5
	40	0,2	<b>11,9</b>	11,9	11,9	11,9	11,9	11,9	11,9
	60	0,0	18,1	18,2	18,1	18,0	18,0	18,0	18,0
	80	0,0	<b>24,7</b>	25,1	24,9	24,8	24,7	24,7	24,7
	100	1,6	28,2	29,5	28,8	28,7	28,2	28,2	28,1
6	20	0,0	<b>1,0</b>	1,0	1,0	1,0	1,0	1,0	1,0
	40	1,2	1,7	1,9	1,8	1,8	1,7	1,7	1,6
	60	0,0	<b>2,1</b>	2,2	2,2	2,1	2,1	2,1	2,1
	80	0,0	<b>3,0</b>	3,0	3,0	3,0	3,0	3,0	3,0
	100	0,0	3,4	3,4	3,4	3,4	3,4	3,4	3,3
7	20	0,0	<b>5,5</b>	5,5	5,5	5,5	5,5	5,5	5,5
	40	0,0	11,4	11,4	11,3	11,1	11,1	11,1	11,1
	60	0,1	15,9	16,2	15,9	16,0	15,8	15,9	15,8
	80	0,0	<b>23,2</b>	23,2	23,2	23,2	23,2	23,2	23,2
	100	0,0	27,2	27,7	27,5	27,4	27,1	27,1	27,1
8	20	0,0	<b>5,8</b>	5,8	5,8	5,8	5,8	5,8	5,8
	40	0,0	11,4	11,4	11,4	11,3	11,3	11,3	11,3
	60	0,0	16,2	16,2	16,3	16,2	16,2	16,1	16,1
	80	0,0	22,5	22,6	22,5	22,6	22,4	22,4	22,4
	100	0,0	<b>27,8</b>	28,4	28,1	28,0	27,9	27,8	27,8
9	20	0,0	<b>14,3</b>	14,3	14,3	14,3	14,3	14,3	14,3
	40	0,0	<b>27,8</b>	27,8	27,8	27,8	27,8	27,8	27,8
	60	0,0	<b>43,7</b>	43,8	43,7	43,7	43,7	43,7	43,7
	80	0,0	<b>57,7</b>	57,7	57,7	57,7	57,7	57,7	57,7
	100	0,0	<b>69,5</b>	69,5	69,5	69,5	69,5	69,5	69,5
10	20	0,0	<b>4,2</b>	4,3	4,2	4,3	4,2	4,2	4,2
	40	0,0	<b>7,4</b>	7,5	7,4	7,4	7,4	7,4	7,4
	60	0,7	10,2	10,4	10,2	10,2	10,1	10,0	10,0
	80	0,0	13,0	13,0	13,0	13,0	12,8	12,9	12,8
	100	0,2	15,9	16,6	16,2	16,2	15,9	15,9	15,8
(1 – 10)			7257	7360	7284	7275	7243	7241	7234

\*Valores em negrito correspondem a soluções equivalentes às melhores

o que é vantajoso quando comparado a um método como o BRKGA, que utiliza vários parâmetros difíceis de serem calibrados.

### 5.4.2 Busca Tabu

O algoritmo busca tabu utilizado nesta seção é o BT/VN detalhado na Seção 4.9. Os melhores resultados obtidos na versão atual deste trabalho com o BT/VN são produzidos quando a solução inicial é construída através da heurística construtiva proposta com o empacotamento através do Best EM, que por sua vez deve receber permutações de entrada que representam sequências aleatórias de itens para empacotar. O parâmetro que limita o número máximo de soluções tabu  $T_{max}$  recebeu o valor 10. Esta quantidade fornece um bom balanceamento para a memória adaptativa durante o processo de busca, onde o impacto na eficiência do algoritmo é baixa em termos de tempo de execução para administrar o tabu. A utilização de todas as vizinhanças  $V_2$ ,  $v_3$  e  $V_4$  antes de executar  $V_1$  causa um impacto significativo positivo na qualidade das soluções finais, sendo que o reempacotamento é sempre realizado através do Best EM. A Tabela 5.7 relaciona a disposição utilizada no BT/VN desta seção, exibindo os componentes com sua respectiva configuração.

Tabela 5.7: Configuração dos componentes do BT/VN.

Componente	Configuração
Critério de parada	10 segundos de execução em CPU
Função objetivo	FOT
Solução Inicial/ Algoritmo 4.2	Empacotamento com Best EM Sequência de itens aleatória
Tamanho máximo do tabu	$T_{max} = 10$
VN	Com $V_2$ , $V_3$ e $V_4$ executadas antes de $V_1$
$V_1$ , $V_2$ , $V_3$ e $V_4$	Reempacotamento com Best EM

Para cada instância do conjunto estabelecido do 3BP e do 2BP, 10 soluções iniciais foram sintetizadas com permutações aleatórias geradas com sementes sequencias no intervalo  $[1, 10]$ . Essas soluções foram passadas como parâmetro de entrada para o BT/VN, que, por sua vez, executou por 10 segundos com cada uma delas, contados a partir da geração da própria solução inicial. A solução reportada para o BT/VN é a média dessas execuções.

A Tabela 5.8 apresenta os resultados obtidos para as instâncias do 3BP sem rotação, onde os valores são as médias de 10 instâncias da classe  $Cl$  para o número de itens  $n$  relacionados nas colunas 1 e 2. As colunas seguintes são os resultados referentes ao BT/VN, onde  $Z$  é a média da qualidade do conjunto de instâncias e  $TT$  é a média do tempo total para encontrar a melhor solução. O restante relaciona os resultados reportados pelos respectivos trabalhos da literatura com exceção do MVP. O MVP foi

executado no próprio ambiente computacional deste trabalho com o critério de parada sendo 60 segundos de execução em CPU. As duas últimas linhas apresentam o acumulado de caixas obtidas em todas as instâncias para a comparação entre os algoritmos.

Tabela 5.8: Resultado preliminar do BT/VN para o 3BP sem rotação.

$Cl$	$n$	$TT(s)$	$Z$	MVP	TS3	GLS	GASP	EHGH2	GVND	BRKGA
1	50	0,1	<b>13,4</b>	13,6	13,4	13,4	13,4	13,8	13,4	13,4
	100	0,2	<b>26,6</b>	27,4	26,6	26,7	26,9	27,6	26,6	26,6
	150	6,9	<b>36,4</b>	37,7	36,7	37,0	37,0	39,8	36,4	36,4
	200	4,0	50,9	52,0	51,2	51,2	51,6	50,6	50,9	50,8
2	50	0,8	<b>13,8</b>	13,9	13,8	—	—	—	13,8	13,8
	100	1,4	25,7	26,4	25,7	—	—	—	25,7	25,6
	150	0,9	37,0	38,1	37,2	—	—	—	36,9	36,6
	200	1,5	49,5	50,2	50,1	—	—	—	49,4	49,4
3	50	0,1	<b>13,3</b>	13,6	13,3	—	—	—	13,3	13,3
	100	5,6	26,1	27,1	26,0	—	—	—	26,0	25,9
	150	8,2	37,6	39,2	37,7	—	—	—	37,6	37,5
	200	1,1	50,2	51,0	50,5	—	—	—	50,0	49,8
4	50	0,0	<b>29,4</b>	29,4	29,4	29,4	29,4	29,4	29,4	29,4
	100	0,0	<b>59,0</b>	59,2	59,0	59,0	59,0	59,5	59,0	59,0
	150	0,0	<b>86,8</b>	87,5	86,8	86,8	86,8	90,4	86,8	86,8
	200	0,1	<b>118,8</b>	119,0	118,8	119,0	118,8	119,0	118,8	118,8
5	50	1,1	<b>8,3</b>	9,2	8,4	8,3	8,4	7,9	8,3	8,3
	100	3,2	15,0	17,1	15,0	15,1	15,1	14,6	15,0	15,0
	150	2,3	20,1	24,0	20,4	20,2	20,6	21,5	20,1	20,0
	200	0,1	27,2	28,2	27,6	27,2	27,7	29,6	27,1	27,1
6	50	0,1	9,8	9,8	9,9	9,8	9,9	11,8	9,8	9,7
	100	0,2	19,1	19,3	19,1	19,1	19,1	19,2	19,0	18,9
	150	5,3	29,3	30,3	29,4	29,4	29,5	29,8	29,2	29,0
	200	5,1	37,7	37,8	37,7	37,7	38,0	38,7	37,4	37,3
7	50	0,8	<b>7,4</b>	7,9	7,5	7,4	7,5	7,4	7,4	7,4
	100	0,1	12,5	15,5	12,5	12,3	12,7	13,5	12,5	12,2
	150	0,1	16,0	19,9	16,1	15,8	16,6	18,2	16,0	15,3
	200	1,3	23,5	24,2	23,9	23,5	24,2	24,1	23,5	23,4
8	50	1,6	<b>9,2</b>	9,5	9,3	9,2	9,3	9,4	9,2	9,2
	100	4,3	<b>18,9</b>	21,4	18,9	18,9	19,0	18,9	18,9	18,9
	150	0,5	24,2	27,5	24,1	23,9	24,8	26,0	24,1	23,6
	200	8,1	29,8	31,1	30,3	29,9	31,1	35,8	29,8	29,3
(1, 4 – 8)			7293	7585	7320	7302	7364	7565	7286	7258
(1 – 8)			9825	10180	9863				9813	9777

\*Valores em negrito correspondem a soluções equivalentes às melhores

A tabela mostra que várias soluções obtidas pelo BT/VN têm qualidade equivalente ao melhor valor reportado da literatura. Assim como o GRASP/VND, o BT/VN provê o provável ótimo global em poucas iterações para este conjunto, sendo que todos os resultados obtidos pelo BT/VN para o conjunto de instâncias tridimensional são melhores ou



equivalentes às soluções encontradas pelo GRASP/VND. Muitas vezes os valores reportados na tabela se aproximam do melhor reportado, onde a melhor solução encontrada em um trabalho da literatura apresenta um acumulado com 1 ou 2 caixas a menos para o determinado subconjunto de 10 instâncias. O BT/VN obtém soluções melhores ou equivalentes a todos os resultados encontrados pelo MVP, que foi executado no mesmo ambiente computacional com um tempo de execução em CPU 6 vezes maior. Além disso, a tabela mostra que o BT/VN obtém soluções melhores ou equivalentes para a maioria das instâncias comparado com o GASP, EHG2, TS3 e o GLS com um acúmulo final de caixas melhor. Porém, o GVND e o BRKGA reportam um acúmulo final de caixas melhor que o BT/VN, sendo que o algoritmo proposto neste trabalho obtém várias soluções equivalentes. O teste de Wilcoxon pareado mostra que o BT/VN é significativamente melhor que o MVP, GASP e EHG2; equivalente ao TS3 e GLS; e pior do que o GVND e BRKGA.

A Tabela 5.9 exhibe os resultados para as instâncias do 2BP sem rotação com a mesma disposição da tabela anterior. As duas primeiras colunas relacionam a classe  $Cl$  e a quantidade de itens do conjunto composto de 10 instâncias, onde cada valor presente nas linhas são as médias para este conjunto de instância. As colunas 3 e 4 são os resultados referentes ao BT/VN para a média da solução obtida  $Z$  e a média do tempo total  $TT$  de execução para encontrar a melhor solução. O restante é referente aos métodos comparados da literatura.

A tabela para as instâncias bidimensionais mostra que várias soluções obtidas pelo BT/VN são equivalentes às melhores reportadas pela literatura, sendo que o tempo de execução em CPU para encontrá-las é inferior à 1 segundo no recurso computacional utilizado. Tais resultados são o provável ótimo global para o respectivo conjunto de instâncias. O acumulado final de caixas obtido pelo BT/VN mostra que este método é melhor que o TS3, GLS e o HPB. Porém, o BT/VN perde por uma margem pequena para o SCH, GVND e o BRKGA. O teste de Wilcoxon pareado diz que as diferenças observadas na tabela não têm significância estatística.

Observe que o BT/VN é um método simples comparado com os outros algoritmos propostos na literatura, bem como o GRASP/VND. Como citado na seção anterior, alguns métodos utilizam estruturas complexas com base em grafos ou procedimentos que lidam com soluções inviáveis para encontrar boas configurações para o 3BP e o 2BP. Além disso, alguns desses métodos são difíceis de serem estendidos para o caso com rotação ou para versões que assumam restrições físicas adicionais. Finalmente, o BT/VN é um algoritmo que utiliza um único parâmetro calibrável, uma vantagem sobre métodos que utilizam vários parâmetros desse tipo, por exemplo, o BRKGA.

Tabela 5.9: Resultado preliminar do BT/VN para o 2BP sem rotação.

Cl	$n$	$TT(s)$	$Z$	TS3	GLS	HBP	SCH	GVND	BRKGA
1	20	0,2	<b>7,1</b>	7,1	7,1	7,1	7,1	7,1	7,1
	40	0,2	<b>13,4</b>	13,5	13,4	13,4	13,4	13,4	13,4
	60	0,2	<b>20,0</b>	20,1	20,1	20,1	20,0	20,0	20,0
	80	0,1	<b>27,5</b>	28,2	27,5	27,5	27,5	27,5	27,5
	100	0,2	<b>31,7</b>	32,6	32,1	31,8	31,7	31,7	31,7
2	20	0,3	<b>1,0</b>	1,0	1,0	1,0	1,0	1,0	1,0
	40	0,3	<b>1,9</b>	2,0	1,9	1,9	1,9	1,9	1,9
	60	0,3	<b>2,5</b>	2,7	2,5	2,5	2,5	2,5	2,5
	80	0,3	<b>3,1</b>	3,3	3,1	3,1	3,1	3,1	3,1
	100	0,3	<b>3,9</b>	4,0	3,9	3,9	3,9	3,9	3,9
3	20	0,1	<b>5,1</b>	5,5	5,1	5,1	5,1	5,1	5,1
	40	0,2	<b>9,4</b>	9,7	9,4	9,5	9,4	9,4	9,4
	60	0,0	14,0	14,0	14,0	14,0	13,9	13,9	13,9
	80	0,1	19,0	19,8	19,1	19,1	18,9	18,9	18,9
	100	0,1	<b>22,3</b>	23,6	22,6	22,6	22,3	22,3	22,3
4	20	0,3	<b>1,0</b>	1,0	1,0	1,0	1,0	1,0	1,0
	40	0,3	<b>1,9</b>	1,9	1,9	1,9	1,9	1,9	1,9
	60	0,3	<b>2,5</b>	2,6	2,5	2,5	2,5	2,5	2,5
	80	0,3	3,2	3,3	3,3	3,3	3,2	3,1	3,1
	100	0,3	3,8	4,0	3,8	3,8	3,8	3,8	3,7
5	20	0,2	<b>6,5</b>	6,6	6,5	6,5	6,5	6,5	6,5
	40	0,2	<b>11,9</b>	11,9	11,9	11,9	11,9	11,9	11,9
	60	0,2	18,1	18,2	18,1	18,0	18,0	18,0	18,0
	80	0,2	<b>24,7</b>	25,1	24,9	24,8	24,7	24,7	24,7
	100	0,4	28,2	29,5	28,8	28,7	28,2	28,2	28,1
6	20	0,3	<b>1,0</b>	1,0	1,0	1,0	1,0	1,0	1,0
	40	0,4	1,7	1,9	1,8	1,8	1,7	1,7	1,6
	60	0,4	<b>2,1</b>	2,2	2,2	2,1	2,1	2,1	2,1
	80	0,5	<b>3,0</b>	3,0	3,0	3,0	3,0	3,0	3,0
	100	0,2	3,4	3,4	3,4	3,4	3,4	3,4	3,3
7	20	0,1	<b>5,5</b>	5,5	5,5	5,5	5,5	5,5	5,5
	40	0,2	<b>11,1</b>	11,4	11,3	11,1	11,1	11,1	11,1
	60	0,4	15,9	16,2	15,9	16,0	15,8	15,9	15,8
	80	0,5	<b>23,2</b>	23,2	23,2	23,2	23,2	23,2	23,2
	100	0,2	27,2	27,7	27,5	27,4	27,1	27,1	27,1
8	20	0,3	<b>5,8</b>	5,8	5,8	5,8	5,8	5,8	5,8
	40	0,3	<b>11,3</b>	11,4	11,4	11,3	11,3	11,3	11,3
	60	0,5	16,2	16,2	16,3	16,2	16,2	16,1	16,1
	80	0,5	22,5	22,6	22,5	22,6	22,4	22,4	22,4
	100	0,5	<b>27,8</b>	28,4	28,1	28,0	27,9	27,8	27,8
9	20	0,3	<b>14,3</b>	14,3	14,3	14,3	14,3	14,3	14,3
	40	0,4	<b>27,8</b>	27,8	27,8	27,8	27,8	27,8	27,8
	60	0,3	<b>43,7</b>	43,8	43,7	43,7	43,7	43,7	43,7
	80	0,4	<b>57,7</b>	57,7	57,7	57,7	57,7	57,7	57,7
	100	0,3	<b>69,5</b>	69,5	69,5	69,5	69,5	69,5	69,5
10	20	1,2	<b>4,2</b>	4,3	4,2	4,3	4,2	4,2	4,2
	40	0,1	<b>7,4</b>	7,5	7,4	7,4	7,4	7,4	7,4
	60	0,3	10,2	10,4	10,2	10,2	10,1	10,0	10,0
	80	0,3	13,0	13,0	13,0	13,0	12,8	12,9	12,8
	100	0,2	15,9	16,6	16,2	16,2	15,9	15,9	15,8
(1 – 10)			7251	7360	7284	7275	7243	7241	7234

\*Valores em negrito correspondem a soluções equivalentes às melhores

### 5.4.3 Biased Random-Key Genetic Algorithm

A versão utilizada nesta seção é o BRKGA/VCD, que é uma extensão do método proposto em (ZUDIO et al., 2017b). O BRKGA/VCD é detalhado na Seção 4.10, onde os valores utilizados para os parâmetros calibráveis são os mesmos do BRKGA proposto em (GONÇALVES; RESENDE, 2013). A finalidade é efetuar uma comparação direta com os resultados reportados no último trabalho citado, adotando o mesmo o critério de parada. O parâmetro da taxa de escolha do VCD foi especificado como 0,0, para o VCD sempre reproduzir com novas *random-keys* durante todo o processo da meta-heurística. A Tabela 5.10 sumariza as configurações utilizadas para os componentes do BRKGA/VCD desta seção.

Tabela 5.10: Configuração dos componentes do BRKGA/VCD.

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	$\rho = 0, 7$
Taxa de escolha do VCD	$\alpha = 0, 0$

O BRKGA/VCD executou 10 vezes para cada instância com sementes no intervalo de  $[1, 10]$  para o gerador de números pseudoaleatórios. Assim, todos os resultados desta seção são a média dessas execuções. A Tabela 5.11 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) para as mesmas instâncias.

Tabela 5.11: 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 e do BRKGA/VCD 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 significativamente 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 5.12 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 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 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 5.11 mostra que o método proposto neste trabalho é capaz de gerar soluções melhores que o reportado pelo BRKGA com somente  $\frac{1}{4}$  das iterações feitas.

A Tabela 5.13 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

Tabela 5.12: Resultado preliminar do BRKGA/VCD para o 3BP sem rotação.

Cl	$n$	$200g(s)$	$Z$	MVP	TS3	GLS	GASP	EHGH2	GVND	BRKGA
1	50	12	<b>13,4</b>	13,6	<b>13,4</b>	<b>13,4</b>	<b>13,4</b>	13,8	<b>13,4</b>	<b>13,4</b>
	100	65	<b>26,6</b>	27,4	<b>26,6</b>	26,7	26,9	27,6	<b>26,6</b>	<b>26,6</b>
	150	164	<b>36,3</b>	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	<b>50,6</b>	50,9	50,8
2	50	12	<b>13,8</b>	13,9	<b>13,8</b>	—	—	—	<b>13,8</b>	<b>13,8</b>
	100	66	<b>25,5</b>	26,4	25,7	—	—	—	25,7	25,6
	150	165	<b>36,6</b>	38,1	37,2	—	—	—	36,9	<b>36,6</b>
	200	320	<b>49,3</b>	50,2	50,1	—	—	—	49,4	49,4
3	50	12	<b>13,3</b>	13,6	<b>13,3</b>	—	—	—	<b>13,3</b>	<b>13,3</b>
	100	64	<b>25,9</b>	27,1	26,0	—	—	—	26,0	<b>25,9</b>
	150	171	<b>37,5</b>	39,2	37,7	—	—	—	37,6	<b>37,5</b>
	200	331	<b>49,8</b>	51,0	50,5	—	—	—	50,0	<b>49,8</b>
4	50	11	<b>29,4</b>	<b>29,4</b>	<b>29,4</b>	<b>29,4</b>	<b>29,4</b>	<b>29,4</b>	<b>29,4</b>	<b>29,4</b>
	100	61	<b>58,9</b>	59,2	59,0	59,0	59,0	59,5	59,0	59,0
	150	159	<b>86,8</b>	87,5	<b>86,8</b>	<b>86,8</b>	<b>86,8</b>	90,4	<b>86,8</b>	<b>86,8</b>
	200	318	<b>118,8</b>	119,0	<b>118,8</b>	119,0	<b>118,8</b>	119,0	<b>118,8</b>	<b>118,8</b>
5	50	30	<b>8,3</b>	9,2	8,4	<b>8,3</b>	8,4	7,9	<b>8,3</b>	<b>8,3</b>
	100	121	<b>15,0</b>	17,1	<b>15,0</b>	15,1	15,1	14,6	<b>15,0</b>	<b>15,0</b>
	150	278	<b>19,9</b>	24,0	20,4	20,2	20,6	21,5	20,1	20,0
	200	531	<b>27,1</b>	28,2	27,6	27,2	27,7	29,6	<b>27,1</b>	<b>27,1</b>
6	50	9	<b>9,7</b>	9,8	9,9	9,8	9,9	11,8	9,8	<b>9,7</b>
	100	47	<b>18,9</b>	19,3	19,1	19,1	19,1	19,2	19,0	<b>18,9</b>
	150	127	<b>29,0</b>	30,3	29,4	29,4	29,5	29,8	29,2	<b>29,0</b>
	200	257	<b>37,2</b>	37,8	37,7	37,7	38,0	38,7	37,4	37,3
7	50	24	<b>7,4</b>	7,9	7,5	<b>7,4</b>	7,5	<b>7,4</b>	<b>7,4</b>	<b>7,4</b>
	100	99	<b>12,2</b>	15,5	12,5	12,3	12,7	13,5	12,5	<b>12,2</b>
	150	228	<b>15,2</b>	19,9	16,1	15,8	16,6	18,2	16,0	15,3
	200	428	<b>23,4</b>	24,2	23,9	23,5	24,2	24,1	23,5	<b>23,4</b>
8	50	27	<b>9,2</b>	9,5	9,3	<b>9,2</b>	9,3	9,4	<b>9,2</b>	<b>9,2</b>
	100	122	<b>18,8</b>	21,4	18,9	18,9	19,0	18,9	18,9	18,9
	150	232	<b>23,6</b>	27,5	24,1	23,9	24,8	26,0	24,1	<b>23,6</b>
	200	451	<b>29,3</b>	31,1	30,3	29,9	31,1	35,8	29,8	<b>29,3</b>
(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

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

Tabela 5.13: Resultado preliminar do BRKGA/VCD para o 2BP sem rotação.

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

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

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. O teste diz que os resultados da tabela não provê uma diferença estatisticamente significativa. Porém, a Tabela 5.11 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. Apesar do VCD introduzir um novo parâmetro calibrável nos testes computacionais deste trabalho, o valor de  $\alpha$  foi 0. Isto significa que este parâmetro não foi utilizado e a escolha para a reprodução foi sempre aleatória. 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.

## Capítulo 6

### Conclusão

Este trabalho apresentou uma proposta de dissertação com três heurísticas baseadas nas meta-heurísticas GRASP, Busca Tabu e BRKGA para a resolução do problema de empacotamento tridimensional e bidimensional. Cada método proposto é composto por diversos componentes que podem ser combinados em diferentes versões para cada algoritmo. Um desses componentes é uma heurística construtiva que encontra soluções desde o princípio e que foi estendida para resolver o problema considerando rotação. Além disso, os componentes podem ser facilmente implementados para resolver tanto a versão clássica quanto para solucionar uma versão com restrições físicas adicionais. Extensivos testes computacionais sintetizaram uma base de resultados preliminares com uma versão promissora de cada método. Tais resultados preliminares mostraram que os algoritmos propostos que utilizam o GRASP e a Busca Tabu obtêm resultados equivalentes a outros métodos estado da arte da literatura, que por sua vez utilizam procedimentos e estruturas complexas de serem implementadas ou diversos parâmetros calibráveis. Os dados preliminares também mostram que a heurística proposta baseada no BRKGA testada foi capaz de obter soluções melhores ou equivalentes para os mesmos métodos comparados da literatura, construindo um novo conjunto de melhores resultados conhecidos para 99% das instâncias testadas. Finalmente, este trabalho também analisou a complexidade de cada componente, a fim de melhor descrever o seu comportamento prático e o seu processo de implementação.

Cada algoritmo proposto foi continuamente desenvolvido ao longo deste ano. Inicialmente, o trabalho (ZUDIO et al., 2017a) introduziu o algoritmo GRASP/VND, e foi apresentado no SBPO 2017. Este documento estendeu este algoritmo com novos componentes, por exemplo, o DFTRC e o FOA. Tais componentes não foram extensivamente testados, assim como diversos componentes do BT/VN. Posteriormente, o trabalho (ZUDIO et al., 2017b), 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. Entretanto, componentes como o Best EM e o Best Fit não foram extensivamente testados em conjunto com o BRKGA/VCD e o BRKGA. As-



sim, como proposta final para esta dissertação, objetiva-se determinar a melhor versão de cada algoritmo proposto através de testes computacionais com todas as combinações possíveis de componentes para cada uma no mesmo âmbito computacional utilizado neste trabalho. As melhores versões devem ser comparadas entre si e executadas novamente com o mesmo conjunto de instâncias para uma nova comparação com a literatura. Além disso, pretende-se submeter estas heurísticas para testes do 3BP e do 2BP com rotação.

## Referências

- BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, INFORMS, v. 6, n. 2, p. 154–160, 1994.
- BERKEY, J. O.; WANG, P. Y. Two-dimensional finite bin-packing algorithms. *Journal of the operational research society*, Springer, v. 38, n. 5, p. 423–429, 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, v. 53, n. 4, p. 735–736, 2005.
- BOSCHETTI, M. A. New lower bounds for the three-dimensional finite bin packing problem. *Discrete Applied Mathematics*, Elsevier, v. 140, n. 1, p. 241–258, 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, v. 74, n. 1, p. 507–527, 2016.
- CAPOROSSI, G.; HANSEN, P.; MLADENOVIC, N. Variable neighborhood search. In: *Metaheuristics*. [S.l.]: Springer, 2016. p. 77–98.
- COELHO, I. M. et al. Optframe: a computational framework for combinatorial optimization problems. In: *Agra, Agostinho and Doostmohammadi, Mahdi (2011) A Polyhedral Study of Mixed 0-1 Set. In: Proceedings of the 7th ALIO/EURO Workshop. ALIO-EURO 2011, Porto, pp. 57-59*. [S.l.: s.n.], 2011. p. 51.
- CRAINIC, T.; PERBOLI, G.; TADEI, R. *A greedy adaptive search procedure for multi-dimensional multi-container packing problems*. [S.l.]: CIRRELT, 2012. 1–21 p. <https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2012-10.pdf>.
- CRAINIC, T. G.; PERBOLI, G.; TADEI, R. Ts 2 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, 2009.
- DELL’AMICO, M.; MARTELLO, S.; VIGO, D. A lower bound for the non-oriented two-dimensional bin packing problem. *Discrete Applied Mathematics*, Elsevier, v. 118, n. 1, p. 13–24, 2002.

- FAROE, O.; PISINGER, D.; ZACHARIASEN, M. Guided local search for the three-dimensional bin-packing problem. *Inform's journal on computing*, INFORMS, v. 15, n. 3, p. 267–283, 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, 2001.
- FEKETE, S. P.; SCHEPERS, J. A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research*, INFORMS, v. 29, n. 2, p. 353–368, 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, 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, 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, 2007.
- GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979. ISBN 0716710447.
- GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, Elsevier, v. 13, n. 5, p. 533–549, 1986.
- GLOVER, F.; LAGUNA, M. Tabu search\*. In: *Handbook of Combinatorial Optimization*. [S.l.]: Springer, 2013. p. 3261–3362.
- GLOVER, F. W.; KOCHENBERGER, G. A. *Handbook of metaheuristics*. [S.l.]: Springer Science & Business Media, 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, 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, p. 993–1000, 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, 2014.
- HOLLAND, J. H. Adaptation in natural and artificial systems. an introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI: University of Michigan Press*, 1975.

- 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, v. 219, n. 3, p. 1287–1299, 2012.
- KARP, R. M. Reducibility among combinatorial problems. In: *Complexity of computer computations*. [S.l.]: 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: INSTITUTE OF INDUSTRIAL ENGINEERS-PUBLISHER. *IIE Annual Conference. Proceedings*. [S.l.], 2014. p. 2039.
- 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, 2013.
- 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, 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, 2002.
- LODI, A.; MARTELLO, S.; VIGO, D. Tspack: a unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research*, Springer, v. 131, n. 1-4, p. 203–213, 2004.
- 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, 2012.
- MARTELLO, S.; PISINGER, D.; VIGO, D. The three-dimensional bin packing problem. *Operations Research*, INFORMS, v. 48, n. 2, p. 256–267, 2000.
- MARTELLO, S. et al. Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. *ACM Transactions on Mathematical Software (TOMS)*, ACM, v. 33, n. 1, p. 7, 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, 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, 1998.
- MONACI, M.; TOTH, P. A set-covering-based heuristic approach for bin-packing problems. *INFORMS Journal on Computing*, INFORMS, v. 18, n. 1, p. 71–85, 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, 2017.
- PARREÑO, F. et al. A hybrid grasp/vnd algorithm for two-and three-dimensional bin packing. *Annals of Operations Research*, Springer, v. 179, n. 1, p. 203–220, 2010.

- PARREÑO, F. et al. Neighborhood structures for the container loading problem: a vns implementation. *Journal of Heuristics*, Springer, v. 16, n. 1, p. 1–22, 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*. [S.l.]: Springer, 2016.
- 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, 2015.
- SZWARCFITER, J. L.; MARKENZON, L. *Estruturas de Dados e seus Algoritmos*. 3. ed. [S.l.]: LTC, 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, 2007.
- ZUDIO, A. et al. Algoritmo grasp/vnd para o problema clássico de empacotamento tridimensional e bidimensional. *XLIX Simpósio Brasileiro de Pesquisa Operacional*, Proceedings of SBPO 2017, 2017.
- ZUDIO, A. et al. Brkga/vnd hybrid algorithm for the classic three-dimensional bin packing problem. *5th International Conference on Variable Neighborhood Search*, Proceedings of VNS 2017, 2017.