57º SBPO
SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
Gramado • RS
DE 5 A 9 DE OUTUBRO DE 2025

SIMULATED ANNEALING APLICADO AO PROBLEMA DE BIN PACKING COM CONFLITOS

Rafael Coelho Monte Alto
Universidade Federal de Ouro Preto
Rua Professor Paulo Magalhães Gomes, 122 - Bauxita, Ouro Preto - MG, 35400-000
rafaelmalto@hotmail.com

Prof. Dr. Puca Huachi Vaz Penna
Universidade Federal de Ouro Preto
Rua Professor Paulo Magalhães Gomes, 122 - Bauxita, Ouro Preto - MG, 35400-000
puca@ufop.edu.br

RESUMO

Este trabalho aborda o Problema de Empacotamento com Conflitos, ou *Bin Packing Problem with Conflicts* (BPPC). Este problema é uma variante do clássico Problema de Empacotamento, do inglês *Bin Packing Problem* (BPP). No BPPC, uma série de itens devem ser empacotados no menor número de *bins* possíveis, respeitando as restrições de capacidade e lidando com itens que apresentam conflito entre si e, portanto, não podem ser alocados ao mesmo *bin*. O método *Simulated Annealing* com uma solução inicial aleatória e quatro estruturas de vizinhanças é proposto para resolver o BPPC. O algoritmo é testado em 2.160 instâncias de teste da literatura e os resultados demonstram que foram obtidas soluções de alta qualidade. O desvio relativo médio em relação as melhores soluções conhecidas foi de apenas $4,81\%$, e a otimalidade foi alcançada em $24,21\%$ dos casos.

PALAVRAS CHAVE. ***Simulated Annealing*, *Bin Packing* com Conflitos, Meta-heurística.**

Tópicos: MH – Meta-heurísticas, L&T – Logística e Transportes

ABSTRACT

This paper tackles the combinatorial optimization problem called the Bin Packing Problem with Conflicts (BPPC). The problem is variant of the classic Bin Packing Problem (BPP). 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. A Simulated Annealing method with a random initial solution and four neighborhood moves is implemented. The proposed algorithm is tested on 2,160 instances from the literature and results show that high quality solutions were obtained. The average gap from the best known solutions was 4.81% and optimality was reached in 24.21% of cases.

KEYWORDS. Simulated Annealing, Bin Packing with Conflicts, Metaheuristics.

Paper topics: MH – Metaheuristics, L&T – Logistics and Transport

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br

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 itens em caixas (*bins*) de um tamanho fixo, minimizando o número de caixas utilizadas. 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). Existe, também, o BPP bi-dimensional (2D-BPP) e o tridimensional (3D-BPP) . No 2D-BPP, uma série de itens retangulares devem ser empacotados em *bins* idênticos também retangulares. O 3D-BPP envolve itens com medidas de largura, altura e profundidade. Para mais informações sobre variantes do BPP, referenciamos [Coffman et al., 2013].

Este trabalho foca na variante conhecida como Problema de *Bin Packing* com Conflitos, do inglês *Bin Packing Problem with Conflicts* (BPPC). Diferentemente do problema clássico BPP, essa variante possui itens que são conflitantes entre si 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 estar juntos em uma mesma carga [Hamdi-Dhaoui et al., 2014].

O BPP, assim como o BPPC, é um problema classificado como NP-difícil [Jansen e Öhring, 1997] e, portanto, não existe, até então, um algoritmo que o resolva em tempo polinomial. Neste sentido, para abordar o problema é apresentado um método baseado na meta-heurística Simulated Annealing (SA) [Kirkpatrick et al., 1983]. O SA se mostrou bastante promissor encontrando diversas soluções ótimas conhecidas na literatura.

O restante do trabalho está organizado como segue. Na Seção 2 é apresentada uma breve revisão da literatura. Na Seção 3 o problema é descrito. O método desenvolvido para resolver o problema é detalhado na Seção 4. Os resultados computacionais são apresentados na Seção 5. Por fim, a Seção 6 conclui o trabalho.

2. Trabalhos Relacionados

[Jansen e Öhring, 1997] introduziram o BPPC em seu trabalho sobre agendamento de trabalhos com restrição de tempo. O objetivo era 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 *bins*, tomando cuidado para não adicionar itens conflitantes a um mesmo *bin*.

[Gendreau et al., 2004] descrevem dois limites inferiores do problema e seis métodos heurísticos para resolver o BPPC. Estes métodos utilizam de estratégias de coloração de grafos, construção de cliques e heurísticas populares da literatura, como o *First Fit Decreasing* (FFD). A heurística que apresentou o melhor desempenho nos experimentos de [Gendreau et al., 2004] foi o método *H6*, onde os itens que não apresentam conflitos com nenhum outro são separados em um conjunto H. Em seguida, um clique D é computado no grafo de conflito, agrupando itens conflitantes entre si. Para cada item $i \in D$, um clique D_i é encontrado no grafo de não conflito, evidenciando itens que podem estar presentes em um mesmo *bin* que i. O FFD é utilizado para resolver cada conjunto D_i separadamente. Por fim, os itens que não possuem conflito com nenhum outro são alocados nos *bins* parcialmente preenchidos, ou, se for preciso, em novos.

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. Novas soluções são geradas por operações de *crossover* e passam por uma Busca Tabu [Glover e 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

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br

empacotamento. Os autores também apresentaram 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 *pricing* é resolvido através de programação dinâmica ou de um *branch-and-bound* com busca em profundidade.

Em [Capua et al., 2018], uma meta-heurística *Iterated Local Search* (ILS) foi proposta para resolver o BPPC. Os autores implementaram o ILS com o método construtivo *Modified First Fit Decreasing* (MFFD), uma busca local e vizinhanças de tamanho exponencial, alcançando resultados com uma média de qualidade superior à melhor meta-heurística até então conhecida. Os resultados dos autores só não superou os de [Sadykov e Vanderbeck, 2013], obtidos pela aplicação de seu método exato.

3. Descrição do Problema

No BPPC, é fornecida uma quantidade infinita de *bins* B de capacidade fixa C, uma lista com n itens, cada um com um peso w_i $(i = 1, ..., n)$ e 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 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. Além disso, itens conflitantes entre si não podem estar alocados em um mesmo *bin*. 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 et al., 2016].

O BPPC pode ser formulado como um modelo de Programação Linear Inteiro (PLI). A seguinte formulação é apresentada 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. Já a variável x_{ik} será 1 se o item i estiver presente no *bin* k e 0, caso contrário.

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

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_{ik} \leq C y_k \quad (k = 1, \ldots, n) \tag{2}$$

$$\sum_{k=1}^{n} x_{ik} = 1 \quad (i = 1, \ldots, n) \tag{3}$$

$$x_{ik} + x_{jk} \leq 1 \quad ((i, j) \in E, k = 1, \ldots, n) \tag{4}$$

$$y_k = 0 \text{ ou } 1 \quad (k = 1, \ldots, n) \tag{5}$$

$$x_{ik} = 0 \text{ ou } 1 \quad (i, k = 1, \ldots, n) \tag{6}$$

A função objetivo (1) assegura a minimização do número de *bins* utilizados no empacotamento. As restrições (2) garantem que a soma dos pesos de todos os itens presentes em um *bin* não irá ultrapassar sua capacidade , além de definir os valores de y_k como 1 (*bin* utilizado) ou 0 (*bin* inutilizado). As restrições (3) determinam que cada item deve estar presente em um único *bin*. O conjunto de restrições (4) garantem que itens conflitantes não estarão presentes em um mesmo *bin*. Finalmente, as restrições indicadas em (5) e (6) dizem respeito ao domínio das variáveis de decisão.

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br

O somatório da variável y_k na função objetivo passa pelos *bins* 1 até n. Isso se dá pelo motivo de que n é a menor quantidade de *bins* necessárias para empacotar os itens. A formulação do problema dita que existem infinitos *bins* disponíveis para o empacotamento. Porém, no pior dos casos, cada item é empacotado em um *bin* sozinho.

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 e Öhring, 1997].

4. Método de Resolução

Para resolver o BPPC foi proposto um método baseado na meta-heurística *Simulated Annealing* (SA) apresentado por [Kirkpatrick et al., 1983]. Essa meta-heurística faz uma analogia ao processo físico de recozimento de metais, utilizado na metalurgia para transformação das propriedades físicas de um sólido através de um tratamento térmico. A meta-heurística trabalha como uma simulação desse processo de recozimento.

Nas seções a seguir apresentamos cada componente do método em detalhes.

4.1. 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. Cada elemento dentro de B_i representa um item empacotado. 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.

4.2. Solução Inicial

Foram desenvolvidos dois métodos construtivos para gerar uma solução inicial para o problema: o método guloso *Modified First Fit Decreasing* (MFFD), descrito por [Gendreau et al., 2004], e um método aleatório. Ambos métodos produzem soluções viáveis para o BPPC.

No método construtivo aleatório, 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 método *Next Fit* (NF) [Coffman et al., 2013], onde há apenas um *bin* aberto por vez. Seu pseudocódigo é descrito no Algoritmo 1.

Algoritmo 1: Pseudocódigo do método construtivo aleatório

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

```
S ← ∅;
bin ← ∅;
para cada item ∈ V em ordem aleatória faça
    se item couber em bin E não ter conflito com nenhum item do bin então
        adicionar item ao bin;
    senão
        adicionar bin à solução S;
        abrir novo bin e adicionar item;
retorna S;
```

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br

A entrada do Algoritmo 1 consiste em uma lista de itens V, a capacidade máxima C dos *bins*, 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 (linha 7). Em seguida, um novo *bin* é aberto para receber o item (linha 8). A solução construída é retornada na linha 9, após finalizado o empacotamento de todos os itens.

Diferentemente do construtivo aleatório, no método 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 2.

Algoritmo 2: Pseudocódigo do Modified First Fit Decreasing (MFFD)

```
Entrada: lista de itens V, capacidade dos bins C, grafo de conflito G(V,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      se item não coube em nenhum bin então
8          criar novo bin e adicionar item;
9          adicionar bin à solução S;
10 retorna S;
```

O método construtivo aleatório serve de entrada para o algoritmo, já que o SA se aproveita de soluções iniciais "ruins" em qualidade. Isso ocorre por causa da natureza do método, que começa sua busca piorando ou bagunçando a solução inicial, por aceitar soluções piores em qualidade. Com o passar das iterações e com o resfriamento da temperatura, o algoritmo vai convergindo para soluções melhores.

O método guloso, por seguir uma lógica *First Fit* (FF), resulta em soluções de maior qualidade inicial quando comparada à solução aleatória. Esta é a solução inicializada como a melhor solução do algoritmo SA. Ela é retornada caso o SA não encontre outra solução viável de menor custo.

4.3. Estruturas de Vizinhança

A exploração do espaço de busca é feita a partir da aplicação de diferentes estruturas de vizinhança. Uma série de movimentos são aplicadas à solução atual para gerar soluções vizinhas. As estruturas de vizinhança ($\mathcal{N}$) implementadas foram a *realocação*, *troca*, *destrói e reconstrói* e *adiciona*.

Realocação ($\mathcal{N}^1$) – um item aleatório de um *bin* qualquer é realocado para outro *bin* da solução.

Troca ($\mathcal{N}^2$) – 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.

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br

Destrói e Reconstrói ($\mathcal{N}^3$) – 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.

Adiciona ($\mathcal{N}^4$) – um novo *bin* é aberto e adicionado à solução, recebendo um item. A escolha do item é feita com foco em resolver os conflitos ou a capacidade ultrapassada de um dos *bins* da solução.

As estruturas de vizinhanças $\mathcal{N}^1$ e $\mathcal{N}^2$ são 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. A vizinhança destrói e reconstrói ($\mathcal{N}^3$), do inglês *ruin and recreate*, é uma estrutura de vizinhança que causa uma mudança profunda, ao tentar eliminar um *bin* inteiro da solução.

Uma nova vizinhança ($\mathcal{N}^4$) é proposta com o objetivo de possibilitar o aumento no número de *bins* de uma solução, oferecendo mais espaço para a resolução de violações de restrições. A solução vizinha gerada resulta da transferência de um item para um novo *bin* que é aberto especificamente para essa realocação.

Diferentemente da vizinhança de realocação aleatória, a escolha do item a ser movido em $\mathcal{N}^4$ é feita de forma direcionada, com o foco em resolver conflitos ou violações de capacidade em um dos *bins* existentes.

A movimentação ocorre da seguinte maneira: é verificado se, na solução, existe alguma violação de conflito (cenário 1), alguma violação da restrição de capacidade (cenário 2) ou se a solução é válida (cenário 3). Para cada um dos cenários, a escolha do item a ser realocado ao novo *bin* é diferente:

Cenário 1 identifica-se o *bin* com o maior número de conflitos. Dentre os itens deste *bin*, é selecionado aquele que gera mais conflitos com os demais.

Cenário 2 o *bin* com o maior excesso de peso é selecionado. O item escolhido para ser removido é aquele cuja retirada permite que o *bin* ultrapassado fique o mais próximo possível do limite de capacidade, sem ultrapassá-lo.

Cenário 3 quando a solução não possui violações, um item é selecionado aleatoriamente e transferido para o novo *bin*.

Todas as estruturas de vizinhança podem retornar soluções inviáveis. As estruturas $\mathcal{N}^3$ e $\mathcal{N}^4$, no entanto, podem levar uma solução inviável a um estado viável, resolvendo algum conflito entre itens ou a restrição de capacidade de algum *bin*. Elas podem, também, não conseguir resolver todos os conflitos existentes na solução, retornando ainda assim uma solução inviável.

As estruturas $\mathcal{N}^1$ e $\mathcal{N}^2$ podem levar uma solução viável a se tornar inviável, ao movimentar itens de maneira que resulte em alguma violação de restrição do problema. Estas duas estruturas também podem, ao acaso, resolver algum conflito na solução, ao movimentar itens aleatoriamente. O objetivo dessa mistura de estruturas de vizinhanças é diversificar a busca do algoritmo, na esperança de se encontrar soluções cada vez melhores.

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 à quantidade de vezes que ela obtém sucesso ao gerar um vizinho de melhor qualidade.

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br

4.4. Função de Avaliação

O método proposto percorre uma série de soluções durante sua execução, estas sendo soluções viáveis e inviáveis. Soluções podem 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. Uma solução viável é uma relação de itens e *bins* onde nenhuma restrição do problema é violada.

Devido à variedade de soluções (viáveis e inviáveis) que o algoritmo encontra, surge 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*.

$$fa(S) = \kappa_1 * bins + \kappa_2 * excessWeight + \kappa_3 * conflicts; \tag{7}$$

Na Equação (7), $bins$ é o número de *bins* utilizados na solução e $excessWeight$ 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 κ_1, κ_2 e κ_3 são associados, respectivamente, a cada um dos fatores descritos, influenciando diretamente o resultado da função de avaliação.

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.

4.5. 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 um limite inferior como ponto de referência ao avaliar soluções e interromper sua busca.

[Falkenauer, 1996] comenta sobre o número mínimo teórico de *bins*, obtido através da soma do peso de todos os itens dividida pela capacidade C de um *bin*. Este limite é denominado limite inferior trivial conforme Equação (8). Esse é o menor número de *bins* necessário para acomodar todos os itens, no contexto do BPP.

$$lb = \frac{\sum_{i=1}^{n} w_i}{C} \tag{8}$$

Importante notar que o limite inferior trivial não leva em consideração a restrição de conflitos entre itens. Tratando-se do BPPC, esse valor pode estar distante da real solução ótima global. No entanto, em instâncias com poucos conflitos entre os itens esse valor pode ser utilizado.

4.6. Pós-otimização

Com a finalidade de realizar uma última tentativa de melhorar a qualidade da solução, uma busca local é aplicada à solução retornada pelo SA como método de pós-otimização.

O método funciona de maneira simples, seguindo os seguintes passos: (i) os *bins* da solução são ordenados dos mais cheios até os mais vazios; (ii) para cada *bin* que contenha apenas um item, tenta-se realocar este item em outro *bin* da solução, seguindo uma ordem *First Fit* (FF) e respeitando as restrições de conflito e capacidade.

No melhor caso, todos os *bins* que possuíam um único item são esvaziados e retirados da solução. No pior caso, se nenhuma movimentação for possível de ser executada, a solução retornada é aquela pelo SA.

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br

4.7. O Método SA

O pseudocódigo do método proposto é representado pelo Algoritmo 3. 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 trivial do problema lb é alcançado. A melhor solução viável encontrada é retornada. A seguir detalhamos o funcionamento do método.

Algoritmo 3: Pseudocódigo do Simulated Annealing

Entrada: V, C, $G(V, E)$, T_i,α, T_{min}, SA_{max}, κ_1, κ_2, κ_3, $\mathcal{N}$
Saída: solução S^*

```
1  S ← solucaoAleatoria(V, C, G);
2  S* ← solucaoInicialMFFD(V, C, G);
3  T ← T_i;
4  iter_T ← 0;
5  lb ← lowerBound();
6  ρ ← {1};
7  enquanto T > T_min e S* > ⌈lb⌉ faça
8      enquanto iter_T < SA_max faça
9          N' ← selecionaVizinhança(N, ρ);
10         S' ← geraVizinho(S, N');
11         Δ ← fa(S') − fa(S);
12         se Δ < 0 então
13             S ← S';
14             ρ ← atualizaProbabilidade(ρ);
15             se S' é válida e fo(S') <= fo(S*) então
16                 S* ← S';
17         senão
18             x ← valorAleatorio[0, 1];
19             se x < e^(−Δ/T) então
20                 S ← S';
21         iter_T ← iter_T + 1;
22     T ← T × α;
23     iter_T ← 0;
24 S* ← posOtimizacao(S*);
25 retorna S*;
```

O Algoritmo 3 recebe, de entrada, o conjunto V de itens, a capacidade máxima dos *bins* C e o grafo de conflito G com vértices V e arestas E. Também são passados como parâmetro de entrada os valores de temperatura inicial T_i, fator de resfriamento α, temperatura mínima T_{min}, número máximo de repetições por temperatura SA_{max} e os fatores de penalidade κ_1, κ_2 e κ_3.

A solução que servirá de ponto de partida do método é gerada na linha 1, com o método construtivo aleatório. Uma solução construída pelo método construtivo MFFD é guardada em S^*

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br

na linha 2. Esta solução servirá de reserva, e será retornada caso nenhuma outra solução melhor seja encontrada durante a busca.

A temperatura inicial T e a variável $iter_T$ são inicializadas nas linhas 3 e 4, enquanto que o limite inferior é calculado na linha 5. A variável ρ, inicializada na linha 6, se refere ao peso de escolha de cada uma das vizinhanças. Esse peso é recalculado cada vez que uma vizinhança gerar uma melhora na solução. No início do algoritmo, todas as estruturas de vizinhanças possuem a mesma probabilidade de serem escolhidas.

O laço principal começa na linha 7 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 8-21). A cada iteração, uma estrutura de vizinhança é selecionada de acordo com uma probabilidade (linha 9) e uma solução vizinha S' é gerada (linha 10). Em seguida, o valor Δ é calculado, sendo a diferença dos valores da função de avaliação em S' e a solução atual S na linha 11.

Se a solução vizinha S' tiver o valor da função de avaliação melhor, ele se torna a solução corrente (linhas 12-13). Além disso, verifica-se se S' 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 15). Se este for o caso, S^* é atualizado na linha 16.

Se a solução vizinha tiver uma qualidade inferior que a solução atual, existe uma possibilidade de que S' seja aceita (linhas 18-20). Após SA_{max} iterações, a temperatura é reduzida de acordo com o fator de resfriamento e a variável $iter_T$ é reinicializada (linhas 22-23). Finalizado o laço principal do SA, o método de pós-otimização é executado (linha 24). Por fim, o algoritmo retorna a solução S^*, ou seja, a melhor dentre todas analisadas.

Um detalhe é adicionado no critério de atualização da melhor solução encontrada. Em vez de apenas verificar se a solução vizinha gerada utiliza menos *bins* que a melhor atual, é permitido que S' utilize a mesma quantidade de *bins*. Este é chamado de movimento lateral, onde a solução atual é atualizada para outra onde a função objetivo não aumenta nem diminui, ela se mantém igual. Essa variação permite uma melhor diversidade na busca.

5. Experimentos Computacionais

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.

A calibração dos valores dos pesos da função de avaliação foi feita utilizando o *Iterated racing for automatic algorithm configuration* (IRACE) [López-Ibáñez et al., 2016]. A faixa de valores para calibração dos fatores de penalidade κ foi o conjunto de números inteiros entre 1 e 500. A melhor configuração encontrada se deu com os fatores de penalidade κ_1, κ_2 e κ_3 iguais a 15, 338 e 38, respectivamente. Os demais parâmetros SA foram estabelecidos empiricamente. O SA_{max} foi definido como 1.000, temperatura inicial como 500, fator de resfriamento α igual a $0{,}98$ e temperatura mínima igual a $0{,}01$.

5.1. Instâncias de teste

O método SA desenvolvido foi aplicado em uma série de instâncias de teste da literatura, disponíveis em math.u-bordeaux.fr/ rsadykov/. Um resumo das informações sobre parâmetros das instâncias é apresentado na Tabela 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]. [Sadykov e Vanderbeck, 2013] geraram as

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br

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 1: Instâncias de teste do BPPC

instâncias (ta) e (ua) com as mesmas características das classes (t) e (u) de [Muritiba et al., 2010], mudando apenas o procedimento de geração dos grafos de conflito. Os autores observaram que o número médio de itens por *bin* nos casos de teste anteriores era pequeno (não excedia três). Com isso, dois novos grupos de instâncias mais difíceis foram gerados, (d) e (da). 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].

Para todas as instâncias utilizadas, todos os seus itens possuem peso menor que a capacidade C dos *bins*. Isso garante uma instância válida do BPPC.

Não são conhecidas as soluções ótimas de todas as instâncias de teste. Para 193 instâncias, nos referimos à melhor solução conhecida na literatura: *Best Known Solution* (BKS). Estas instâncias estão distribuídas nas classes (ua), (ta) e (da), sem confirmação de que a BKS é a solução ótima global. Tais dados foram obtidos através de comunicação pessoal com a Dra. Renatha Capua, que utilizou estes dados no seu trabalho de 2018 sobre o BPPC [Capua et al., 2018].

5.2. Resultados Obtidos

Os resultados computacionais são expostos e discutidos nesta seção. O método desenvolvido foi aplicado um total de 10 vezes em cada instância. A Tabela 2 apresenta os resultado dos testes, agrupando-os pela classe de instâncias e quantidade de itens. A coluna "Instância" aponta a classe daquele conjunto de casos de teste e a coluna "n" expõe a quantidade de itens a serem empacotados pelo algoritmo.

As seguintes métricas foram analisadas:

#opt. quantidade de instâncias nas quais a solução ótima, ou BKS, foram encontradas.

gap (%) desvio médio entre a melhor solução encontrada em relação à solução ótima, ou BKS.

T (s) tempo médio de execução do algoritmo, em segundos.

Soluções ótimas são encontradas diversas vezes com a aplicação do SA, em um total de 523 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 251, 136 e 64 vezes, respectivamente, dentro das 360 instâncias de cada grupo.

A qualidade geral das soluções podem ser analisadas pela métrica *gap (%)*, que mede a distância à solução ótima. O valor médio geral foi baixo, apenas $4,81\%$. Para a classe (d), esse valor se mostrou especialmente pequeno; a média obtida é menor que 1%. Isso mostra a flexibilidade do método, que alcançou soluções de ótima qualidade para instâncias de dimensões muito diferentes, desde 120 a 1.000 itens.

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br

Instância		**SA**		
Classe	n	*#opt.*	*gap (%)*	*T (s)*
u	120	40	2.18	87.42
	250	9	3.62	178.11
	500	1	4.04	412.44
	1.000	0	4.30	1338.83
t	60	72	0.97	36.69
	120	43	2.39	82.51
	249	17	3.42	168.04
	501	4	3.56	404.91
ua	120	0	4.89	69.52
	250	0	6.55	132.50
	500	0	7.16	358.38
	1.000	0	7.38	1296.21
ta	60	64	1.37	26.53
	120	0	5.71	44.76
	249	0	7.58	111.73
	501	0	8.06	284.96
d	120	85	0.26	76.45
	250	72	0.44	158.94
	500	59	0.54	398.53
	1.000	35	0.69	1333.95
da	120	16	7.37	52.89
	250	6	9.64	118.89
	500	0	11.17	333.44
	1.000	0	12.03	1250.35

Tabela 2: Resultados por dimensão das instâncias

O método se mostra mais eficiente em instâncias com poucos itens, para todas as classes de instâncias. É possível observar que o *gap (%)* aumenta junto com a quantidade de itens a serem empacotados pelo algoritmo.

Em relação ao tempo de execução do algoritmo, como era de se esperar, ele também 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 uma média de mais de 20 minutos. Tardamentos como estes são uma desvantagem em aplicações de larga escala que demandariam agilidade, ou a execução do algoritmo repetidas vezes. O elevado tempo de processamento se deve principalmente à utilização de vizinhanças pesadas como *destrói e reconstrói* e *adiciona*, que executam diversas passagens e verificações dentro das soluções e seus *bins*, além do método de pós otimização.

Ao comparar instâncias de grafo de conflito de intervalo e arbitrário, percebemos que (ua), (ta) e (da) foram mais difíceis de resolver do que as suas versões (u), (t) e (d).

6. Conclusão

Este trabalho se propôs a estudar métodos aproximados para resolver o problema de *Bin Packing* com Conflitos unidimensional, derivado do clássico *Bin Packing Problem*.

Para tratar o problema foi desenvolvido um método baseado na meta-heurística *Simulated Annealing* (SA). O SA utiliza quatro estruturas de vizinhanças para obter novas soluções a cada

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br

iteração. Experimentos computacionais foram empregados para se relatar a eficiência da metaheurística *Simulated Annealing* e os resultados são analisados e discutidos. O algoritmo implementado foi testado em um conjunto de instâncias de teste comumente referenciadas na literatura.

O desvio relativo médio em relação as melhores soluções conhecidas foi de apenas $4,81\%$, e a otimalidade foi alcançada em $24,21\%$ dos casos.

Em trabalhos futuros, pode ser interessante o estudo e aplicação de estruturas de vizinhança de larga escala como *Ejection Chains* e *Granade* [Capua et al., 2018]. Estas estruturas causam uma mudança drástica na solução, possibilitando uma maior diversificação durante a busca. Outra proposta seria a implementação de alguns mecanismos de reaquecimento ou recomeço no SA após uma série de iterações sem melhora na solução.

Referências

Capua, R., Frota, Y., Ochi, L. S., e Vidal, T. (2018). A study on exponential-size neighborhoods for the bin packing problem with conflicts. *Journal of Heuristics*, 24:667–695.

Coffman, E. G., Csirik, J., Galambos, G., Martello, S., Vigo, D., et al. (2013). Bin packing approximation algorithms: Survey and classification. In *Handbook of combinatorial optimization*, p. 455–531. Springer.

Delorme, M., Iori, M., e Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2:5–30.

Gendreau, M., Laporte, G., e Semet, F. (2004). Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research*, 31(3):347–358.

Glover, F. e Laguna, M. (1998). *Tabu search*. Springer.

Hamdi-Dhaoui, K., Labadie, N., e Yalaoui, A. (2014). The bi-objective two-dimensional loading vehicle routing problem with partial conflicts. *International Journal of Production Research*, 52 (19):5565–5582.

Jansen, K. e Öhring, S. (1997). Approximation algorithms for time constrained scheduling. *Information and computation*, 132(2):85–108.

Kirkpatrick, S., Gelatt Jr, C. D., e Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., e Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Muritiba, A. E. F., Iori, M., Malaguti, E., e Toth, P. (2010). Algorithms for the bin packing problem with conflicts. *Informs Journal on computing*, 22(3):401–415.

Sadykov, R. e Vanderbeck, F. (2013). Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS Journal on Computing*, 25(2):244–255.

Schneider, J. J. e Kirkpatrick, S. (2006). Ruin & recreate. *Stochastic Optimization*, p. 73–77.

DOI: 10.59254/sbpo-2025-212339
https://proceedings.science/p/212339?lang=pt-br