Pontifícia Universidade Católica de Goiás

Alysson Victor Almeida Souza

Lucas Teixeira Correia

GRASP e Simulated Annealing para Problema do Empacotamento Unidimensional

Goiânia

2024

Sumário

1. Fundamentação Teórica

1.1 Problema do Bin Packing Unidimensional

O problema do **Bin Packing** unidimensional (BPP, Bin Packing Problem) é um problema clássico de otimização combinatória que pertence à classe dos problemas NP-difíceis. No BPP, um conjunto de N itens, cada um com um peso W, deve ser distribuído em um número mínimo de caixas, ou bins, de capacidade C, de forma que a soma dos pesos dos itens alocados a cada caixa não exceda a capacidade da caixa.

Este problema tem uma ampla gama de aplicações práticas, desde logística e armazenagem, em que é necessário empacotar produtos em contêineres, até computação, como na alocação de recursos em múltiplos processos. Apesar de sua simplicidade conceitual, o Bin Packing é um problema onde sua solução ótima se torna inviável para instâncias grandes devido à sua complexidade combinatória, e, por isso, métodos heurísticos e metaheurísticos são frequentemente utilizados.

1.2. Algoritmos Metaheurísticos

Metaheurísticas são algoritmos de busca de propósito geral que exploram o espaço de soluções de problemas complexos, sem a necessidade de um modelo matemático completo do problema. Elas são particularmente úteis em problemas NP-difíceis, onde métodos exatos se tornam impraticáveis para grandes instâncias. Entre as principais metaheurísticas aplicadas ao Bin Packing estão o **GRASP** e o **Simulated Annealing**, ambos utilizados neste trabalho.

1.3. GRASP (Greedy Randomized Adaptive Search Procedure)

O **GRASP** é uma metaheurística iterativa que combina aleatoriedade com técnicas gulosas (greedy) para a construção de soluções. Uma das características principais deste algoritmo é a liberdade que o implementador tem para construir suas soluções, já que cada iteração do GRASP é dividida em apenas duas fases, sendo elas, construção da solução inicial e busca local. Essas duas fases podem ser realizadas de várias formas, nesta implementação criamos duas funções para a construção, uma para a busca gulosa (, mas com estratégias de composição das caixas de maneira aleatória, e outra onde é adicionado um parâmetro α que influencia diretamente a composição da RCL (lista de candidatos que podem ser inseridos na caixa). Após a construção da solução inicial, realiza-se uma busca local nesta solução com o intuito de melhorá-la.

Diante disso, para que um item x possa fazer parte da RCL na primeira função, basta que ele caiba na caixa. Já na segunda função, ocorre da seguinte forma:

$$x \in RCL \,\big|\, x \leq \big(Min + \alpha \cdot (Max - Min)\big)\big|$$

Neste caso, a utilização de um $\alpha = 1$ representa escolhas puramente aleatórias, ou seja, quanto mais este parâmetro se aproxima de 0, mais guloso se torna o algoritmo, levando em conta que com um α = 0, apenas o item com menor custo será aceito na RCL. Na equação, Min e Max representam respectivamente os itens com menor e maior peso que ainda não foram alocados.

Também implementamos duas heurísticas para lidar com a busca local, sendo elas: FI (First Improvement) e BI (Best Improvement). Ambas possuem estratégias semelhantes para lidar com a busca local, partindo do ponto que buscamos melhorar a solução inicial movendo itens de uma

caixa para outra de forma sequencial em todas as caixas. A diferença entre os dois está essencialmente no fato do FI interromper a busca local quando a primeira melhoria acontece, ou seja, um item é realocado de uma caixa para outra, excluindo a caixa de onde o item foi movido caso ela fique vazia. Já no BI, todas as melhorias possíveis são exploradas, aumentando, assim, a complexidade do algoritmo.

Segue abaixo o pseudocódigo do algoritmo:

Algoritmo GRASP

1 $S_{melhor} \leftarrow S_{inicial}\ (melhor\ solucao)$

2 $Para\ cada\ iteracao$:

3 $\quad S_{atual} \leftarrow GreedySolution\ ou\ GreedyRandomSolution$

4 $\quad S_{atual} \leftarrow BuscaLocal\ (S_{atual})$

5 $\quad Se\ S_{atual} < S_{melhor}$:

6 $\quad\quad S_{melhor} \leftarrow S_{atual}$

7 $retorne\ S_{melhor}$

Aplicado ao problema de Bin Packing, o GRASP tenta alocar os itens em caixas de maneira que o espaço nas caixas seja utilizado da forma mais eficiente possível, levando em consideração a capacidade das caixas e os pesos dos itens. O algoritmo pode gerar soluções rapidamente, mas há uma troca entre a qualidade da solução e a rapidez com que ela é gerada, já que o parâmetro α randomizado pode impedir que soluções de alta qualidade sejam encontradas consistentemente.

1.4. Simulated Annealing (SA)

O Simulated Annealing (SA) é uma metaheurística inspirada no processo de recozimento físico dos metais. A ideia central é a busca de soluções através da aceitação de soluções piores com uma certa probabilidade, controlada por uma função de temperatura que diminui ao longo do tempo. Essa técnica permite que o algoritmo escape de mínimos locais, com o objetivo de encontrar soluções globais de melhor qualidade.

Partindo da S_{atual} inicia os procedimentos que dependem de alguns parâmetros, a temperatura inicial $T_{inicial}$, que é atualizada a cada iteração sendo multiplicada pela taxa de resfriamento α .

Algoritmo Simulated Annealing

1 $S_{atual} \leftarrow S_{inicial}(solucao\ inicial)$

2 $S_{melhor} \leftarrow S_{atual}\ (melhor\ solucao)$

3 $T_{atual} \leftarrow T_{inicial}\ (temperatura\ inicial)$

4 $T_{final}\ (temperatura\ final)$

5 $\alpha\ (Taxa\ de\ resfriamento)$

6 $Enquanto\ T_{atual} > T_{final}$:

7 $N_{solucao} \leftarrow randomized(S_{atual})$

8 $\Delta_s = N_{solucao} - S_{atual}\ (taxa\ de\ variacao\ das\ solucoes)$

9 $Se\ \Delta_s < 0\ ou\ random(0,1) < e^{\frac{-\Delta_s}{T_{atual}}}$

10 $S_{atual} \leftarrow N_{solucao}$

11 $se\ S_{atual} < S_{melhor}$:

12 $S_{melhor} \leftarrow S_{atual}$

13 $T_{atual} \leftarrow T_{atual} \times \alpha\ (atualiza\ a\ Temperatura)$

14 $retorne\ S_{melhor}$

A função $e^{\frac{-\Delta_s}{T_{atual}}}$ **decresce exponencialmente** conforme Δ_s (o "custo" de aceitar a pior solução) aumenta. Quando a temperatura é alta, o valor de $\frac{-\Delta_s}{T_{atual}}$ é pequeno, então $e^{\frac{-\Delta_s}{T_{atual}}} \approx 1$, ou seja, temos uma alta probabilidade de aceitar a nova solução, já na temperatura baixa, $e^{\frac{-\Delta_s}{T_{atual}}}$ tende a um valor muito próximo de 0, reduzindo a chance de aceitar soluções piores.

No contexto do Bin Packing, o SA utiliza uma solução inicial (gerada, por exemplo, pelo GRASP ou por heurísticas como Best Fit ou First Fit) e, a partir dela, realiza perturbações (modificações aleatórias) para explorar o espaço de soluções. A cada iteração, executamos o algoritmo Best Fit ou First Fit na solução vizinha para avaliar a qualidade da nova solução, se a nova solução encontrada for melhor que a anterior, ela é aceita. Caso contrário, ela pode ser aceita com uma probabilidade que depende da temperatura e da diferença de qualidade entre as soluções.

A ideia principal do Best Fit é alocar cada item na posição que melhor aproveita o espaço, de modo a minimizar o desperdício. Para cada novo item, verifica se ele cabe em alguma caixa já existente. Se houver várias operações possíveis, ele escolhe a caixa que deixa o menor espaço sobrando após a adição do item.

O SA é eficiente para escapar de mínimos locais, mas sua eficiência depende muito da definição dos parâmetros, como a temperatura inicial, a taxa de resfriamento (alfa) e o critério de parada.

Por outro lado, o Simulated Annealing mostrou-se mais eficaz em encontrar soluções de alta qualidade sem a necessidade de uma solução inicial sofisticada, bastando uma heurística simples como o Best Fit para gerar essa solução. As perturbações na solução eram realizadas através do embaralhamento da ordem dos itens e permitia uma exploração eficiente do espaço de busca.

1.5. Hibridização de GRASP com SA

Diante das dificuldades encontradas com a abordagem de DQN, optou-se por uma hibridização entre GRASP e Simulated Annealing. O GRASP foi utilizado para gerar uma solução inicial rapidamente, com a expectativa de que ela servisse de bom ponto de partida para o SA. No entanto, após a execução dos testes, constatou-se que, para instâncias maiores, o impacto da solução inicial gerada pelo GRASP no desempenho do SA era pequeno, com um custo de tempo adicional significativo.

Algoritmo Hibridizado

1 $S_{atual} \leftarrow S_{inicial}(solucao\ inicial)$

2 $S_{melhor} \leftarrow GRASP(S_{atual})$
$(chama\ meu\ algoritmo\ GRASP\ para\ retornar\ melhor\ solucao)$

3 $T_{atual} \leftarrow T_{inicial}\ (temperatura\ inicial)$

4 $T_{final}\ (temperatura\ final)$

5 α *(Taxa de resfriamento)*

6 $Enquanto\ T_{atual} > T_{final}$:

7 $N_{solucao} \leftarrow randomized(S_{atual})$

8 $\Delta_s = N_{solucao} - S_{atual}\ (taxa\ de\ variacao\ das\ solucoes)$

9 $Se\ \Delta_s < 0\ ou\ random(0,1) < e^{\frac{-\Delta_s}{T_{atual}}}$

10 $S_{atual} \leftarrow N_{solucao}$

11 $se\ S_{atual} < S_{melhor}$:

12 $S_{melhor} \leftarrow S_{atual}$

13 $T_{atual} \leftarrow T_{atual} \times \alpha\ (atualiza\ a\ Temperatura)$

14 $retorne\ S_{melhor}$

1.7. Deep Q-Network (DQN)

A ideia inicial deste trabalho envolvia a utilização de **Deep Q-Networks (DQN)**, uma técnica de aprendizado por reforço profundo. O DQN combina a abordagem de Q-learning com redes neurais profundas, permitindo que o agente de aprendizado por reforço possa lidar com grandes espaços de estado. No caso do Bin Packing, o DQN seria treinado para aprender a melhor forma de alocar os itens nas caixas, melhorando iterativamente a solução.

No entanto, após uma análise mais detalhada, observou-se que o treinamento do DQN para este problema específico seria muito custoso em termos de tempo de execução, especialmente para instâncias grandes, como aquelas com mais de 1000 itens. Além disso, o DQN requer uma quantidade significativa de recursos computacionais e dados de treinamento adequados para alcançar bons resultados.

2. Implementação e Adaptações

2.1. Linguagem de Programação e Ferramentas Utilizadas

A implementação foi realizada em **Python**, utilizando bibliotecas nativas como a biblioteca Random, Time e Matplotlib para, respectivamente, lidar com características aleatórias dos algoritmos, calcular o tempo de execução dos algorítmos e para a geração de gráficos. A simplicidade dessa abordagem garantiu controle sobre o comportamento dos algoritmos sem a necessidade de dependências adicionais.

2.2. Estruturação do Código

Link das Implementações (Github)

Definição do Problema

O problema de Bin Packing consiste em alocar um conjunto de itens com pesos variáveis em um número mínimo de caixas, cada uma com capacidade limitada. A função objetivo busca minimizar o número de caixas necessárias para armazenar todos os itens, respeitando a capacidade de cada uma.

Função de Custo

A função de avaliação do algoritmo, também conhecida como função de custo, calcula a quantidade de caixas necessárias para acomodar a configuração atual dos itens a cada iteração do **SA**. Esta função constrói as caixas de acordo com o **algoritmo Best Fit**, onde os itens são distribuídos nas caixas da maneira mais eficiente possível.

Inicialização da Solução

O algoritmo inicia gerando uma solução inicial a partir do Best Fit. Esta solução serve como ponto de partida para o processo iterativo de exploração que o Simulated Annealing realiza ao longo do tempo.

Perturbação da Solução

A perturbação da solução atual é implementada utilizando a função que embaralha os itens de maneira aleatória. Essa abordagem visa explorar novas regiões do espaço de soluções e, assim, evitar que o algoritmo fique preso em mínimos locais.

Critério de Aceitação

Se a nova solução resultar em uma menor quantidade de caixas, ela é automaticamente aceita. Caso contrário, a solução é aceita com um certo grau de aleatoriedade envolvendo a seguinte probabilidade:

$$P(\Delta_s) = e^{\frac{-\Delta_s}{T_{atual}}}$$

onde Δ_s representa a diferença de custo entre a solução atual e a solução proposta, e T_{atual} é a temperatura. Este critério permite que o algoritmo aceite temporariamente soluções piores, evitando mínimos locais, especialmente nas fases iniciais, quando a temperatura ainda é alta.

Controle da Temperatura

A temperatura inicial e o fator de resfriamento α foram ajustados de forma a equilibrar o tempo de execução do algoritmo em diferentes tamanhos de instâncias, garantindo que ele fosse eficiente tanto em instâncias menores quanto em maiores.

Resumo da Estrutura

1. Função de Custo: Avalia a solução atual em termos do número de caixas necessárias.
2. Perturbação da Solução: A solução é modificada embaralhando os itens, o que permite explorar diferentes configurações.
3. Critério de Aceitação: Baseado na probabilidade de Metropolis, aceitando soluções piores no início para escapar de mínimos locais.
4. Resfriamento: A temperatura é gradualmente reduzida, diminuindo a probabilidade de aceitar soluções piores e levando o algoritmo a convergir para uma solução ótima.

3. Variações Testadas

3.1. Abordagem Inicial com Deep Q-Network (DQN)

Ainda que a implementação do DQN tenha apresentado resultados positivos para as instâncias, se aproximando muitas das vezes do resultado ótimo, os problemas dessa abordagem logo ficaram claros conforme a quantidade de itens aumentava, constatando dificuldade para lidar com diferentes instâncias, por dois motivos: A necessidade de um treinamento do modelo para cada instância e o tempo necessário para lidar com grandes instâncias.

Embora o DQN tivesse potencial para melhorar o ponto de partida do Simulated Annealing, o custo em termos de tempo de treinamento inviabilizou sua utilização prática no projeto. Assim, optamos por não seguir com o DQN e focar em métodos heurísticos para gerar soluções iniciais, seguido de uma metaheurística para refinamento da solução.

3.2. Hibridização GRASP e SA

A implementação foi dividida em duas partes principais: o GRASP, responsável por gerar uma solução inicial, e o SA, que realiza o refinamento da solução. As partes foram modularizadas para facilitar testes independentes e ajustes de parâmetros.

- **GRASP**: Implementado para gerar soluções iniciais.

- **Simulated Annealing**: O algoritmo de SA foi implementado para receber uma solução inicial (fornecida pelo GRASP ou pelo Best Fit) e realizar perturbações aleatórias na solução.

A hibridização entre **GRASP** e **Simulated Annealing** foi inicialmente utilizada para melhorar o refinamento de soluções. O GRASP gerava uma solução inicial rápida, que seria refinada pelo SA. No entanto, observou-se que, para instâncias maiores, o GRASP não oferecia uma melhoria significativa na qualidade da solução final, apenas adicionando tempo ao processo sem um ganho relevante.

3.3. Simulated Annealing com Perturbações Aleatórias

O algoritmo de SA faz uso da aleatoriedade para perturbar as soluções, embaralhando o vetor de itens antes de passar para o Best Fit. Essa aleatoriedade é essencial para permitir a exploração do espaço de soluções, permitindo ao algoritmo que, muitas das vezes, escape de mínimos locais.

3.4. Decisão Final: Apenas Simulated Annealing

Testes comparativos foram realizados entre GRASP, Simulated Annealing, e a hibridização dos dois. A hibridização produziu resultados comparáveis ao SA isolado, logo, pelo tempo de execução adicional, a hibridização foi descartada. O SA, utilizando o Best Fit para gerar a solução inicial, foi a abordagem com maior custo-benefício em relação ao tempo e qualidade de solução.

4. Justificativa para as Escolhas

4.1. Escolha do Simulated Annealing como Algoritmo Principal

Após os testes da hibridização do GRASP com o SA, ficou claro que o SA seria o algoritmo principal utilizado. O GRASP foi inicialmente considerado por sua capacidade de gerar soluções rápidas e diversificadas, mas, para instâncias maiores, o tempo de execução adicional não trazia uma melhoria significativa na qualidade das soluções finais.

O Simulated Annealing mostrou-se uma escolha mais eficaz, pois, além de sua flexibilidade para perturbar e refinar soluções, ele foi capaz de encontrar boas soluções sem a necessidade de uma solução inicial complexa. Ao utilizar apenas uma heurística simples para gerar a solução inicial, o SA partia de uma base razoável e, por meio da função de perturbação, conseguia escapar de mínimos locais e melhorar a solução ao longo das iterações, sem a necessidade de uma segunda metaheurística, que acabaria aumentando a complexidade do algoritmo desnecessariamente.

4.2. Uso da Função shuffle para Perturbação de Soluções

Inicialmente, a estratégia de perturbação no Simulated Annealing envolvia a troca de apenas duas posições aleatórias no vetor de itens. A ideia era gerar pequenas mudanças na configuração da solução para explorar o espaço de soluções de maneira controlada. No entanto, essa abordagem mostrou-se limitada, pois não conseguia explorar adequadamente o espaço completo de soluções que o problema de Bin Packing oferece e os resultados mostraram pouca melhoria nas soluções finais.

O problema de Bin Packing tem um espaço de soluções enorme, com um número de possibilidades que cresce fatorialmente em relação ao número de itens (o que pode ser descrito como N!, onde N é o número de itens). Trocar apenas duas posições do vetor não permitia uma exploração eficiente desse vasto espaço.

Diante disso, optou-se por uma abordagem mais abrangente, utilizando a função shuffle da biblioteca Random para embaralhar completamente todos os itens no vetor a cada perturbação. Essa mudança possibilitou uma exploração muito mais ampla e eficaz do espaço de soluções, permitindo ao Simulated Annealing escapar de mínimos locais com mais facilidade e encontrar soluções melhores em menos iterações.

4.3. Configuração dos Parâmetros do Simulated Annealing

Os parâmetros do Simulated Annealing, como a temperatura inicial, o fator de resfriamento α e a temperatura final, foram configurados com o objetivo de equilibrar a eficiência do algoritmo em diferentes tamanhos de instâncias. Testes mostraram que essas configurações eram essenciais para garantir que o algoritmo tivesse um desempenho consistente tanto em instâncias pequenas quanto grandes.

A temperatura inicial foi ajustada de forma que o algoritmo tivesse flexibilidade para explorar soluções alternativas, sem se prender a mínimos locais logo no início. O fator de resfriamento foi escolhido para garantir uma redução gradual da temperatura, mantendo a capacidade de aceitação de soluções piores por mais tempo nas fases iniciais, enquanto permitia uma maior precisão nas fases finais. Essas escolhas foram baseadas na necessidade de manter o algoritmo eficiente e evitar tempos de execução prolongados, especialmente em instâncias grandes, como as de 1000 itens.

5. Parâmetros: Configurações Específicas do Algoritmo e Detalhes dos Experimentos

5.1. Temperatura Inicial

A temperatura inicial no algoritmo de Simulated Annealing foi configurada para 1000. Esse valor foi escolhido para garantir que o algoritmo tivesse flexibilidade para explorar diferentes soluções e aceitar temporariamente soluções subótimas no início do processo de resfriamento. Isso permitiu uma exploração eficaz do espaço de soluções, especialmente para instâncias grandes.

5.2. Fator de Resfriamento α

O fator de resfriamento foi definido como 0.95, o que resulta em uma diminuição gradual da temperatura ao longo das iterações. Esse valor foi escolhido para equilibrar a exploração e o refinamento da solução.

5.3. Critério de Parada

O critério de parada do algoritmo foi configurado de forma que a execução continue enquanto a temperatura for maior que 1. Isso proporcionou um controle eficiente do tempo de execução do algoritmo.

5.4. Experimentos Realizados

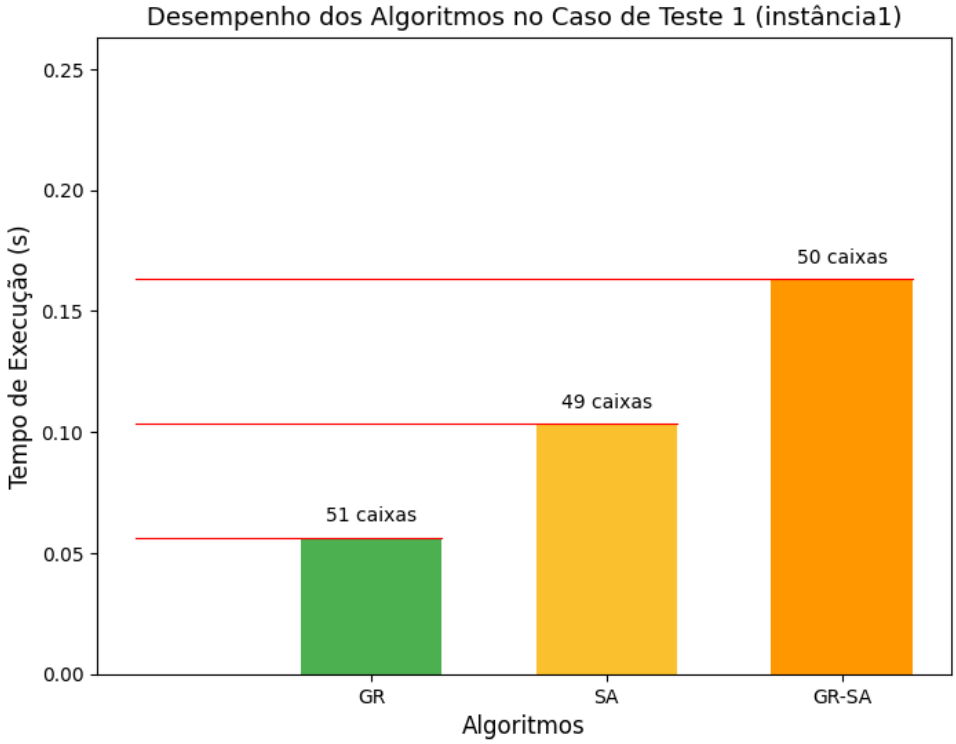
Os experimentos utilizaram os parâmetros descritos acima em diferentes instâncias do problema de Bin Packing, e os resultados mostraram uma boa combinação entre tempo de execução e qualidade das soluções, especialmente nas instâncias maiores. O desempenho foi monitorado ao longo de 3 instancias com quantidade de itens diferentes.

6. Resultados: Desempenho do Algoritmo em Diferentes Instâncias

6.1. Desempenho em Instâncias Pequenas

Nas instâncias menores, como as contidas na instancia1.txt (figura fornecida), o Simulated Annealing apresentou um ótimo desempenho em termos de tempo de execução e qualidade das soluções. Com até 100 itens, o número de iterações foi estimado em cerca de 140.

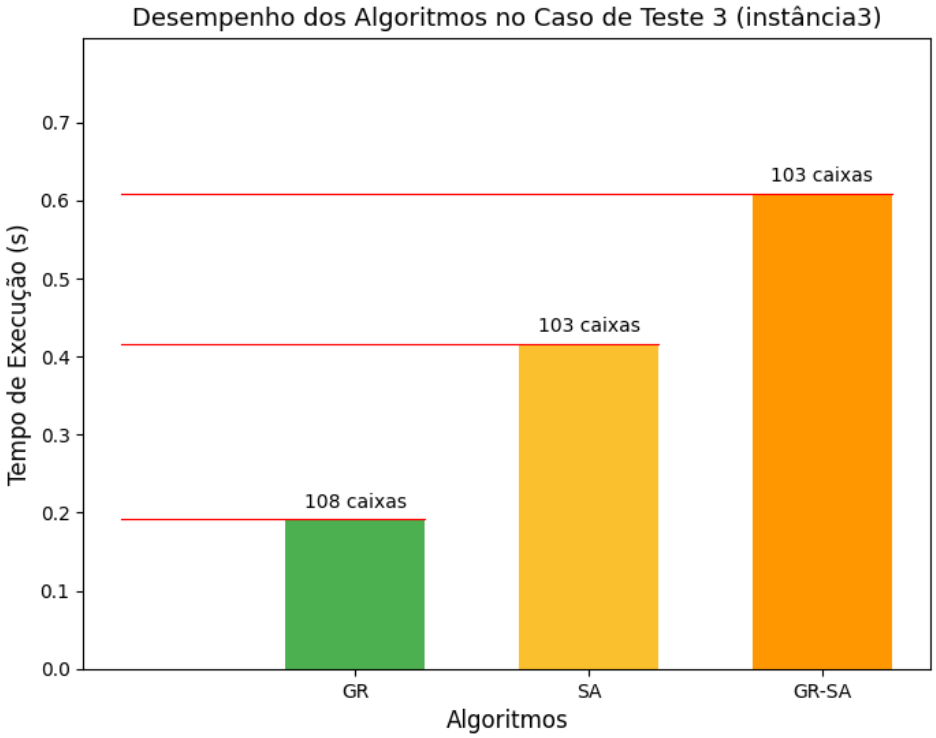
- GR (GRASP): O GRASP foi o mais rápido dos três algoritmos, como mostrado no gráfico, com um tempo de execução de aproximadamente 0.05 segundos, mas a solução encontrada (51 caixas) foi inferior ao SA e GR-SA.
- GR-SA (GRASP com Simulated Annealing): Embora a hibridização de GRASP com SA tenha encontrado uma solução um pouco melhor que o GR (50 caixas), o tempo de execução foi o maior de todos, chegando a aproximadamente0.17 segundos, o que o torna menos eficiente para a mesma qualidade de solução que o SA isolado.



6.2. Desempenho em Instâncias Médias

Para instâncias médias, como as do arquivo instancia3.txt com 250 itens, o desempenho do Simulated Annealing foi eficiente. O tempo de execução foi intermediário, cerca de 0.42 segundos, mas o algoritmo conseguiu encontrar uma solução de 103 caixas.

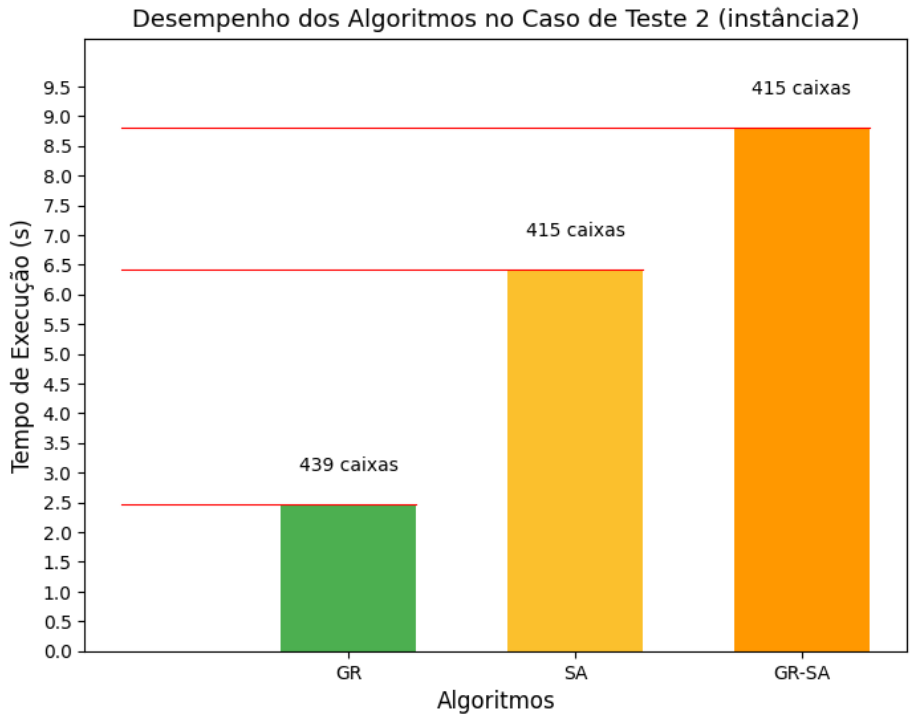
- GR (GRASP): O GRASP foi mais rápido, com um tempo de execução de cerca de 0.20 segundos, mas encontrou uma solução de 108 caixas, que foi inferior tanto ao SA quanto ao GR-SA.
- GR-SA (GRASP com Simulated Annealing): A hibridização de GRASP com SA levou mais tempo, com 0.60 segundos, e o número de caixas encontradas foi o mesmo que o SA isolado, 103 caixas, o que sugere que a hibridização não trouxe ganhos significativos em qualidade, mas apenas aumento no tempo de execução.



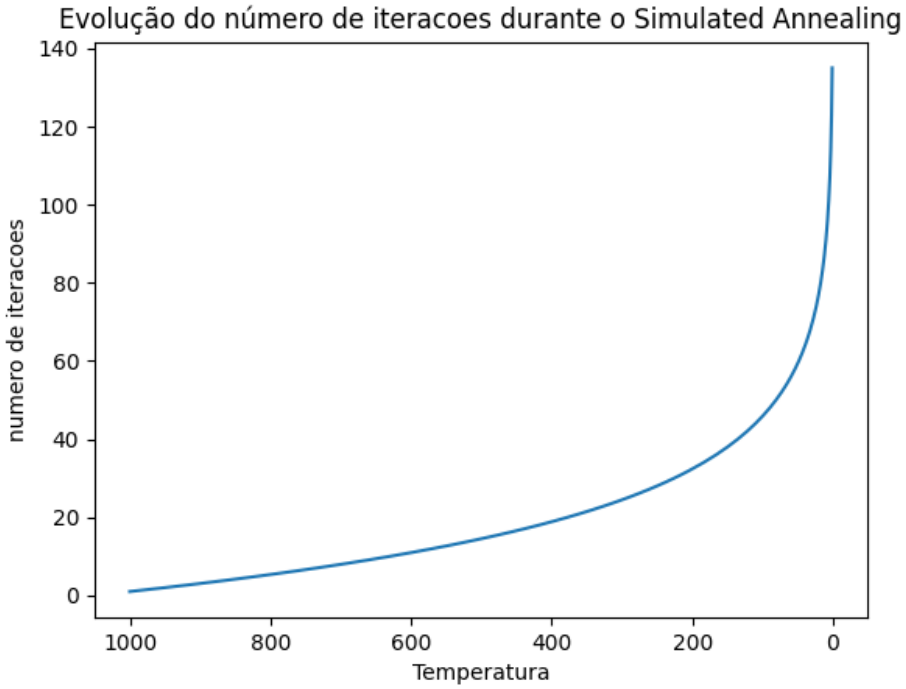
6.3. Desempenho em Instâncias Grandes

Nas instâncias grandes, como as contidas no arquivo instancia2.txt, com 1000 itens.O tempo de execução continuou sendo satisfatório e a solução encontrada de 415 caixas demonstrou a eficácia do algoritmo em explorar o espaço de soluções em uma instância mais complexa.

- GR (GRASP): O GRASP novamente foi mais rápido, com um tempo de cerca de 2.5 segundos, mas encontrou uma solução de 439 caixas, claramente inferior ao SA e ao GR-SA.
- GR-SA (GRASP com Simulated Annealing): A hibridização de GRASP com SA demorou significativamente mais, com 9 segundos, e encontrou a mesma solução que o SA isolado, de 415 caixas.



A seguir o gráfico relacionando o número de iterações com a temperatura:

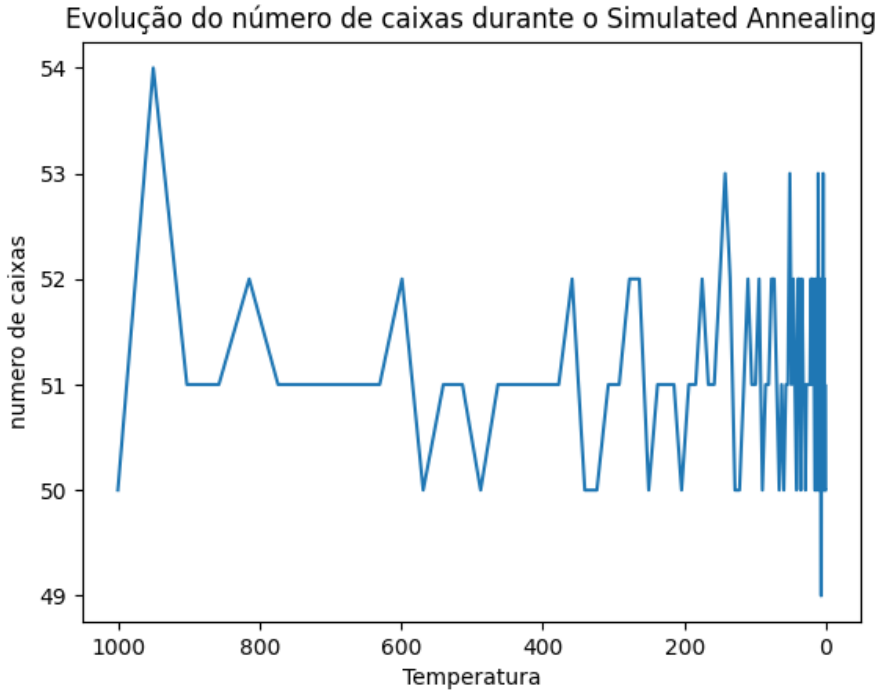


Como os parâmetros foram os mesmos (temperatura inicial, taxa de resfriamento) o número de iterações do algoritmo foi o mesmo em todas as instâncias.

6.4.Resultados: Evolução da Quantidade de Caixas durante o Simulated Annealing

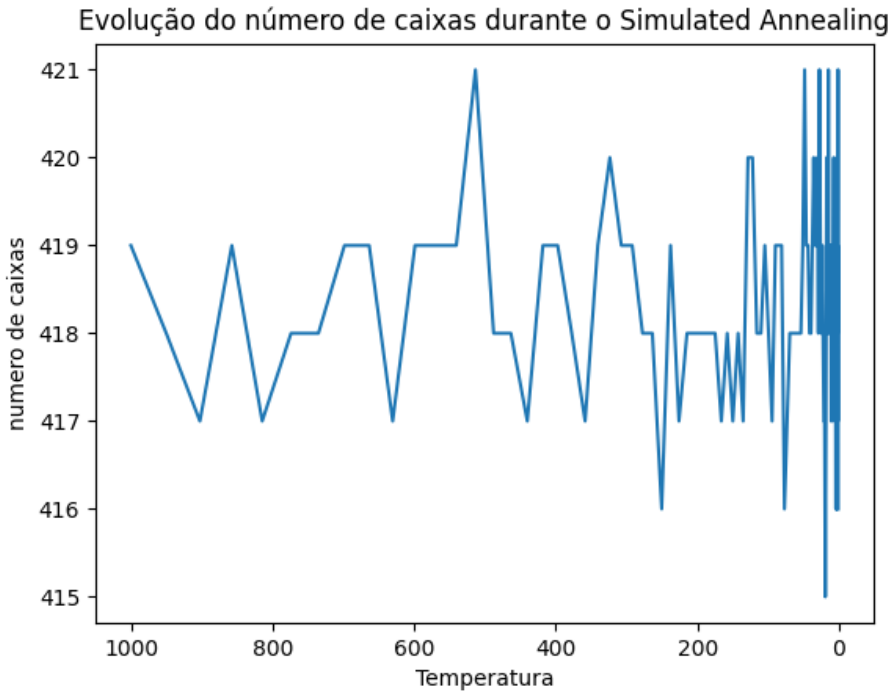
Instância 1

No gráfico da Instância 1, observamos que o número de caixas varia entre 49 e 54 caixas ao longo do processo, sem se estabilizar completamente, mesmo quando a temperatura se aproxima de zero. Isso reflete a natureza do Simulated Annealing, onde mesmo em temperaturas mais baixas, o algoritmo continua a explorar variações nas soluções.



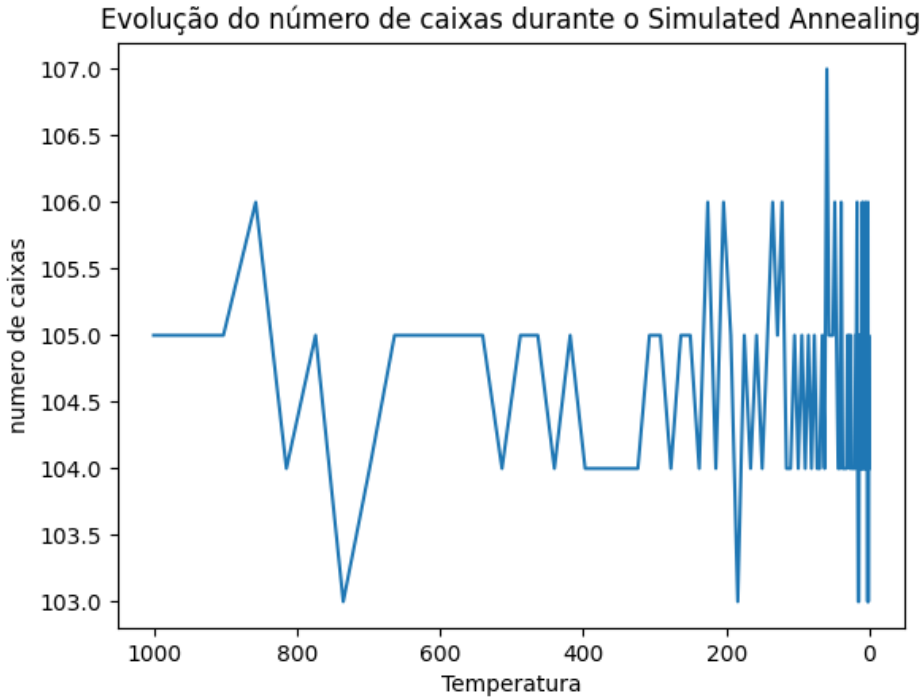
Instância 2

No gráfico da Instância 2 (instância grande com 1000 itens), o número de caixas oscila entre 415 e 421 caixas. Assim como na Instância 1, não há uma estabilização completa, com variações persistindo até as últimas iterações, mesmo quando a temperatura já está baixa.



Instância 3

O gráfico da Instância 3 (com 250 itens) mostra uma variação entre 103 e 107 caixas ao longo do processo, novamente sem uma estabilização clara. O comportamento do SA aqui reflete a mesma tendência observada nas outras instâncias: a aleatoriedade continua a desempenhar um papel, mesmo em temperaturas mais baixas, resultando em flutuações no número de caixas.



Conclusão sobre os Gráficos

Em todas as instâncias, o comportamento do SA mostra que o número de caixas não se estabiliza completamente, com variações até as últimas iterações. Essa flutuação é característica do SA, que continua explorando o espaço de soluções e fazendo pequenas perturbações, até mesmo em

temperaturas mais baixas. O algoritmo não busca convergência para uma solução, mas sim uma solução melhor, mantendo a possibilidade de explorar variações até o final.

7. Discussão

Os resultados apresentados nas seções anteriores mostram o comportamento do Simulated Annealing (SA) nas diferentes instâncias do problema do Empacotamento Unidimensional. A seguir, discutimos os pontos mais importantes observados nos experimentos.

7.1. Simulated Annealing vs. GRASP

Um dos aspectos mais notáveis dos resultados foi a diferença de desempenho entre o Simulated Annealing e o GRASP. Embora o GRASP tenha sido significativamente mais rápido na geração de soluções, especialmente em instâncias maiores, a qualidade das soluções obtidas foi inferior ao SA em todas as instâncias.

- **Qualidade da Solução**: O **SA** foi mais eficaz na minimização do número de caixas, com uma diferença de até **20 caixas** em instâncias maiores, como visto na **Instância 2**. Isso demonstra que, enquanto o **GRASP** é útil para gerar soluções rápidas, sua capacidade de explorar o espaço de soluções de forma aprofundada é limitada, e ele tende a fornecer soluções mais distantes do ótimo.
- **Tempo de Execução**: O **GRASP** foi mais rápido em todas as instâncias, com tempos de execução significativamente menores do que o **SA**. Isso reflete a natureza do algoritmo, que é guloso e aleatório, mas não realiza uma busca refinada das soluções.

7.2. Simulated Annealing: Exploração vs. Exploração Refinada

Os gráficos mostraram que o Simulated Annealing é altamente eficaz em explorar o espaço de soluções, mas, em vez de estabilizar em uma única solução ótima, ele continua a explorar variações até o final do processo.

- **Variação no Número de Caixas**: Em todos os gráficos, o número de caixas continuou a flutuar até a última iteração, mesmo quando a temperatura estava baixa. Isso indica que o SA continua a aceitar pequenas perturbações nas soluções. No entanto, isso também sugere que o SA poderia ser ajustado para parar de perturbar as soluções em fases mais tardias, como quando não há melhoras nas soluções após uma grande quantidade de iterações.
- **Benefícios de Explorar até o Final**: A vantagem de manter essa aleatoriedade até o final é que o SA consegue evitar mínimos locais, explorando melhor o espaço de soluções em busca de melhorias. Embora o número de caixas não tenha estabilizado completamente, o algoritmo foi capaz de encontrar boas soluções que superaram o GRASP.

7.3. Hibridização GRASP-SA

A hibridização entre o GRASP e o Simulated Annealing foi testada com a expectativa de que ela combinaria o melhor de ambos: a rapidez do GRASP e o refinamento do SA. No entanto, os resultados mostraram que essa combinação não trouxe os benefícios esperados.

- **Desempenho**: O tempo de execução do GR-SA foi maior do que o do SA isolado, enquanto a qualidade das soluções permaneceu a mesma. Isso indica que o tempo extra gasto pelo GRASP para gerar uma solução inicial não contribuiu significativamente para a melhoria da solução final, especialmente em instâncias maiores, como a **Instância 2**.
- **Conclusão sobre Hibridização**: A hibridização se mostrou ineficaz, pois não trouxe melhorias em termos de qualidade da solução, mas apenas adicionou um custo computacional adicional. Isso sugere que, para problemas como o Bin Packing, o Simulated Annealing isolado é uma abordagem mais eficiente do que combinar o GRASP e o SA.

7.4. Ajustes no Simulated Annealing

Com base nos resultados, algumas considerações podem ser feitas para ajustar o Simulated Annealing:

- **Taxa de Resfriamento**: A taxa de resfriamento pode ser ajustada dependendo do trade-off entre o tempo de execução e a qualidade da solução. Uma taxa de resfriamento mais alta, como a de **0.95** usada, oferece mais chances de encontrar soluções melhores, mas aumenta o tempo de execução. Por outro lado, uma taxa de resfriamento mais baixa reduziria o número de iterações, levando a soluções mais rápidas, mas potencialmente menos otimizadas.
- **Critério de Parada**: O critério de parada poderia ser ajustado pela temperatura final, como já implementado. Uma temperatura final mais alta permitiria que o algoritmo encerrasse mais cedo, enquanto uma temperatura final mais baixa permitiria que o algoritmo continuasse explorando, potencialmente encontrando soluções melhores com mais tempo de execução.

8.Referências Bibliográficas

Festa, P., Resende, M.G.C. (2016). **GRASP**. In: Martí, R., Panos, P., Resende, M. (eds) Handbook of Heuristics. Springer, Cham. https://doi.org/10.1007/978-3-319-07153-4_23-1

Resende, M.G.C., Ribeiro, C.C. (2014). **GRASP: Greedy Randomized Adaptive Search Procedures**. In: Burke, E., Kendall, G. (eds) Search Methodologies. Springer, Boston, MA. https://doi.org/10.1007/978-1-4614-6940-7_11

Resende, M.G.C., Ribeiro, C.C. (2019). **Greedy Randomized Adaptive Search Procedures: Advances and Extensions**. In: Gendreau, M., Potvin, JY. (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol 272. Springer, Cham. https://doi.org/10.1007/978-3-319-91086-4_6

Delahaye, D., Chaimatanan, S., Mongeau, M. (2019). **Simulated Annealing: From Basics to Applications**. In: Gendreau, M., Potvin, JY. (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol 272. Springer, Cham. https://doi.org/10.1007/978-3-319-91086-4_1

Munien, Chanaleä & Mahabeer, Shiv & Dzitiro, Esther & Singh, Sharad & Zungu, Siluleko & Ezugwu, Absalom. (2020). **Metaheuristic Approaches for One-Dimensional Bin Packing Problem: A Comparative Performance Study**. IEEE Access. PP. 10.1109/ACCESS.2020.3046185.

Maru, 2020. P, NP, **NP Hard and NP Complete Problem** | Reduction | NP Hard and NP Compete | Polynomial Class.

Dowsland, K. (1993). **Some experiments with simulated annealing techniques for packing problems**. European Journal of Operational Research, 68, 389-399. https://doi.org/10.1016/0377-2217(93)90195-S.

Loh, K., Golden, B., & Wasil, E. (2008). **Solving the one-dimensional bin packing problem with a weight annealing heuristic**. Comput. Oper. Res., 35, 2283-2291. https://doi.org/10.1016/j.cor.2006.10.021.

Haouari, M., & Serairi, M. (2009). **Heuristics for the variable sized bin-packing problem**. Comput. Oper. Res., 36, 2877-2884. https://doi.org/10.1016/j.cor.2008.12.016.

Fleszar, K., & Hindi, K. (2002). **New heuristics for one-dimensional bin-packing**. Comput. Oper. Res., 29, 821-839. https://doi.org/10.1016/S0305-0548(00)00082-4.