UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

RAFAEL COELHO MONTE ALTO
Orientador: Prof. Dr. Puca Huachi Vaz Penna

SIMULATED ANNEALING APLICADO AO PROBLEMA DE BIN PACKING COM CONFLITOS

Ouro Preto, MG
2025

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

RAFAEL COELHO MONTE ALTO

SIMULATED ANNEALING APLICADO AO PROBLEMA DE BIN PACKING COM CONFLITOS

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Puca Huachi Vaz Penna

Ouro Preto, MG
2025

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Rafael Coelho Monte Alto

Simulated Annealing aplicado ao problema de Bin Packing com Conflitos

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 3 de Abril de 2025.

Membros da banca

Puca Huachi Vaz Penna (Orientador) - Doutor - Universidade Federal de Ouro Preto
Pablo Luiz Araújo Munhoz (Examinador) - Doutor - Universidade Federal de Ouro Preto
Bárbara Letícia Rodrigues Milagres (Examinadora) - - Universidade Federal de Ouro Preto

Puca Huachi Vaz Penna, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 3/04/2025.

Documento assinado eletronicamente por **Puca Huachi Vaz Penna**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 08/04/2025, às 10:49, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0886298** e o código CRC **90D42847**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Agradecimentos

Aos meus pais Ida Coelho e Rômulo Monte Alto por serem minhas inspirações e por serem meus maiores apoiadores. Aos meus irmãos Vanessa e Rodrigo, obrigado por estarem sempre presentes.

Ao Gabriel, meu companheiro de todos os dias, obrigado por me fazer ser uma pessoa melhor todos os dias e me apoiar no desenvolvimento deste estudo.

Aos amigos do Limonada e aos amigos do 2019/1 de computação, admiro muito vocês.

Ao meu orientador Puca, obrigado pelos ensinamentos. Ao DECOM e à UFOP, pelo ensino gratuito e de qualidade.

À minha amada República OxiGênios, todos os ex-alunos e moradores, por me fazerem ser quem eu sou hoje.

Resumo

Este trabalho aborda o Problema de Empacotamento com Conflitos, ou *Bin Packing Problem with Conflicts* (BPPC), derivado do clássico Problema de Empacotamento, do inglês *Bin Packing Problem* (BPP). No problema original, uma série de itens devem ser empacotados no menor número de *bins* possíveis, respeitando as restrições de capacidade. O BPPC possui o mesmo objetivo do BPP, porém lidando com itens que apresentam conflito entre si e, portanto, não podem ser empacotados no mesmo *bin*. Esta classe de problemas possui aplicações em contextos de logística, transporte, alocação de recursos, computação paralela e nas indústrias de corte e empacotamento. Devido a característica combinatorial do problema, o método *Simulated Annealing* e quatro estruturas de vizinhanças são propostas para resolver o BPPC. O algoritmo é testado em instâncias de teste da literatura e o seu desempenho é satisfatório, no geral. A solução ótima foi encontrada em 17,5% das instâncias totais.

Palavras-chave: Otimização Combinatória. Bin Packing. Bin Packing com Conflitos. Heurísticas. Meta-heurísticas. Simulated Annealing.

Abstract

This paper tackles the combinatorial optimization problem called the Bin Packing Problem with Conflicts (BPPC), derived from the classic Bin Packing Problem. In the original problem, a set of items is to be packed in the minimum number of same size containers, respecting its maximum weight capacity. The BPPC has the same objective but presents the case where items may have conflicts with each other and cannot be packed in the same bin together. This class of problems has real world applications in logistics, transportation, resource allocation, parallel computing and in the cutting and packing industry. A Simulated Annealing method with a random initial solution and four neighborhood moves is implemented. The proposed algorithm is tested on instances from the literature, and its performance is satisfactory. Optimal solutions were reached in 17.5% of cases.

Keywords: Combinatorial Optimization, Bin Packing, Bin Packing with Conflicts. Heuristics. Meta-heuristics. Simulated Annealing.

Lista de Ilustrações

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviaturas e Siglas

1D-BPP	*One Dimensional Bin Packing Problem*
AWF	*Almost Worst Fit*
BF	*Best Fit*
BPP	*Bin Packing Problem*
BPPC	*Bin Packing Problem with Conflicts*
BKS	*Best Known Solution*
CSP	*Cutting Stock Problem*
FF	*First Fit*
MFFD	*Modified First Fit Decreasing*
NF	*Next Fit*
SA	*Simulated Annealing*
WF	*Worst Fit*

Sumário

1 Introdução

O Problema de Empacotamento, do inglês *Bin Packing Problem* (BPP), é um problema clássico de otimização combinatória. O objetivo do problema é empacotar uma série de objetos em caixas (*bins*) de um tamanho fixo, minimizando o número de caixas utilizadas. Com trabalhos datando desde a década de 1970 (EILON; CHRISTOFIDES, 1971; GAREY; GRAHAM; ULLMAN, 1972; JOHNSON, 1974), o BPP é bastante estudado e discutido na literatura.

O BPP faz parte de um domínio de pesquisa que recebe atenção por sua aplicação em áreas de empacotamento, corte, agendamento, alocação de recursos, gestão de cadeia de abastecimento, etc. Por exemplo, Rhiat *et al.* (2020) descrevem a abordagem de um sistema de empacotamento de produtos de um *e-commerce* a serem entregues por um robô móvel. É levado em consideração as características de peso, volume, fragilidade, capacidade de empilhamento e orientação dos itens, minimizando o número de caixas necessárias para a entrega.

Outras aplicações do BPP envolvem o agendamento de tarefas independentes a processadores (COFFMAN JR; GAREY; JOHNSON, 1978), alocação de comerciais a intervalos de televisão (BROWN, 1971), roteamento de veículos capacitados (LABBÉ; LAPORTE; MERCURE, 1991), dentre outros.

O BPP possui um grande número de variantes, com alterações em restrições e aspectos físicos do problema. Sua definição clássica é conhecida como BPP unidimensional (1D-BPP). Existem, também, as variantes do BPP bidimensional (2D-BPP) (LODI et al., 2014) e tridimensional (3D-BPP) (MARTELLO; PISINGER; VIGO, 2000). No 2D-BPP, uma série de itens retangulares devem ser empacotados em *bins* idênticos também retangulares, de largura e altura fixos. O 3D-BPP envolve itens com medidas de largura, altura e profundidade. A Figura 1.1 apresenta um exemplo do 3D-BPP que utiliza dois *bins* para acomodar itens.

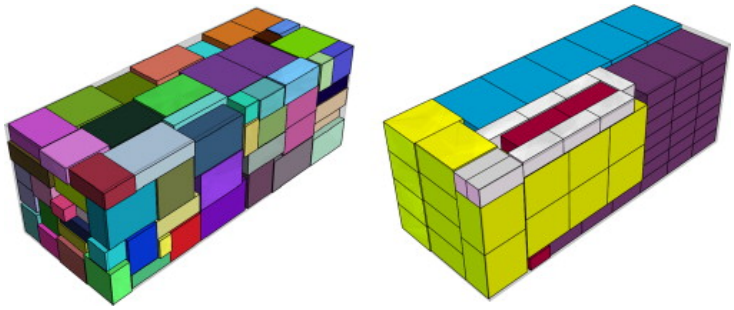

Figura 1.1 – *Bin Packing* tridimensional. Fonte: Gonçalves e Resende (2013)

Outras variantes derivam da versão clássica do problema quando tratamos com *bins* de custo ou tamanho variados (SEIDEN; STEE; EPSTEIN, 2003; BALDI et al., 2012), fragmentação de itens (MENAKERMAN; ROM, 2001) ou propriedades de fragilidade ou compatibilidade

(BANSAL; LIU; SANKAR, 2009). Para mais informações sobre variantes do BPP, referenciamos Coffman *et al.* (2013).

O 1D-BPP trabalha com a otimização do empacotamento de itens levando em consideração apenas uma dimensão, ou diretriz. Essa medida pode ser o peso de um item, seu tamanho, volume, comprimento, etc. Neste trabalho, vamos nos referenciar sempre ao peso de um item. A Figura 1.2 ilustra o processo de *Bin Packing* clássico unidimensional, com cinco itens e dois exemplos de soluções para o mesmo problema. Na primeira solução, três *bins* acomodam os itens, enquanto que na segunda, são utilizados dois *bins*. A dimensão utilizada nos exemplos é representada por quadrados e os *bins* têm capacidade máxima de 10 quadrados.

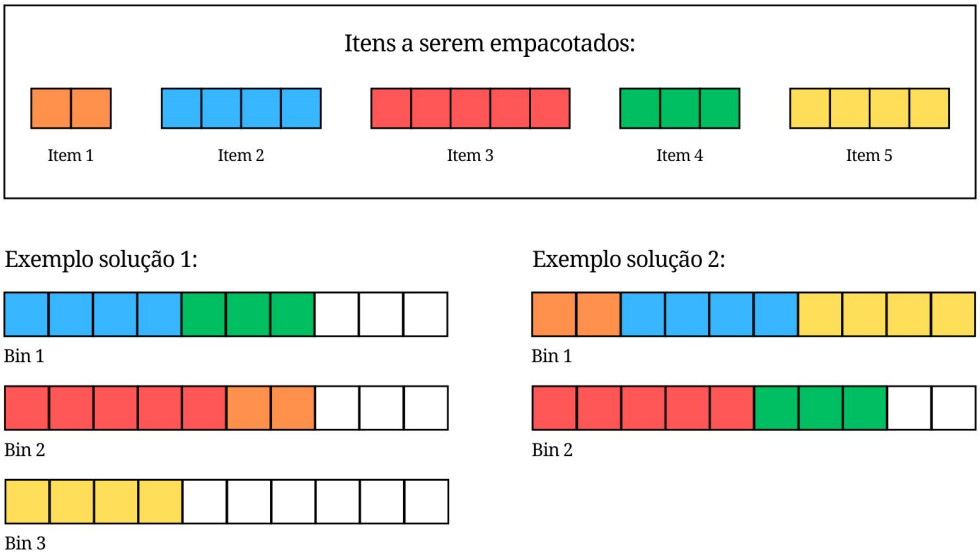

Figura 1.2 – Processo de *Bin Packing* unidimensional.

Este estudo contextualiza o *Bin Packing* clássico e foca em sua variante conhecida como Problema de *Bin Packing* com Conflitos, do inglês *Bin Packing Problem with Conflicts* (BPPC). Diferente do problema clássico BPP, essa variante define itens que são conflitantes e, portanto, não podem estar presentes em um mesmo *bin*. Esse problema possui aplicações em diversas áreas como a de transporte e logística, onde produtos podem não ser compatíveis. Um exemplo seria o transporte de produtos inflamáveis, tóxicos e explosivos, que não devem ser carregados juntos em uma mesma carga (HAMDI-DHAOUI; LABADIE; YALAOUI, 2014).

Um problema que pode ser resolvido traçando uma analogia ao BPPC envolve o desafio de alocar tarefas a um sistema multiprocessado, sendo que alguns processos não devem ser executados pelo mesmo processador por motivos de sobrecarga de CPU, visando designar um número mínimo de processadores ao grupo de tarefas (JANSEN, 1999). O autor menciona que o BPPC também pode ser aplicado em procedimentos de armazenamento onde, para seguir uma

estratégia de tolerância a falhas, é interessante manter cópias de um mesmo arquivo ou banco de dados em servidores diferentes.

Irani e Leung (1996) estudam o problema de execução e agendamento de trabalhos que competem por recursos limitados. Tarefas que necessitam de um mesmo recurso não poderão ser executadas ao mesmo tempo e, portanto, serão conflitantes. Os autores aplicam seus estudos ao problema de controle de interseção de tráfico.

O BPP, assim como o BPPC, é um problema classificado como NP-difícil (GAREY; JOHNSON, 1979; JANSEN; ÖHRING, 1997) e, portanto, não existe, até então, um algoritmo que o resolva em tempo polinomial. Mesmo assim, existem diversos estudos sobre a aplicação de métodos exatos para o BPP e o BPPC (MURITIBA et al., 2010; SADYKOV; VANDERBECK, 2013; DELORME; IORI; MARTELLO, 2016; WEI et al., 2020). Procurando alternativas para os métodos exatos, o foco de muitos pesquisadores é desenvolver e aprimorar métodos aproximados para resolução do problema. Estes métodos não garantem que chegarão na melhor solução, mas encontram uma solução considerada satisfatória, ou quase ótima, em tempo computacionalmente aceitável.

O BPP é comparado muitas vezes na literatura com o Problema de Corte de Estoque, do inglês *Cutting Stock Problem* (CSP), que envolve o desafio de cortar pedaços de certos tamanhos de uma peça padrão, visando diminuir o desperdício de matéria-prima (DELORME; IORI; MARTELLO, 2016). Materiais como cabos, madeira, canos, tapetes, dentre outros. são fornecidos em um comprimento padrão C. Pedidos são feitos de material em medidas de comprimento que não ultrapassam C. O problema envolve usar a menor quantidade de peças padrão para acomodar uma lista de demandas (JR; GAREY; JOHNSON, 1984). Os BPP e o CSP são problemas comumente agrupados em uma só classe na literatura, chamada *Cutting/Packing Problems*, ou Problemas de Corte/Empacotamento.

1.1 Justificativa

A classe de problemas BPP é um domínio de pesquisa que recebe muito reconhecimento e cobertura por sua relevância em aplicações industriais e em diversas áreas de logística em transportes, inventário, cortes nas indústrias de vidro e madeira, alocação de recursos, computação paralela, entre outras (JANSEN; ÖHRING, 1997; HAMDI-DHAOUI; LABADIE; YALAOUI, 2014). Avanços em métodos de resolução garantem soluções mais próximas da otimalidade e possibilitam uma diminuição de custos expressivos para empresas especializadas em serviços que envolvem algum tipo de alocação de itens a caixas ou cargas, visando minimizar o número de recursos utilizados.

Impactos ambientais também podem ser obtidos com resoluções de problemas modelados como o BPP . Baker *et al.* (2018) por exemplo, aplica técnicas de *Bin Packing* para minimizar os danos ecológicos causados pelo consumo excessivo de energia da computação em nuvem e do

aumento exponencial de produção de dados e tráfego de rede.

1.2 Objetivos

Este estudo tem como objetivo geral avaliar abordagens aproximadas para tratar do BPP e do BPPC. Os objetivos específicos são:

1. Trazer uma contextualização do BPP e do BPPC, a evolução histórica da pesquisa em problemas de empacotamento e corte, as principais abordagens utilizadas na literatura e aplicações populares na indústria.
2. Implementação de um método meta-heurístico *Simulated Annealing* para resolver o BPPC.
3. Realização de experimentos computacionais para avaliar o desempenho do método proposto utilizando instâncias de teste populares da literatura.
4. Comparação do desempenho do algoritmo com métodos construtivos guloso e randômico.

1.3 Organização do Trabalho

O restante deste estudo está dividido como segue. O Capítulo 2 apresenta uma revisão da literatura e um contexto histórico do problema abordado. O Capítulo 3 expõe o desenvolvimento do trabalho e a metodologia empregada enquanto que o Capítulo 4.3 exibe os resultados obtidos. Finalmente, o Capítulo 5 conclui o estudo e discute as possibilidades de continuação do trabalho.

2 Revisão Bibliográfica

O objetivo deste capítulo é apresentar um panorama geral do BPP e do BPPC, conceitos teóricos importantes e a evolução histórica de métodos e resultados. São analisados diversos trabalhos de pesquisadores que abordaram o problema e suas variantes.

2.1 Fundamentação Teórica

Esta seção apresenta a fundamentação teórica base necessária para se discutir o BPP e problemas derivados. Na Seção 2.1.1 é apresentada uma formulação matemática para o BPP e o BPPC. A Seção 2.1.2 apresenta algumas heurísticas clássicas utilizadas comumente na literatura. Na Seção 2.1.4 é mencionado o papel de limites inferiores para o algoritmos do BPP e é definido um limite inferior trivial do problema.

2.1.1 Formulação Matemática

No BPP, é fornecida uma quantidade infinita de *bins* B de capacidade fixa C e uma lista com n itens, cada um com um peso w_i $(i = 1, ..., n)$. O problema envolve designar itens a *bins* de forma que a soma dos pesos dos itens em um *bin* seja menor ou igual a sua capacidade C e o número de *bins* utilizados é mínimo. Em sua definição normalizada equivalente, os pesos dos itens são números reais no intervalo $[0, 1]$ e a capacidade dos *bins* é igual a 1 (DELORME; IORI; MARTELLO, 2016).

O BPP pode ser formulado como um modelo de Programação Linear Inteiro (PLI). A seguinte formulação foi baseada no trabalho de Gendreau *et al.* (2004), onde a variável y_k assumirá o valor 1 se o *bin* k contiver algum item e 0 se não estiver sendo utilizado. x_{ik} vai possuir o valor 1 se o item i estiver presente no *bin* k. Caso contrário, vai assumir o valor 0.

$$\min \quad z = \sum_{k=1}^{n} y_k \tag{2.1}$$

$$\text{s.a} \quad \sum_{i=1}^{n} w_i x_{ik} \leq C y_k \quad (k = 1, \ldots, n) \tag{2.2}$$

$$\sum_{k=1}^{n} x_{ik} = 1 \quad (i = 1, \ldots, n) \tag{2.3}$$

$$y_k = 0 \text{ ou } 1 \quad (k = 1, \ldots, n) \tag{2.4}$$

$$x_{ik} = 0 \text{ ou } 1 \quad (i, k = 1, \ldots, n) \tag{2.5}$$

A função objetivo (2.1) determina a minimização da quantidade de *bins* utilizados no processo de empacotamento. As restrições (2.2) garantem que, somando o peso de todos os itens de um *bin*, o valor total não ultrapassará sua capacidade máxima C. O conjunto de restrições (2.3) garantirá que um item estará associado a apenas um *bin*. As restrições (2.4) e (2.5) definem o universo binário das variáveis de decisão.

O BPPC recebe os parâmetros de entrada do BPP juntamente com um grafo de conflito $G = (V, E)$. Cada vértice em V representa um item a ser empacotado e existe uma aresta (i, j) em E se os itens i e j são conflitantes entre si.

O BPPC pode ser formulado como (GENDREAU; LAPORTE; SEMET, 2004):

$$\min \quad z = \sum_{k=1}^{n} y_k \tag{2.6}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_{ik} \leq C y_k \quad (k = 1, \ldots, n) \tag{2.7}$$

$$\sum_{k=1}^{n} x_{ik} = 1 \quad (i = 1, \ldots, n) \tag{2.8}$$

$$x_{ik} + x_{jk} \leq 1 \quad ((i, j) \in E, k = 1, \ldots, n) \tag{2.9}$$

$$y_k = 0 \text{ ou } 1 \quad (k = 1, \ldots, n) \tag{2.10}$$

$$x_{ik} = 0 \text{ ou } 1 \quad (i, k = 1, \ldots, n) \tag{2.11}$$

A função objetivo (2.6) assegura a minimização do número de *bins* utilizados no empacotamento. As restrições (2.7) irão garantir que a soma dos pesos de todos os itens presentes em um *bin* não irá ultrapassar sua capacidade. As restrições (2.8) determinam que cada item deve estar presente em um único *bin*. O conjunto de restrições (2.9) garantem que itens conflitantes não estarão presentes em um mesmo *bin*. Finalmente, as restrições indicadas em (2.10) e (2.11) dizem respeito ao domínio das variáveis de decisão.

Nota-se que o BPPC é uma generalização do BPP quando não existe nenhum conflito entre os itens ($E = \emptyset$) e do Problema de Coloração de Grafos, ou *Vertex Coloring Problem* (VCP), quando os pesos dos itens são desconsiderados ($\forall i, w_i = 0$) (JANSEN; ÖHRING, 1997). O VCP trata o problema de colorir os vértices de um grafo sendo que vértices adjacentes não podem ter a mesma cor, visando minimizar o número de cores diferentes utilizadas (GALINIER et al., 2013).

2.1.2 Heurísticas

Como visto anteriormente, diversos pesquisadores da literatura procuram resolver casos do BPP com abordagens exatas apesar da classificação NP-difícil do problema. No entanto, tais

métodos não são eficientes para problemas com dimensões elevadas. Neste sentido, métodos heurísticos também possuem bastante foco e trazem resultados muito satisfatórios.

"Uma heurística é uma técnica computacional aproximativa que visa alcançar uma solução avaliada como aceitável para um dado problema que pode ser representado em um computador, utilizando um esforço computacional considerado razoável, [...]" (GOLDBARG; GOLDBARG; LUNA, 2017).

Algumas heurísticas clássicas são discutidas desde a época em que se iniciaram os estudos sobre o BPP. Uma dessas heurísticas, conhecida como *Next Fit* (NF), segue o seguinte procedimento: após empacotar o primeiro item, cada item sucessivo é designado ao *bin* do último item que foi guardado, se este couber. Se a adição do item fizer com que a capacidade da carga seja ultrapassada, NF fecha o *bin* e empacota o item no próximo *bin* (COFFMAN et al., 2013).

Outro método clássico para o BPP é a heurística *First Fit* (FF) . Ela difere da última mencionada, pois não fecha nenhum *bin*. Ao empacotar cada item, verifica-se se ele cabe nos *bins* abertos até o momento. O item será empacotado no primeiro *bin* em que couber (HALL et al., 1988).

Uma heurística também bastante utilizada na literatura é chamada de *Best Fit* (BF). O seu propósito é tentar encontrar o melhor encaixe para um item dentre todos os *bins* abertos. O BF segue os seguintes passos: se não existe nenhum *bin* com capacidade para acomodar o item atual i, BF empacota i em um *bin* vazio. Caso contrário, o algoritmo adiciona o item ao *bin* que, após a alocação, resulte no menor espaço residual entre os *bins* abertos até o momento. Havendo mais de um *bin* para escolher, o item é adicionado ao *bin* de menor índice.

Oposto ao *Best Fit*, a heurística *Worst Fit (WF)* procura pelo *"pior"* encaixe de um item. WF irá alocar um item i ao *bin* aberto de maior espaço residual. Se não existir *bin* aberto que comporte a capacidade de i, um novo *bin* é aberto. Com uma pequena alteração no seu procedimento, o WF se torna *Almost Worst Fit* (AWF) quando um item é alocado no *bin* com **segundo** maior espaço residual. Se não é possível, o algoritmo tenta alocar o item no *bin* mais vazio. Se o item não for alocado em nenhuma das duas tentativas anteriores, um novo *bin* é aberto.

Johnson (1974) menciona a melhora substancial no pior caso do algoritmo AWF quando essa mudança é aplicada ao WF. De acordo com Garey, Graham e Ullman (1979), o BF e o FF são os dois algoritmos de melhor desempenho, quando comparamos seus piores casos com o NF e o WF.

David S. Johnson (1974) fala sobre a possibilidade de se trabalhar, no FF, com estruturas de árvores ordenadas, onde existem n nós para n itens e os nós folha guardam o espaço residual de cada *bin*. Os nós internos guardam o maior valor de espaço residual dentre seus dois filhos. Assim, com menos de $\log_2 n$ comparações, é possível encontrar onde um item deve ser alocado no *First Fit*. Algoritmos como o *Best Fit*, *Worst Fit* e *Almost Worst Fit* podem ser implementados

em árvores binárias balanceadas (AVL), que conseguem buscar o *bin* apropriado para um item em tempo $O(\log n)$. Agora o FF, que depende da ordem de *bins*, não pode se aproveitar de estruturas de árvores onde os *bins* serão reordenados.

Os conceitos de algoritmo *on-line* e *off-line* tornam-se importantes com o avanço de heurísticas para resolução do BPP. Um algoritmo *on-line* empacota um item no momento que ele é inspecionado, sem utilizar de conhecimentos de itens subsequentes da lista. Os itens são empacotados na ordem que são dados. Opostamente, um algoritmo *off-line* permite o pré-processamento de todos os itens para ordenação, rearranjo ou agrupamento antes da rotina de empacotamento (COFFMAN et al., 2013).

As heurísticas NF, FF e BF mencionadas são implementadas em algoritmos *on-line*, mas possuem variações *off-line* conhecidas como *Next Fit Decreasing* (NFD), *First Fit Decreasing* (FFD) e *Best Fit Decreasing* (BFD). Esta classe de heurísticas executa um preprocessamento nos itens, ordenando-os em ordem decrescente de acordo com seus pesos antes de executar o NF, FF ou BF, respectivamente.

A rotina de ordenação dos itens para que os de maior peso sejam empacotados primeiro resulta em soluções que utilizam menos *bins* no geral. Simchi-Levi (1994) prova que a razão de desempenho absoluta, do inglês *absolute performance ratio* das heurísticas FF e BF não é maior que 1.75 e das heurísticas FFD e BFD não é maior que 1.5. Esse parâmetro representa o desvio máximo da solução heurística à otimalidade, dado todos os possíveis casos.

2.1.3 Simulated Annealing

O *Simulated Annealing* (SA) é um método proposto por Kirkpatrick, Gelett e Vecchi (1983) e Cerny (1985). O SA é uma meta-heurística que se baseia em uma analogia ao processo físico de recozimento de metais.

O recozimento, ou *annealing*, é utilizado na metalurgia para transformação das propriedades físicas de um sólido através de um tratamento térmico. Esse procedimento envolve aquecer um material acima de sua temperatura de recristalização, manter uma temperatura apropriada por um período de tempo e depois resfriar a substância. Durante o lento processo de recozimento, o material passa por vários estados e seus átomos são acomodados em um uma estrutura uniforme com energia mínima (AARTS; KORST; MICHIELS, 2005). A menor energia interna é buscada para que haja uma redução dos defeitos do produto final.

A meta-heurística trabalha como uma simulação do processo de recozimento. Os estados possíveis do material são representados pelas diferentes soluções para o problema de otimização combinatória. A energia interna de cada estado é o valor de custo da solução, ou valor da função objetivo. Por se tratar de um problema de minimização, o estado de menor energia do metal corresponde à solução ótima local. Um parâmetro de temperatura é introduzido para que o resfriamento ocorra controladamente.

Um detalhe importante acontece durante a busca pela solução de menor custo. Soluções vizinhas candidatas são geradas e aceitas dependendo de um parâmetro probabilístico. Existe a probabilidade de se aceitar soluções piores durante a passagem pelo espaço de busca, mas essa chance diminui com a redução da temperatura.

Um trabalho comumente referenciado na literatura é o estudo de Kampke (1988), onde o autor aborda o BPP. Dado uma solução inicial, gerada randômica ou seguindo alguma lógica (ex.: FF), parte-se para geração de uma solução vizinha. Determina-se dois grupos: *bins* cheios (*bins* que não estão com carga máxima. Seleciona-se um *bin* de cada grupo e um item de cada *bin*, aleatoriamente. Os dois itens são trocados. Agora começa o procedimento de *smoothing* para tratar os excessos de carga que a troca provavelmente causou. No processo de *smoothing*, o item de menor tamanho do *bin* problemático é selecionado e adicionado ao *bin* de menor carga da solução. Esse processo se repete até que uma solução válida seja gerada.

O SA é uma meta-heurística popular por sua implementação direta e baixo custo de memória. O Algoritmo 2.1 apresenta um método SA padrão.

Algoritmo 2.1: Pseudocódigo padrão de um Simulated Annealing

```
   Entrada: S, T_i,α, T_min, SA_max
   Saída: solução S*
1  S* ← S;
2  T ← T_i;
3  iter_T ← 0;
4  enquanto T > T_min faça
5      enquanto iter_T < SA_max faça
6          S' ← neighbor(S);
7          Δ ← fo(S') − fo(S);
8          se Δ < 0 então
9              S ← S';
10             se fo(S') < fo(S*) então
11                 S* ← S';
12         senão
13             x ← random[0, 1];
14             se x < e^{−Δ/T} então
15                 S ← S';
16         iter_T ← iter_T + 1;
17     T ← T ∗ α;
18     iter_T ← 0;
19 retorna S*;
```

No Algoritmo 2.1, a temperatura inicial, fator de resfriamento, temperatura mínima, e solução inicial são passadas como parâmetro de entrada do método, além de SA_{max}, que determina a quantidade de iterações a serem executadas em uma mesma temperatura. O laço principal (linha 4) é executado enquanto a temperatura atual é maior que a temperatura mínima.

O laço interior (linha 5) executa SA_{max} iterações em uma mesma temperatura. Uma solução vizinha S' é gerada na linha 6 e os valores de função objetivo das soluções vizinha e atual são calculados. Se a solução vizinha for melhor que a atual, ela é aceita (linhas 8 e 9). Verifica-se, na linha 11, se S' supera, em qualidade, a melhor solução avaliada S^*. Se esse for o caso, S^* é atualizada. Caso a solução vizinha seja pior que a atual, existe uma possibilidade de que esta seja aceita (linhas 13-15). Essa probabilidade é influenciada pela temperatura atual. Na linha 17 a temperatura é reduzida. A melhor solução entre todas as avaliadas é retornada na linha 19.

2.1.4 Limite Inferior

Em toda a literatura, a busca por *lower bounds*, ou limites inferiores para a classe de problemas BPP atraiu muita atenção. Muitos algoritmos utilizam de um limite inferior como ponto de referência ao avaliar soluções e interromper sua busca.

Falkenauer (1996) comenta sobre o ***Theoretical Miminum Number of Bins (theo)***, ou número mínimo teórico de *bins*, obtido através da soma do peso de todos os itens dividido pela capacidade C de um *bin*. A formula é apresentada na Equação (2.12). Esse é o menor número de *bins* necessários para acomodar todos os itens, para o BPP; portanto, uma solução que encontre *theo bins*, terá encontrado um ótimo global.

$$theo = \frac{\sum_{i=1}^{n} w_i}{C} \tag{2.12}$$

2.2 Trabalhos Relacionados

Esta seção apresenta, em 2.2.1, um conjunto de trabalhos que abordam heurísticas para o BPP desde a década de 1970, com os trabalhos de S. Eilon e N. Christofides (1971) e M. R. Garey, J. D. Ullman e R. L. Graham (1972) . A Seção 2.2.2 aborda estudos que tem como foco resolver o BPPC.

2.2.1 Bin Packing Problem

Em 1971, Eilon e Christofides descreveram e compararam um método exato com um método heurístico para resolver o problema de *Bin Packing*, descrito como *"Loading Problem"* pelos autores (EILON; CHRISTOFIDES, 1971). No método heurístico, são buscados itens que preencham a capacidade restante de *bins*, de forma a completar *bins* inteiros. Caso não sejam encontrados pares de itens e bins correspondentes, o item de maior magnitude (peso) é adicionado ao bin com menor capacidade disponível.

Os autores conduziram experimentos em instâncias de 50 itens e o método heurístico alcançou a otimalidade em 48 dos 50 casos de teste, todos em um tempo computacional inferior ao

método exato. Uma observação interessante é que os autores propuseram, no método heurístico, uma rotina de rearranjo ou embaralhamento da solução que não atingira um número mínimo ótimo de *bins*, suposto inicialmente com base em um limite inferior.

Fleszar e Hindi (2002) trabalharam com a heurística *Minimum Bin Slack (MBS)* de Gupta e Ho (1999) e propuseram algumas variações que reduziram o tempo de processamento. A cada passo, o algoritmo *MBS* procura por combinações de itens que resultem em um espaço residual mínimo nos *bins*, ou um espaço residual igual a zero, no melhor cenário. Itens não alocados a *bins* são guardados em uma lista Z' e são testadas todas as combinações de itens a fim de encontrar a melhor. Devido ao longo tempo computacional necessário, algumas modificações no algoritmo clássico são propostas, juntamente com quatro variações do *MBS*.

Os autores também estudaram diferentes vizinhanças no BPP para aplicar a meta-heurística *Variable Neighborhood Search* (VNS). Uma extensa análise computacional foi realizada para comparar o desempenho dos diferentes algoritmos, e a combinação mais eficaz foi a variante *Perturbation MBS'*, seguida pela aplicação do VNS.

Osogami e Okano (2003) analisaram diversas combinações de heurísticas construtivas (NF, FF e FFD) e buscas locais para resolver o BPP e o que eles chamaram de *real-world BPPs*. As buscas locais utilizadas nos experimentos foram *First Improvement* (FI), *Best Improvement* (BI) e duas propostas pelos autores, variações chamadas de *Prioritized Improvement* (PI) 1 e 2. Os autores também trabalharam com uma função objetivo diferente de (2.1) devido à dificuldade de se encontrar vizinhanças apropriadas e eficientes na redução do número de *bins* da solução. Eles se basearam no trabalho de Kampke (1988) e utilizaram a função objetivo que foca em equilibrar o conteúdo dos *bins*.

No FI, soluções vizinhas são geradas em alguma ordem (geralmente aleatória) e a primeira solução que superar a atual é escolhida. Já o BI avalia todas as soluções na vizinhança e escolhe a melhor. No *Primary Improvement*, uma ordem específica para gerar soluções na vizinhança é proposta e o algoritmo se move para a primeira solução vizinha encontrada que superar a solução atual.

O *Better Fit Heuristic* (BFH) é um exemplo de algoritmo que é *online* e *offline* ao mesmo tempo. O método remove um item de seu *bin* e troca com o item atual se este preencher melhor o *bin* (BHATIA; HAZRA; BASU, 2009). Este algoritmo supera o *Best Fit* (BF) *on-line* em todas as instâncias e se comporta como um *Best Fit Decreasing* (BFD) quando os itens estão em ordem decrescente. Sua melhor aplicação se dá quando os itens estão desordenados e possuem pesos distintos e distribuídos uniformemente.

Além de técnicas heurísticas, diversas técnicas meta-heurísticas foram estudadas e aplicadas no BPP. Estas possuem mecanismos que permitem escapar de ótimos locais do espaço de busca. Meta-heurísticas trabalham com uma série de operações como estruturas de vizinhança, buscas locais, recomeços, perturbações, memórias de curto e longo prazo, procedimentos de

intensificação e diversificação, dentre outros (CAPUA et al., 2018).

Um trabalho comumente referenciado na literatura é o estudo de Kampke (1988). O autor aplica a meta-heurística *Simmulated Annealing* (SA) (KIRKPATRICK; JR; VECCHI, 1983) para resolver o BPP.

Meta-heurísticas inspiradas em comportamentos da natureza ou que simulam fenômenos físicos, biológicos e sociais são comumente estudadas na literatura. Um exemplo é o trabalho de Zendaoui e Layeb (2016) que apresentaram uma versão adaptada do algoritmo de *Cuckoo Search via Levy Flights* (YANG; DEB, 2009) para resolver o 1D-BPP. Esta abordagem é baseada na estratégia agressiva de reprodução de aves da família Cuculídeos, que põem seus ovos em ninhos de outras espécies. As aves hospedeiras podem estranhar o ovo intruso e despejá-lo ou, então, se mudar e reconstruir o ninho em outro lugar. Porém, se a ave não notar o ovo de Cuco, este costuma se chocar antes dos demais ovos e, por instinto, se livrar do restante do ninho. Isso faz com que o Cuco ganhe uma maior porção de comida proveniente da ave hospedeira e aumenta suas chances de sobrevivência na natureza. O *Cuckoo Search* (CS) é uma meta-heurística populacional, similar a algoritmos genéticos. As soluções de melhores qualidades serão passadas para as próximas gerações, existindo a chance dos ovos serem notados pelas aves hospedeiras e estes serem abandonados.

No geral, meta-heurísticas são eficientes em resolver instâncias complexas do BPP (ABDEL-BASSET et al., 2018; EL-ASHMAWI; ELMINAAM, 2019) e demonstram um nível satisfatório em resultados computacionais (MUNIEN; EZUGWU, 2021).

Para uma revisão da literatura mais detalhada sobre o BPP, sugerimos os trabalhos de Coffman *et al.* (2013) e Garey e Johnson (1981).

2.2.2 Bin Packing com Conflitos

Jansen e Ohring (1997) abordaram o *Problema do Bin Packing com Conflitos* em seu trabalho sobre agendamento de trabalhos com restrição de tempo. O problema consiste em alocar uma série de trabalhos a diferentes máquinas, evitando que tarefas conflitantes sejam executadas por uma mesma máquina. Nota-se a semelhança com o desafio de alocar uma série de itens a diferentes caixas, tomando cuidado para não adicionar itens conflitantes a uma mesma caixa.

Gendreau *et al.* (2004) descreveram seis heurísticas para o BPPC. Uma delas é uma adaptação do *First Fit Decreasing (FFD)*, que segue o procedimento do *FFD* com uma verificação para identificar se há conflito em um *bin* ao empacotar um novo item. Outras três heurísticas se baseiam em um procedimento de coloração de grafos antes de aplicar o *FFD*. As duas últimas heurísticas propostas no trabalho funcionam com detecção de cliques nos grafos de conflito E e $\overline{E}$. Uma aresta (i, j) estará presente em $\overline{E}$ se i e j não são conflitantes e $w_i + w_j \leq C$.

A heurística que apresentou o melhor desempenho nos experimentos de Gendreau et. al (2004) foi o método que possui os seguintes passos: todos os itens que não apresentam conflito

com nenhum outro item são separados em um conjunto H. O algoritmo, então, encontra um clique D no grafo de conflito. Para cada item i em D, é computado um clique D_i no grafo de não-conflito. O procedimento *Modified First Fit (MFF)* é utilizado para resolver o BPP em cada um dos conjuntos D_i separadamente. Por fim, os itens em H são alocados nos *bins* parcialmente preenchidos, ou, se for preciso, em novos.

O *MFF* é uma heurística que segue os passos do FF porém, antes de adicionar um item a um *bin*, é verificado se este item é conflitante com os outros presentes no *bin*. O item é alocado somente se não acarretar conflito (GENDREAU; LAPORTE; SEMET, 2004). Adicionando um procedimento de ordenação dos itens de acordo com seus pesos antes do empacotamento, temos o método *Modified First Fit Decreasing* (MFFD).

Outra grande contribuição de Gendreau *et al.* (2004) foi a introdução das instâncias de referências *(benchmark instances)* mais utilizadas na literatura do BPPC. O procedimento de construção destas instâncias de teste foram baseadas no trabalho de Faulkenauer (1996) e adaptadas para as restrições de conflito do BPPC.

A Heurística Populacional *(PH)*, sigla do inglês *Population Heuristic*, proposta por Muritiba *et al.* (2010) utiliza da heurística *H6* de Gendreau *et al.* (2004) com modificações na busca pelo clique maximal no grafo de conflito. Um clique é um subconjunto de vértices no qual todos são adjacentes entre si. Novas soluções são geradas por operações de *crossover* e passam por uma Busca Tabu (GLOVER; LAGUNA, 1998), que trabalha com soluções parcialmente viáveis. O objetivo da estrutura de vizinhança é, dada uma partição V_{k+1} de *bins*, transferir os itens de V_{k+1} para os *bins* $V_1 ... V_k$, diminuindo em uma unidade o número de cargas utilizadas no empacotamento.

Muritiba *et al.* (2010) também apresentou um método exato baseado em um algoritmo *branch-and-price*, assim como avanços em limites inferiores e superiores para o problema, derivados dos limites estudados por Gendreau *et al.* (2004).

Sadykov e Vanderbeck (2013) melhoraram os resultados obtidos pelo método exato de Muritiba *et al.* (2010), com a implementação de um algoritmo genérico *branch-and-price*. O algoritmo conta com um esquema genérico de ramificação e heurística primal para geração de colunas. Dependendo da estrutura do grafo de conflito, o problema de *princing* é resolvido através de programação dinâmica ou de um *branch-and-bound* com busca em profundidade.

Pela falta de instâncias de teste com grafos de conflito arbitrários na literatura, novos conjuntos foram gerados pelos autores. Estas são instâncias de teste frequentemente utilizadas na literatura do BPPC para avaliação e comparação de métodos do estado da arte.

No trabalho de Capua *et al.* (2018), uma heurística *ILS (Iterated Local Search)* foi proposta para tratar o BPPC. Os autores implementaram um ILS com o método construtivo MFFD, uma busca local e as vizinhanças de tamanho exponencial *Ejection Chains*, *Assignment*, *Grenade* e *Set Covering*, alcançando resultados com uma média de qualidade superior à melhor meta-heurística

até então conhecida, a Heurística Populacional de Muritiba *et al.* (2010). O método proposto só não superou aquele desenvolvido por Sadykov e Vanderbeck (2013).

A Tabela 2.1 menciona trabalhos relevantes da literatura do BPPC e resume suas metodologias. Estes trabalhos são alguns dos mais referenciados quando tratamos de métodos aproximados para resolver o problema.

Autores	Metodologia
Gendreau *et al.* (2004)	Métodos heurísticos
Muritiba *et al.* (2010)	*Population Heuristic* (PH)
Sadykov e Vanderbeck (2013)	*Branch-and-Price*
Capua *et al.* (2018)	Meta-heurística ILS

Tabela 2.1 – Trabalhos relevantes sobre o BPPC

3 Metodologia

Este capítulo apresenta a metodologia empregada no desenvolvimento deste trabalho, explicando conceitos, procedimentos e algoritmos aplicados. A representação de uma solução para o BPPC é apresentada em 3.1. A Seção 3.2 descreve a implementação do algoritmo, seus componentes e detalhes. O método desenvolvido resolve o BPPC *off-line*.

3.1 Representação da Solução

Uma solução para o BPPC é um arranjo de itens em caixas, de modo que a capacidade do *bin* suporte todos os itens presentes nele e garantindo que não há incompatibilidade entre itens de um mesmo *bin*.

A solução pode ser representada por um vetor de vetores B. Cada elemento i no vetor representa um *bin* B_i. O vetor em B_i irá conter os elementos $j...n$. Uma solução viável é aquela em que, para todos os *bins*, a soma dos pesos dos itens em um *bin* não ultrapassa sua capacidade máxima e nenhum par de itens presentes no mesmo *bin* são conflitantes. O foco do problema é acomodar os itens no menor número z de *bins* necessários. Esta quantidade é a soma do número de vetores não vazios em B.

A Figura 3.1 mostra um conjunto de itens a serem empacotados, as relações de conflitos e um exemplo de representação de solução para o *Bin Packing* com Conflitos. Segundo a figura, percebemos que os itens 1 e 2 apresentam conflito entre si, assim como os itens 3 e 4. Nota-se que, na solução, itens conflitantes entre si não estão presentes em um mesmo *bin*.

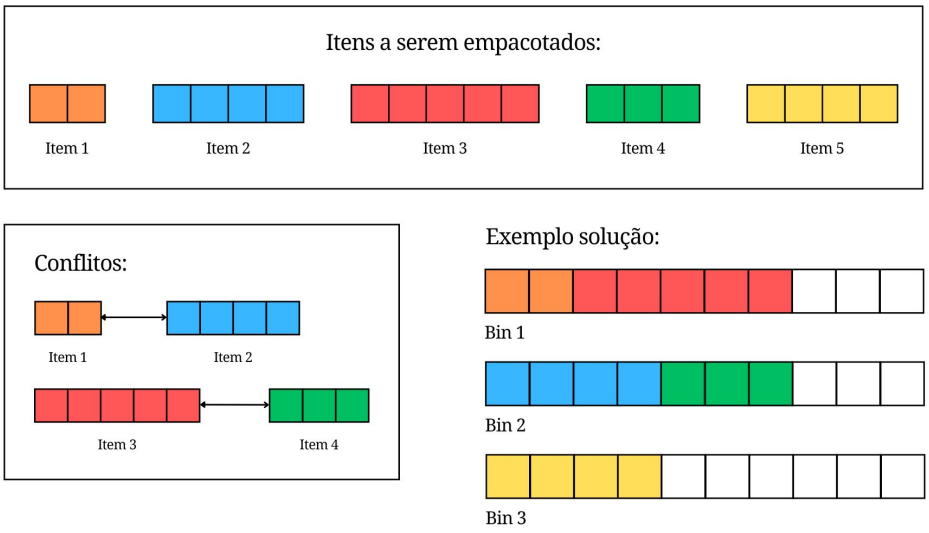

Figura 3.1 – Representação de solução para o *Bin Packing* com Conflitos.

3.2 Método proposto

Esta seção apresenta o algoritmo SA implementado e detalha cada um de seus mecanismos. Dois métodos construtivos para geração da solução inicial são apresentados em 3.2.1. As estruturas de vizinhança exploradas são descritas em 3.2.2, e a função de avaliação usada para comparar diferentes soluções é explicada em 3.2.3.

3.2.1 Solução inicial

O SA recebe como entrada uma solução inicial válida, que vai sofrendo alterações até o fim do algoritmo. Dois métodos construtivos foram desenvolvidos, sendo eles o *Modified First Fit Decreasing* (MFFD), descrito por Gendreau *et al.* (2004), e um método randômico.

No MFFD, todos os itens são ordenados em ordem decrescente de acordo com seus pesos antes de começar o empacotamento. Cada item é designado ao primeiro *bin* que o couber e para o qual não seja conflitante com nenhum item já atribuído. O resultado deste método será uma solução inicial viável que utiliza z *bins*. O pseudocódigo é representado no Algoritmo 3.1.

Algoritmo 3.1: Pseudocódigo do Modified First Fit Decreasing (MFFD)

```
   Entrada: lista de itens V, capacidade dos bins C, grafo de conflito E
   Saída: solução S
1  S ← binVazio;
2  ordenar lista de itens V em ordem decrescente;
3  para cada item ∈ V faça
4      para cada bin ∈ S faça
5          se item couber em bin E não ter conflito com nenhum item em bin então
6              adicionar item ao bin;
7              vá para a linha 3;
8      se item não coube em nenhum bin então
9          criar novo bin e adicionar item;
10         adicionar bin à solução S;
11 retorna S;
```

O Algoritmo 3.1 recebe como entrada uma lista de itens V, o valor da capacidade máxima dos *bins* C e o grafo de conflito E. A solução é inicializada com um *bin* vazio na linha 1 e os itens são ordenados em ordem decrescente (linha 2). O laço principal (linhas 3-10) analisa todos os itens, enquanto que o laço interior (linhas 4-7) analisa todos os *bins* abertos presentes na solução. A linha 5 verifica se o item cabe no *bin* atual e também checa por conflitos com os outros itens já presentes nele. Se nenhuma restrição for violada, o item é adicionado ao *bin* (linha 6). Caso não seja encontrado nenhum *bin* apto a receber o item, um novo é aberto para recebê-lo (linha 9). A solução é atualizada logo em seguida (linha 10).

No método construtivo randômico, itens são selecionados aleatoriamente e adicionados ao único *bin* aberto no momento. Se a adição do item causar conflito com outro já alocado ou fizer com que a capacidade do *bin* seja ultrapassada, o *bin* é fechado e outro é aberto. Este é um método consideravelmente simples e direto, seguindo uma lógica similar ao *Next Fit* (NF), onde há apenas um *bin* aberto por vez. O algoritmo é representado em 3.2.

Algoritmo 3.2: Pseudocódigo do método construtivo aleatório

Entrada: lista de itens **V**, capacidade dos bins **C**, grafo de conflito **E**
Saída: solução **S**

```
1 S ← ∅;
2 bin ← ∅;
3 para cada item ∈ V em ordem aleatória faça
4     se item couber em bin E não ter conflito com nenhum item do bin então
5         adicionar item ao bin;
6     senão
7         adicionar bin à solução S;
8         abrir novo bin e adicionar item;
9 retorna S;
```

A entrada do Algoritmo 3.2 consiste em uma lista de itens V, a capacidade máxima dos *bins* C, e o grafo de conflito E. As linhas 1 e 2 inicializam as variáveis da solução e do *bin* atual como vazias. O laço principal passa por todos os itens a serem empacotados (linhas 3-8), escolhidos aleatoriamente. Se o item escolhido couber no *bin* atual, ele é alocado ao *bin* (linhas 4 e 5). Se a adição do item violar alguma restrição, seja de capacidade ou conflito, o *bin* é fechado, sem a adição do item, e adicionado à solução. Em seguida, um novo *bin* é aberto para receber o item (linha 8). A linha 7 adiciona o último *bin* à solução S, que é retornada na linha 9.

Uma comparação interessante surge com os dois métodos construtivos, onde um segue uma estratégia gulosa no empacotamento dos itens e o outro é totalmente aleatório. O SA, no entanto, se aproveita de soluções iniciais "ruins" em qualidade. Pela sua natureza, ele começa a explorar o espaço de busca gerando uma solução vizinha qualquer, o que na maioria da vezes acaba piorando, ou bagunçando, a solução inicial. Com o passar das iterações e com o resfriamento da temperatura, o algoritmo vai convergindo para soluções melhores.

3.2.2 Estruturas de vizinhança

A exploração do espaço de busca é feita a partir da aplicação de diferentes estruturas de vizinhança. Uma série de movimentações são aplicadas à solução atual para gerar soluções vizinhas. As estruturas de vizinhança implementadas foram a *realocação*, *troca 1x1*, *troca 2x1* e *destruir e reconstruir*.

realocação um item aleatório de um *bin* qualquer é realocado para outro *bin* da solução.

troca 1x1 um item i de um *bin* b_i é trocado de lugar com um item j de um *bin* b_j. A escolha dos *bins* e dos itens é feita aleatoriamente.

troca 2x1 funciona da mesma maneira que a *troca*, mas trocando dois itens de um *bin* por apenas um item de outro *bin*.

destruir e reconstruir o *bin* com a menor quantidade de itens é selecionado e seus itens são realocados para outros *bins*, seguindo uma lógica FF e respeitando as restrições de capacidade e conflito. Ao esvaziar o *bin*, ele é removido da solução.

As três primeiras estruturas são as vizinhanças mais populares na resolução dos problemas da classe do BPP. No entanto, Schneider e Kirkpatrick (2006) apontam a dificuldade de se deixar um ótimo local quando movimentações consideradas pequenas são aplicadas às soluções. Eles defendem a aplicação de movimentações substanciais, que mudem parte da configuração da solução. O destruir e reconstruir, do inglês *ruin and recreate*, é uma estrutura de vizinhança que causa uma mudança profunda, ao tentar eliminar um *bin* inteiro da solução.

Na realocação, o processo de escolha do *bin* destino para o item a ser realocado foi alterado, com o objetivo de permitir uma maior diversificação de soluções. Existe uma chance de um em n_{bins} (quantidade de *bins* da solução) que um novo *bin* seja aberto, receba o item e seja adicionado à solução. Esta abordagem surgiu da necessidade de se ter a possibilidade de aumentar o número de *bins* da solução, para resolver as violações de restrições de conflito e peso em excesso.

No SA, uma solução vizinha é gerada a cada iteração do algoritmo. A probabilidade de escolha de uma vizinhança é ponderada, sendo que o peso de cada uma é proporcional a quantidade de vezes que ela obtém sucesso ao gerar um vizinho de melhor qualidade. Para melhorar o desempenho do algoritmo, algumas estratégias de avaliação inteligente das soluções vizinhas são aplicadas. Por exemplo, no cálculo da função de avaliação da solução, apenas os *bins* envolvidos na movimentação são inspecionados e percorridos para cálculo de conflitos e peso em excesso.

3.2.3 Função de avaliação

O método proposto percorre uma série de soluções durante sua execução, estas sendo soluções viáveis e inviáveis. Soluções podem, então, possuir itens conflitantes entre si em um mesmo *bin*, ou *bins* com sua capacidade máxima ultrapassada. Estas são soluções consideradas inviáveis, e são permitidas durante o procedimento do SA para que haja uma melhor exploração do espaço de busca.

Surge, então, a necessidade de se avaliar soluções de uma maneira diferente que a função objetivo óbvia, que é o número de *bins* de uma solução. A função de avaliação implementada depende de três fatores: a quantidade de *bins* da solução, o número de conflitos entre itens

presentes em um mesmo *bin* e o peso total em excesso. Cada um desses fatores possui um fator de peso associado. Sendo um problema de minimização, a solução ótima para o problema é aquela que utiliza a menor quantidade de *bins* para empacotar os itens e onde não existe conflitos nem violação da capacidade dos *bins*.

$$fa(S) = \omega_1 * n_{bins} + \omega_2 * excess_w + \omega_3 * conflicts; \tag{3.1}$$

Na Equação (3.1), n_{bins} é o número de *bins* utilizados na solução e $excess_w$ é a soma dos pesos dos itens que está sendo ultrapassado nos *bins*. A quantidade de conflitos encontrada nos *bins* da solução é representada em $conflicts$. Os pesos ω_1, ω_2 e ω_3 são associados, respectivamente, a cada um dos fatores descritos, influenciando diretamente o resultado da função de avaliação.

3.2.4 Método SA

O SA recebe uma série de parâmetros, como a temperatura inicial T_i, fator de resfriamento α, temperatura mínima T_{min} e quantidade de iterações por temperatura SA_{max}. Outros três parâmetros determinam os fatores de penalidade de excesso de peso, número de conflitos e quantidade de *bins* de uma solução. Esses fatores de penalidade são a base para a avaliação de soluções vizinhas.

A interrupção do algoritmo acontece quando a temperatura mínima é atingida ou quando o limite inferior do problema apresentado em 2.1.4 é alcançado. No pior dos casos, a solução retornada ao final do algoritmo é a inicial. O pseudocódigo do método proposto é representado pelo Algoritmo 3.3.

Algoritmo 3.3: Pseudocódigo do Simulated Annealing

Entrada: S, E, T_i,α, T_{min}, SA_{max}, ω_1, ω_2, ω_3
Saída: solução S^*

```
1  S* ← S;
2  T ← T_i;
3  iter_T ← 0;
4  lb ← LowerBound();
5  enquanto T > T_min e S* > teto(lb) faça
6      enquanto iter_T < SA_max faça
7          S' ← neighbor(S);
8          Δ ← fa(S') − fa(S);
9          se Δ < 0 então
10             S ← S';
11             se S' é válida e fo(S') <= fo(S*) então
12                 S* ← S';
13         senão
14             x ← random[0, 1];
15             se x < e^(−Δ/T) então
16                 S ← S';
17         iter_T ← iter_T + 1;
18     T ← T * α;
19     iter_T ← 0;
20 retorna S*;
```

O Algoritmo 3.3 apresenta o método SA implementado. Uma solução inicial viável, recebida como entrada, é guardada em S^* (linha 1). A temperatura inicial T e a variável $iter_T$ são inicializadas nas linhas 2 e 3, enquanto que o limite inferior é calculado na linha 4. O laço principal começa na linha 5 e é interrompido quando a temperatura atual é menor que a temperatura mínima definida, ou quando a solução atinge o limite inferior lb. Um laço interno executa SA_{max} iterações do algoritmo sem alterar o valor da temperatura (linhas 6-17). A cada iteração, uma solução vizinha S' é gerada (linha 7). Um Δ é calculado da diferença dos valores da função de avaliação em S' e a solução atual S na linha 8. Se o vizinho for melhor, ele se torna a solução atual (linhas 9-10). Além disso,verifica-se se o vizinho é uma solução válida (sem violações de restrições) e se utiliza menos ou a mesma quantidade de *bins* que a melhor solução até o momento S^* (linha 11). Se este for o caso, S^* é atualizado em 12. Se a solução vizinha for pior que a solução atual, existe uma possibilidade de que S' seja aceita. Após SA_{max} iterações, a temperatura é reduzida de acordo com o fator de resfriamento e a variável $iter_T$ é zerada. O retorno do algoritmo é a solução S^*, a melhor dentre todas analisadas.

4 Experimentação Prática

Este Capítulo apresenta detalhes dos experimentos computacionais que compõem o trabalho, juntamente com os resultados obtidos. A Seção 4.1 descreve as instâncias de teste utilizadas para resolver o BPPC. Uma descrição detalhada dos experimentos é relatada na Seção 4.2. Os resultados dos experimentos são apresentados na Seção 4.3.

Os algoritmos foram implementados em *C++* versão 9.4.0 e os experimentos executados em um computador com sistema operacional Ubuntu 20.04.6 LTS 64-bit com i7-1255U CPU 4.7GHz e 16GB de RAM.

4.1 Instâncias de Teste do BPPC

O algoritmo desenvolvido foi aplicado em uma série de instâncias de teste da literatura[1]. Um resumo das informações sobre parâmetros das instâncias é apresentado na Tabela 4.1. A coluna *"Instância"* aponta a classe daquele conjunto de casos de teste, *"#"* é a quantidade de instâncias disponíveis e *"Fonte"* cita o autor responsável pela geração das instâncias. As colunas *"n"* e *"C"* determinam, respectivamente, a quantidade de itens e a capacidade dos *bins* da classe de instâncias.

Os conjuntos (t) e (u) foram geradas por Muritiba *et al.* (2010), utilizando os mesmos parâmetros adotados por Gendreau *et al.* (2004). Em (t), a quantidade de itens $n \in [60, 120, 249, 501]$ e a capacidade dos *bins* $C = 100$. Os pesos dos itens possuem a mesma característica apresentada na Seção 4.3, onde a soma de cada três itens é igual a capacidade dos *bins*. O conjunto (u) possui *bins* com capacidade $C = 150$ e quantidade de itens $n \in [120, 250, 500, 1.000]$.

Além da lista de itens de entrada, são gerados grafos de conflito E com valores de densidade ρ de 10% a 90%. Para cada valor de n e de ρ, foram geradas 10 instâncias do BPPC. Os grafos foram gerados seguindo o procedimento: foi atribuído um valor p_i para cada vértice $i \in V$, de acordo com uma distribuição uniforme em $[0, 1]$. Para cada par (i, j), um conflito é criado se $(p_i + p_j)/2 \geq \rho$. Ao todo, os autores geraram, para cada conjunto (t) e (u), 360 casos de teste.

Sadykov e Vanderbeck (2013) geraram as instâncias (ta) e (ua) com as mesmas características das classes (t) e (u) de Muritiba *et al.* (2010), mudando apenas os grafos de conflito para grafos arbitrários. As densidades ρ dos grafos vão de 10% a 90%. Para a geração dos grafos de conflito, pares (i, j) de itens foram selecionados, iterativamente, e adicionados ao grafo E. O método era interrompido quando a densidade desejada era atingida. No total, foram geradas 360 instâncias cada.

[1] As instâncias para o BPPC estão disponíveis em: math.u-bordeaux.fr/ rsadykov/.

Os autores observaram que o número médio de itens por *bin* nos casos de teste anteriores eram pequenos (não excedia três). Com isso, dois novos grupos de instâncias mais difíceis foram geradas, (d) e (da). Para (d), o grafo de conflito é um grafo de intervalo, gerado através do mesmo procedimento de (t) e (u). No caso de (da), foram gerados grafos arbitrários, semelhantes a (ta) e (ua).

Nas classes (d) e (da), os pesos dos itens estão dentro do intervalo $[500, 2.500]$ e a capacidade dos *bins* é igual a 10.000. As instâncias possuem a quantidade de itens $n \in [120, 250, 500]$. Para cada valor de n e densidade do grafo de conflito (10% a 90%), foram geradas dez instâncias, totalizando 360 casos de teste para cada classe.

É importante apontar que o objetivo da criação das classes (d) e (da) só foi atingido parcialmente, já que o número médio de itens por *bin* da melhor solução conhecida ainda é baixo, sendo 2,76 para a classe (d) e 5,83 para (da) (CAPUA et al., 2018).

Instância	**#**	**Fonte**	***n***	***C***
u	360	Muritiba *et al.* (2010)	{120, 250, 500, 1.000}	150
t	360	Muritiba *et al.* (2010)	{60, 120, 249, 501}	1.000
ua	360	Sadykov e Vanderbeck (2013)	{120, 250, 500, 1.000}	150
ta	360	Sadykov e Vanderbeck (2013)	{60, 120, 249, 501}	1.000
d	360	Sadykov e Vanderbeck (2013)	{120, 250, 500, 1.000}	10.000
da	360	Sadykov e Vanderbeck (2013)	{120, 250, 500, 1.000}	10.000

Tabela 4.1 – Instâncias de teste para o BPPC

4.2 Experimentos do BPPC

Experimentos computacionais foram realizados com as instâncias de referência da literatura exibidas na Tabela 4.1, para analisar o desempenho do algoritmo desenvolvido.

Nem todas as soluções ótimas dos casos de teste do BPPC são conhecidas. Para estes casos, nos referimos à melhor solução conhecida na literatura: *Best Known Solution* (BKS). Tais dados foram obtidos através comunicação pessoal com a Dra. Renatha Capua, que utilizou estes dados no seu trabalho de 2018 sobre o BPPC (CAPUA et al., 2018).

Os parâmetros empregados no SA são descritos a seguir. A temperatura inicial utilizada pelo algoritmo é 500, com um fator α de resfriamento 0.98 e temperatura mínima 0.01. Para instâncias de 500 itens, SA_{max} é 800 e, nas instâncias de 1.000 itens, é 500. Os pesos ω_1, ω_2 e ω_3 da função de avaliação assumem os valores 1, 30 e 4, respectivamente.

O método de geração da solução inicial escolhido para os experimentos foi o método aleatório, descrito em 3.2.1.

4.3 Resultados Computacionais

Esta Seção descreve e discute os resultados obtidos nos testes computacionais onde o algoritmo SA desenvolvido foi aplicado. A Tabela 4.2 compara os resultados obtidos nos testes computacionais da Monografia I, onde apenas um método construtivo foi implementado, e da Monografia II, onde uma meta-heurística foi trabalhada. A coluna *Instância* aponta a Classe daquele conjunto de casos de teste e a coluna # determina a quantidade de instâncias da Classe. As seguintes métricas foram analisadas:

- ***#opt.***: quantidade de instâncias nas quais a solução ótima, ou BKS, foram encontradas.
- ***gap (%)***: desvio entre a solução encontrada em relação à solução ótima, ou BKS.
- ***T (s)***: tempo de execução do algoritmo, em segundos.

O método desenvolvido foi aplicado um total de 10 vezes em cada instância e as métricas *gap (%)* e *T (s)* representam a média entre todas as execuções.

Instância		**MFFD**			**SA**		
Classe	#	*#opt.*	*gap (%)*	*T (s)*	*#opt.*	*gap (%)*	*T (s)*
u	360	2	64.38	0.27	28	95.11	149.79
t	360	0	34.18	0.05	106	13.56	69.11
ua	360	0	71.19	0.16	4	46.92	135.54
ta	360	0	84.80	0.02	68	25.71	64.51
d	360	1	37.21	0.37	143	13.33	140.12
da	360	0	356.55	0.47	29	74.14	153.36
total	2.160	3	-	-	378	-	-

Tabela 4.2 – Resultado comparativo MFFD e SA

Uma evolução nos resultados é percebível ao analisar a quantidade de vezes que a solução ótima foi encontrada. O MFFD alcança a otimalidade em 3 instâncias e o método SA em 378, de um total de 2.160 instâncias, um salto de 0,13% para 17,5%.

O tempo de execução do SA é muito mais elevado que o tempo de execução do MFFD, como previsto. O algoritmo desenvolvido, com os parâmetros mencionados, efetua aproximadamente meio milhão de iterações por instância, enquanto que o MFFD apenas percorre a lista de itens e a lista de *bins* para realizar o procedimento de empacotamento. A diferença de custo de processamento dos dois métodos é significativa.

No caso das instâncias (u), percebe-se uma piora na média de qualidade das soluções com a aplicação do SA, mesmo ele tendo encontrado mais soluções ótimas. Isso ocorreu pela grande quantidade de vezes em que a solução retornada pelo método era igual a inicial. Uma das causas foi a rápida diminuição da quantidade de *bins* da solução no início da execução. O algoritmo se encontrava, então, preso com soluções com muitos conflitos entre itens e *bins* com

suas capacidades máximas ultrapassadas. Como a única chance de aumentar a quantidade de *bins* da solução era na vizinhança de realocação e mesmo assim era pequena, o algoritmo se perdia em um mar de soluções inviáveis. Esse fenômeno não aconteceu exclusivamente nas instâncias (u), mas essa classe foi a mais afetada.

Para todas as outras classes de instâncias, o *gap* obtido através da aplicação da meta-heurística é inferior àquele obtido pela aplicação do MFFD. Isso demonstra uma melhora na qualidade geral das soluções. Uma melhora substancial acontece no caso das instâncias (da), onde o *gap* de 356.55% passou para 74.14%.

As classes de instâncias mais fáceis de resolver com o MFFD também foram as que produziram melhores soluções com o SA, sendo elas (t) e (d).

A Tabela 4.3 apresenta os resultado dos testes, agrupando-os pela classe de instâncias e quantidade de itens. A coluna n expõe a quantidade de itens a serem empacotados pelo algoritmo.

Instância		**SA**		
Classe	n	*#opt.*	*gap (%)*	*T (s)*
u	120	28	6.321	39.83
	250	0	137.78	70.17
	500	0	201.23	132.83
	1.000	0	35.11	356.33
t	60	69	4.63	24.48
	120	34	6.82	39.59
	249	3	16.71	65.62
	501	0	26.08	146.745
ua	120	0	11.589	38.13
	250	4	29.36	68.47
	500	0	66.77	132.42
	1.000	0	79.97	303.14
ta	60	68	7.11	23.73
	120	0	9.83	37.12
	259	0	21.95	67.20
	501	0	63.96	129.98
d	120	77	3.46	37.65
	250	57	9.81	70.69
	500	5	19.48	137.64
	1.000	4	20.59	314.51
da	120	20	17.67	32.73
	250	7	22.20	71.82
	500	0	55.42	148.53
	1.000	2	201.27	360.36

Tabela 4.3 – Resultado geral do BPPC

Soluções ótimas são encontradas diversas vezes com a aplicação do SA, em um total de 378 instâncias, de 2.160. A classe de instâncias onde a solução ótima foi encontrada mais vezes é a (d), seguida da (t) e (ta). A otimalidade foi alcançada 143, 106 e 68 vezes, respectivamente,

dentro das 360 instâncias de cada grupo. O método se mostra mais eficiente em instâncias com poucos itens, exceto no caso da classe (ua), onde a solução ótima não foi alcançada nenhuma vez em instâncias de 120 itens, mas sim em 4 instâncias de 250 itens.

O tempo de execução do algoritmo aumenta com a a quantidade de itens das instâncias, para todos os casos, devido ao custo de processamento dos itens e suas listas de conflitos. Instâncias com 1.000 itens apresentaram um tempo de execução elevado, chegando a quase 8 minutos para um único caso de teste. Tardamentos como estes são uma desvantagem em aplicações de larga escala que demandariam agilidade, ou a execução do algoritmo repetidas vezes.

Ao comparar instâncias de grafo de conflito de intervalo e arbitrário, percebemos que (ta) e (da) foram mais difíceis de resolver do as suas versões (t) e (d). Isso não aconteceu, porém, no caso de (u) e (ua).

Os piores resultados em qualidade foram obtidos na classe de instâncias (u), onde as soluções retornadas eram, muitas vezes, iguais as soluções iniciais, geradas pelo método construtivo aleatório. O *gap* apresentou valores altos para as instâncias de 250 e 500 itens.

As Tabelas 4.4 e 4.5 expõem os resultados dos testes por classe de instância e valor de densidade do grafo de conflito ρ. Valores menores representam casos com menos conflitos entre itens, enquanto que porcentagens maiores refletem em grafos densos, onde existem muitos conflitos. Para cada valor de densidade, são 40 instâncias contempladas. As colunas são as mesmas da Tabela 4.3, apenas com a troca da coluna n por ρ.

Classe	ρ	*#opt.*	*gap (%)*	*T (s)*
u	10%	0	144.65	104.15
	20%	0	144.57	108.12
	30%	0	152.57	109.63
	40%	3	166.86	116.01
	50%	7	58.11	139.89
	60%	5	54.64	192.41
	70%	5	51.70	194.60
	80%	5	45.94	211.52
	90%	3	36.92	171.79
ua	10%	2	22.77	112.53
	20%	2	24.60	127.64
	30%	0	29.43	130.80
	40%	0	32.78	127.67
	50%	0	37.67	149.22
	60%	0	44.58	133.61
	70%	0	57.37	137.39
	80%	0	80.26	145.75
	90%	0	92.82	155.27
t	10%	2	10.50	53.62
	20%	9	10.52	54.86
	30%	10	18.06	63.48
	40%	14	18.82	78.07
	50%	16	15.50	73.39
	60%	19	14.70	74.44
	70%	16	14.39	77.08
	80%	16	12.11	76.49
	90%	4	7.43	70.55
ta	10%	9	13.89	61.08
	20%	10	17.53	61.95
	30%	10	19.54	62.73
	40%	10	20.14	63.15
	50%	7	18.40	63.54
	60%	3	16.66	64.87
	70%	1	22.46	66.17
	80%	8	44.88	67.55
	90%	10	57.92	69.52

Tabela 4.4 – Resultado por densidade do grafo de conflito (Parte 2)

Soluções ótimas foram encontradas mais vezes em instâncias com densidade de grafo de conflito por volta de 10%, 20% e 30%, para os grupos de grafo de conflito arbitrário (ua), (ta) e (da). As demais classes (u) (t) atingiram mais vezes a otimalidade em instâncias com densidade de grafo de conflito por volta de 50%, 60% e 70%. As instâncias (d) encontraram a solução ótima em pelo menos metade das vezes quando ρ era 20%, 30% e 50%, além de ter um ótimo desempenho nos outros casos.

A qualidade geral das soluções pode ser melhor enquanto a quantidade de vezes que a

Classe	ρ	*#opt.*	*gap (%)*	*T (s)*
d	10%	14	2.84	91.06
	20%	23	19.23	102.87
	30%	20	16.34	128.88
	40%	17	16.14	144.88
	50%	21	14.07	156.33
	60%	17	14.52	172.67
	70%	18	15.26	153.59
	80%	12	13.32	160.63
	90%	1	8.29	150.19
da	10%	11	7.28	120.77
	20%	9	8.60	119.45
	30%	6	14.47	125.41
	40%	3	51.057	126.82
	50%	0	70.73	167.87
	60%	0	122.97	142.82
	70%	0	139.03	177.81
	80%	0	144.58	203.57
	90%	0	108.52	195.71

Tabela 4.5 – Resultado por densidade do grafo de conflito (Parte 3)

solução ótima foi atingida for menor. Esse é o caso para as instâncias (u), (t) e (d), que, para um ρ em 90%, o *gap* é o mais baixo ou segundo mais baixo do grupo, mesmo com a otimalidade sendo alcançada poucas vezes. Isso não acontece com o restante das instâncias (ua), (ta) e (da), onde o *gap* é menor para valores baixos de densidade.

O tempo de execução do SA cresce com o aumento da densidade dos grafos de conflito, no geral. Em algumas classes de instâncias o tempo cai no último valor de ρ, 90%, quando grafos de conflitos são extremamente densos e quase todos os itens possuem conflito entre si.

5 Considerações Finais

Este Capítulo apresenta as considerações finais deste trabalho, com uma breve conclusão na Seção 5.1 e um direcionamento à trabalhos futuros na Seção 5.2.

5.1 Conclusões

Este trabalho se propôs a estudar métodos aproximados para resolver o problema unidimensional de *Bin Packing* com Conflitos, derivado do clássico *Bin Packing Problem*. Experimentos computacionais foram empregados para se relatar a eficiência da meta-heurística *Simulated Annealing* e os resultados são analisados e discutidos.

No início do trabalho, é feita uma sucinta contextualização histórica sobre a classe de problemas do BPP, assim como um relato do panorama geral da evolução de métodos aproximados. O melhor método construtivo dos experimentos de Monografia I foi mantido, o MFFD, e outro método construtivo aleatório foi implementado. Um algoritmo SA com quatro vizinhanças e uma função de avaliação que depende do número de *bins* da solução e da violação de restrições do problema foi desenvolvido. Uma comparação é feita para avaliar a evolução de um método heurístico para um meta-heurístico. Para testar os algoritmos implementados, foram utilizadas instâncias comumente referenciadas na literatura.

O SA alcançou a solução ótima em 378 instâncias e o MFFD em 3, de um total de 2.160 instâncias. Ao se analisar a qualidade geral das soluções, no entanto, percebe-se que o SA obteve valores médios piores apenas para a classe (u) de instâncias. Nos outros casos, a meta-heurística superou o método construtivo e a distância até a otimalidade foi encurtada.

No geral, o SA atingiu resultados satisfatórios, produzindo soluções com qualidade. O método desenvolvido, no entanto, se provou pouco eficiente em alguns casos, principalmente em casos de empacotamento de 500 e 1.000 itens.

O objetivo do trabalho foi, então, alcançado, com um estudo sobre métodos aproximados para resolver o BPPC e com a aplicação de extensos testes computacionais para avaliar o método meta-heurístico desenvolvido.

5.2 Trabalhos Futuros

Um grande desafio ao trabalhar com meta-heurísticas em problemas de otimização combinatória é o ajuste de hiperparâmetros. Estes influenciam diretamente no desempenho final do algoritmo. No SA desenvolvido, os três pesos da função de avaliação desempenham um papel crucial na movimentação pelo espaço de busca e na convergência do método. Um

ajuste manual desses parâmetros pode resultar na ineficiência do método. Em trabalhos futuros, é interessante a realização de um ajuste automático dos parâmetros. Por exemplo, o *Iterated Racing for Automatic Algorithm Configuration* (*irace*) é um algoritmo que encontra a melhor configuração de parâmetros utilizando testes estatísticos.

Uma oportunidade de aumentar a quantidade de *bins* de uma solução pode ser vantajosa para tratar casos onde o algoritmo fica preso em um mar de soluções inviáveis, sem conseguir caminhar até soluções onde as restrições do problema não sejam violadas. A adição de uma vizinhança que abre um novo *bin* e o adiciona à solução é interessante para que o algoritmo consiga resolver os conflitos e pesos em excesso, redistribuindo melhor seus itens. Outra proposta para contornar o obstáculo pode ser trabalhar com uma solução inicial onde cada item do problema está alocado em um *bin* diferente. Com o passar das iterações e geração de vizinhos, *bins* seriam esvaziados e retirados da solução. Assim, o algoritmo convergia, de modo mais lento e mais seguro, sem violar muitas restrições, a uma solução boa em qualidade.

Referências

AARTS, E.; KORST, J.; MICHIELS, W. Simulated annealing. *Search methodologies: introductory tutorials in optimization and decision support techniques*, Springer, p. 187–210, 2005.

ABDEL-BASSET, M.; MANOGARAN, G.; ABDEL-FATAH, L.; MIRJALILI, S. An improved nature inspired meta-heuristic algorithm for 1-d bin packing problems. *Personal and Ubiquitous Computing*, Springer, v. 22, p. 1117–1132, 2018.

BAKER, T.; ALDAWSARI, B.; ASIM, M.; TAWFIK, H.; MAAMAR, Z.; BUYYA, R. Cloud-senergy: A bin-packing based multi-cloud service broker for energy efficient composition and execution of data-intensive applications. *Sustainable Computing: informatics and systems*, Elsevier, v. 19, p. 242–252, 2018.

BALDI, M. M.; CRAINIC, T. G.; PERBOLI, G.; TADEI, R. The generalized bin packing problem. *Transportation Research Part E: Logistics and Transportation Review*, Elsevier, v. 48, n. 6, p. 1205–1220, 2012.

BANSAL, N.; LIU, Z.; SANKAR, A. Bin-packing with fragile objects and frequency allocation in cellular networks. *Wireless Networks*, Springer, v. 15, p. 821–830, 2009.

BHATIA, A. K.; HAZRA, M.; BASU, S. Better-fit heuristic for one-dimensional bin-packing problem. In: IEEE. *2009 IEEE International Advance Computing Conference*. [S.l.], 2009. p. 193–196.

BROWN, A. R. *Optimum packing and depletion: the computer in space-and resource-usage problems*. [S.l.]: Macdonald and Co., American Elsevier, 1971.

CAPUA, R.; FROTA, Y.; OCHI, L. S.; VIDAL, T. A study on exponential-size neighborhoods for the bin packing problem with conflicts. *Journal of Heuristics*, Springer, v. 24, p. 667–695, 2018.

ČERNỲ, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, Springer, v. 45, p. 41–51, 1985.

COFFMAN, E. G.; CSIRIK, J.; GALAMBOS, G.; MARTELLO, S.; VIGO, D. et al. Bin packing approximation algorithms: Survey and classification. In: *Handbook of combinatorial optimization*. [S.l.]: Springer, 2013. p. 455–531.

COFFMAN JR, E. G.; GAREY, M. R.; JOHNSON, D. S. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, SIAM, v. 7, n. 1, p. 1–17, 1978.

DELORME, M.; IORI, M.; MARTELLO, S. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, Elsevier, v. 255, n. 1, p. 1–20, 2016.

EILON, S.; CHRISTOFIDES, N. The loading problem. *Management Science*, INFORMS, v. 17, n. 5, p. 259–268, 1971.

EL-ASHMAWI, W. H.; ELMINAAM, D. S. A. A modified squirrel search algorithm based on improved best fit heuristic and operator strategy for bin packing problem. *Applied Soft Computing*, Elsevier, v. 82, p. 105565, 2019.

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

FLESZAR, K.; HINDI, K. S. New heuristics for one-dimensional bin-packing. *Computers & operations research*, Elsevier, v. 29, n. 7, p. 821–839, 2002.

GALINIER, P.; HAMIEZ, J.-P.; HAO, J.-K.; PORUMBEL, D. Recent advances in graph vertex coloring. *Handbook of Optimization: From Classical to Modern Approach*, Springer, p. 505–528, 2013.

GAREY, M. R.; GRAHAM, R. L.; ULLMAN, J. D. Worst-case analysis of memory allocation algorithms. In: *Proceedings of the fourth annual ACM symposium on Theory of computing*. [S.l.: s.n.], 1972. p. 143–150.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-completeness*. [S.l.]: WH Freeman and Company, New York, 1979.

GAREY, M. R.; JOHNSON, D. S. Approximation algorithms for bin packing problems: A survey. In: *Analysis and design of algorithms in combinatorial optimization*. [S.l.]: Springer, 1981. p. 147–172.

GENDREAU, M.; LAPORTE, G.; SEMET, F. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research*, Elsevier, v. 31, n. 3, p. 347–358, 2004.

GLOVER, F.; LAGUNA, M. *Tabu search*. [S.l.]: Springer, 1998.

GOLDBARG, E.; GOLDBARG, M.; LUNA, H. *Otimização combinatória e metaheurísticas: algoritmos e apliacações*. [S.l.]: Elsevier Brasil, 2017.

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.

GUPTA, J. N.; HO, J. C. A new heuristic algorithm for the one-dimensional bin-packing problem. *Production planning & control*, Taylor & Francis, v. 10, n. 6, p. 598–603, 1999.

HALL, N. G.; GHOSH, S.; KANKEY, R. D.; NARASIMHAN, S.; RHEE, W. T. Bin packing problems in one dimension: heuristic solutions and confidence intervals. *Computers & operations research*, Elsevier, v. 15, n. 2, p. 171–177, 1988.

HAMDI-DHAOUI, K.; LABADIE, N.; YALAOUI, A. The bi-objective two-dimensional loading vehicle routing problem with partial conflicts. *International Journal of Production Research*, Taylor & Francis, v. 52, n. 19, p. 5565–5582, 2014.

IRANI, S.; LEUNG, V. Scheduling with conflicts, and applications to traffic signal control. In: *Proceedings of the seventh annual ACM-SIAM symposium on discrete algorithms*. [S.l.: s.n.], 1996. p. 85–94.

JANSEN, K. An approximation scheme for bin packing with conflicts. *Journal of combinatorial optimization*, Springer, v. 3, n. 4, p. 363–377, 1999.

JANSEN, K.; ÖHRING, S. Approximation algorithms for time constrained scheduling. *Information and computation*, Elsevier, v. 132, n. 2, p. 85–108, 1997.

JOHNSON, D. S. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, Elsevier, v. 8, n. 3, p. 272–314, 1974.

JR, E. G. C.; GAREY, M. R.; JOHNSON, D. S. Approximation algorithms for bin-packing—an updated survey. In: *Algorithm design for computer system design*. [S.l.]: Springer, 1984. p. 49–106.

KÄMPKE, T. Simulated annealing: Use of a new tool in bin packing. *Annals of Operations Research*, Springer, v. 16, p. 327–332, 1988.

KIRKPATRICK, S.; JR, C. D. G.; VECCHI, M. P. Optimization by simulated annealing. *science*, American association for the advancement of science, v. 220, n. 4598, p. 671–680, 1983.

LABBÉ, M.; LAPORTE, G.; MERCURE, H. Capacitated vehicle routing on trees. *Operations Research*, INFORMS, v. 39, n. 4, p. 616–622, 1991.

LODI, A.; MARTELLO, S.; MONACI, M.; VIGO, D. Two-dimensional bin packing problems. *Paradigms of combinatorial optimization: Problems and new approaches*, Wiley Online Library, p. 107–129, 2014.

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

MENAKERMAN, N.; ROM, R. Bin packing with item fragmentation. In: SPRINGER. *Algorithms and Data Structures: 7th International Workshop, WADS 2001 Providence, RI, USA, August 8–10, 2001 Proceedings 7*. [S.l.], 2001. p. 313–324.

MUNIEN, C.; EZUGWU, A. E. Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications. *Journal of Intelligent Systems*, De Gruyter, v. 30, n. 1, p. 636–663, 2021.

MURITIBA, A. E. F.; IORI, M.; MALAGUTI, E.; TOTH, P. Algorithms for the bin packing problem with conflicts. *Informs Journal on computing*, INFORMS, v. 22, n. 3, p. 401–415, 2010.

OSOGAMI, T.; OKANO, H. Local search algorithms for the bin packing problem and their relationships to various construction heuristics. *Journal of Heuristics*, Springer, v. 9, n. 1, p. 29–49, 2003.

RHIAT, A.; AGGOUN, A.; LACHERE, R. Combining mobile robotics and packing for optimal deliveries. *Procedia Manufacturing*, Elsevier, v. 44, p. 536–542, 2020.

SADYKOV, R.; VANDERBECK, F. Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS Journal on Computing*, INFORMS, v. 25, n. 2, p. 244–255, 2013.

SCHNEIDER, J. J.; KIRKPATRICK, S. Ruin & recreate. *Stochastic Optimization*, Springer, p. 73–77, 2006.

SEIDEN, S. S.; STEE, R. V.; EPSTEIN, L. New bounds for variable-sized online bin packing. *SIAM Journal on Computing*, SIAM, v. 32, n. 2, p. 455–469, 2003.

SIMCHI-LEVI, D. New worst-case results for the bin-packing problem. *Naval Research Logistics (NRL)*, Wiley Online Library, v. 41, n. 4, p. 579–585, 1994.

WEI, L.; LUO, Z.; BALDACCI, R.; LIM, A. A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*, INFORMS, v. 32, n. 2, p. 428–443, 2020.

YANG, X.-S.; DEB, S. Cuckoo search via lévy flights. In: IEEE. *2009 World congress on nature & biologically inspired computing (NaBIC)*. [S.l.], 2009. p. 210–214.

ZENDAOUI, Z.; LAYEB, A. Adaptive cuckoo search algorithm for the bin packing problem. In: SPRINGER. *Modelling and Implementation of Complex Systems: Proceedings of the 4th International Symposium, MISC 2016, Constantine, Algeria, May 7-8, 2016, Constantine, Algeria*. [S.l.], 2016. p. 107–120.