

Estudando Variações do Generalized Simulated Annealing no Problema da Mochila com Penalidades

Gabriel Souto, Luidi Simonetti, Pedro Henrique González

Universidade Federal do Rio de Janeiro

R. Antônio Barros de Castro, 119 - Cidade Universitária, Rio de Janeiro - RJ, 21941-853

{soutog, luidi, pegonzalez}@cos.ufrj.br

RESUMO

Otimizar funções multidimensionais é uma tarefa desafiadora em diversos campos de estudo, principalmente quando se trata de funções não convexas. Embora os métodos convencionais possam lidar com funções convexas, eles enfrentam dificuldades com as não-convexas. O *Classical Simulated Annealing* (CSA) é uma metaheurística que tem mostrado grande potencial em encontrar boas aproximações para o ótimo global dentro de um período de tempo razoável, por isso este trabalho visa utilizar o *Generalized Simulated Annealing* (GSA), uma variação do CSA, que tem mostrado bons resultados quando hibridizado com outras técnicas. Neste estudo, o Problema da Mochila com Penalidades (KPF) foi utilizado como objeto de estudo, adicionando complexidade à tarefa de otimização. A partir da análise comparativa das três variações propostas do GSA, nomeadamente GSA original, GSA-LS, GSA-CS, mostrou-se por meio de experimentos computacionais que a distância ficou a 3,5% da melhor solução conhecida para o problema.

PALAVRAS CHAVE. Problema da Mochila com Penalidades, Recozimento Simulado, Metaheurísticas.

ABSTRACT

Optimizing multidimensional functions is a challenging task in several fields of study, especially regarding non-convex functions. Although conventional methods can handle convex functions, they face difficulties with non-convex ones. *Classical Simulated Annealing* (CSA) is a metaheuristic that has shown great potential in finding good approximations to the global optimum within a reasonable time, so this work aims to use *Generalized Simulated Annealing* (GSA), a variation of CSA, which has shown good results when hybridized with other techniques. In this study, the Knapsack Problem with Forfeits (KPF) was used as the object of study, adding complexity to the optimization task. From the comparative analysis of the three proposed variations of the GSA, namely original GSA, GSA-LS, and GSA-CS, it was shown through computational experiments that the distance was 3.5% from the best-known solution to the problem.

KEYWORDS. Knapsack Problem with Forfeits, Simulated Annealing, Metaheuristics.

1. Introdução

A determinação do mínimo global de funções multidimensionais é crucial em diversas áreas, como Física, Química, Estatística, Finanças, Biologia e Matemática Aplicada. A busca pelo mínimo global é vital para resolver problemas complexos que demandam otimização eficiente. Métodos clássicos como gradiente descendente, Newton, quase-Newton e métodos de pontos interiores são eficazes para funções convexas, garantindo a convergência ao mínimo global.

Entretanto, muitas funções são não convexas e possuem múltiplos mínimos locais, complicando a busca pelo mínimo global. Nestes casos, há a necessidade de mecanismos que permitam escapar de ótimos locais, explorando o espaço de busca de forma mais abrangente [Gendreau et al., 2010; Tsallis e Stariolo, 1996]. Uma estratégia não determinística eficaz para encontrar aproximações do ótimo global é o Recozimento Simulado Clássico (Classical Simulated Annealing - CSA), proposto por pesquisadores da IBM na década de 80 [Kirkpatrick et al., 1983]. Esta metaheurística se inspira no recozimento físico de materiais, que envolve aquecimento e resfriamento gradual para atingir um estado de energia mínima.

Com a evolução da pesquisa, variações do CSA foram desenvolvidas. O Recozimento Simulado Rápido (*Fast Simulated Annealing* - FSA) [Szu e Hartley, 1987], que utiliza a distribuição de *Cauchy-Lorentz* ao invés da Gaussiana, permite uma exploração mais eficiente e uma convergência mais rápida. Baseado no CSA e no FSA, surgiu o Recozimento Simulado Generalizado (*Generalized Simulated Annealing* - GSA), que segundo a estatística mecânica de Tsallis [Tsallis e Stariolo, 1996], mostrou-se superior em algumas instâncias [Xiang et al., 1997, 2017].

Desse modo, este trabalho se propõe a realizar um estudo comparativo entre três variações do GSA, usando para isso três passos aleatórios distintos na geração das soluções vizinhas dentro de cada uma das versões do algoritmo. A comparação propriamente dita consiste no estudo de diferentes hibridizações do GSA: A primeira delas mantém o algoritmo em sua versão original, a segunda utiliza busca local (*Local Search* - LS), chamada de GSA-LS, e por último o GSA-CS, que o combina com a técnica de *Clustering Search* - CS Oliveira e Lorena [2007].

Para avaliar essas diferentes abordagens, foi escolhido o Problema da Mochila com Penalidades (*Knapsack Problem with Forfeits* - KPF) [Cerulli et al., 2020], uma variante do Problema da Mochila 0-1 que considera conflitos leves (penalidades). O KPF tem aplicações em várias áreas, como gestão de máquinas, turnos de trabalho, administração de medicamentos e logística [Capobianco et al., 2022].

O restante deste trabalho comparativo é organizado da seguinte forma: a Seção 2 apresenta a descrição e formulação matemática do problema. Em seguida, na Seção 3 é detalhada a proposta do método, os experimentos e resultados são discutidos na Seção 4. Por último, as conclusões e trabalhos futuros são abordados na Seção 5.

2. Definição do Problema e Formulação Matemática

O problema escolhido para estudar as variações no GSA foi o Problema da Mochila com Penalidades (*Knapsack Problem with Forfeits* - KPF). Este problema é um caso particular do problema NP-difícil conhecido como Problema da Mochila [Karp, 2010].

Conforme proposto inicialmente por [Cerulli et al., 2020; Capobianco et al., 2022], seja n a cardinalidade do conjunto de itens, de modo que cada item $i \in I$, possui um benefício associado $p_i \in P \mid p_i > 0$ e um peso $w_i \in W \mid w_i > 0$, onde $i = 1, \dots, n$. De igual modo, foi definido $b > 0$ como a capacidade máxima permitida pela soma dos pesos dos itens selecionados para compor uma solução da mochila. Cada par de itens de penalidades possui um respectivo custo de penalidade $d_{ij} > 0$. Desse modo, o Modelo de Programação Inteira Mista (MIP) para o KPF, pode ser descrito como:

$$\begin{aligned}
 \max \quad & \sum_{i \in \mathcal{I}} p_i x_i - \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{I} | i < j} d_{ij} x_i x_j \\
 & \sum_{i \in \mathcal{I}} w_i x_i \leq b
 \end{aligned} \tag{1}$$

$$x_i \in \{0, 1\} \quad \forall i \in I \tag{2}$$

Este MIP utiliza um conjunto de variáveis binárias, representadas por x_i que indicam se um determinado item $i \in I$ pertencem ou não a uma solução. Sobre as restrições matemáticas do problema, temos que a Restrição (1) impõe que a máxima capacidade da mochila seja respeitada. Já a família de Restrições (2) define o domínio de cada variável de decisão. Por último, a função objetivo visa maximizar o lucro minimizando os custos de penalidades.

3. Metodologia

Nesta seção é apresentada a metodologia de estudo para as três variações propostas para o GSA (GSA, GSA-LS, GSA-CS) utilizadas no estudo comparativo, bem como a heurística construtiva e o mecanismo de perturbação que foi utilizado na geração de soluções vizinhas. Desse modo, essa seção fica organizada da seguinte forma: Na Subseção 3.1 é apresentado o algoritmo básico do GSA com a heurística construtiva responsável por gerar uma solução inicial para o KPF e o mecanismo de perturbação. Em seguida, nas Subseções 3.2 e 3.3 são apresentadas, respectivamente, as hibridizações do GSA com busca local e a técnica de *clustering search*.

3.1. Generalized Simulated Annealing - GSA

Com a finalidade de abordar problemas de otimização que possuem complexas funções de custo não convexas, uma poderosa técnica estocástica, conhecida como *Classical Simulated Annealing* (CSA) foi proposta por Kirkpatrick et al. [1983]. Visando oferecer mais versatilidade e eficiência para encontrar ótimos globais nessas complexas funções, [Tsallis e Stariolo, 1996] propuseram o *Generalized Simulated Annealing* (GSA), construído sobre as ideias do tradicional CSA.

O CSA simula um processo físico de aquecer um material e, em seguida, diminuir lentamente a temperatura para diminuir seus defeitos, encontrando assim um estado com energia mínima. Este método introduz temperaturas artificiais, que são reduzidas gradualmente, com fins de escapar de mínimos locais. Dentre as principais abordagens do CSA, podemos destacar:

- *Classical Simulated Annealing* (CSA): Utiliza uma distribuição gaussiana para as visitas de estado (processo que gera novas soluções a partir de uma corrente) e segue as estatísticas de Boltzmann-Gibbs;
- *Fast Simulated Annealing* (FSA): Emprega uma distribuição *Cauchy-Lorentz*, permitindo um resfriamento mais rápido e maior eficiência, uma vez que por possuir caudas mais pesadas do que a Gaussiana, permite passos maiores que ajuda escapar de ótimos locais mais eficazmente Szu e Hartley [1987].
- *Generalized Simulated Annealing* (GSA): Generaliza tanto a CSA como a FSA dentro de uma estrutura unificada inspirada na termostática generalizada [Tsallis, 1988]. Esta abordagem introduz dois parâmetros, q_v e q_A , que controlam as distribuição de visitas e aceitação, respectivamente, de modo mais eficaz.

Desse modo, o GSA proporciona uma metodologia robusta com fins de alcançar um ótimo global, valendo-se para isso de algumas extensões do FSA e CSA. Através do ajuste fino dos parâmetros q_v e q_A , o GSA pode se adaptar a vários cenários de otimização, garantindo uma convergência mais rápida e maior probabilidade de encontrar mínimos globais em comparação com os métodos tradicionais.

Vale ressaltar que quando $q_v = 2$ e $q_A = 1$, o GSA se comporta como CSA com uma distribuição gaussiana para visitas de estado e usa a estatísticas de Boltzmann-Gibbs para aceitação. Agora, quando $q_v = 1$, o GSA se comporta como o FSA com uma distribuição Cauchy-Lorentz para visitas de estado.

Com vista no exposto acima, o algoritmo do GSA pode ser resumido, como a seguir:

- **Inicialização:** Comece com um estado inicial arbitrário x_1 e uma temperatura inicial alta $T_{q_v}(1)$.
- **Iteração:** Para cada passo t :
 1. Obtenha x_{t+1} a partir de x_t , usando a distribuição de visitação $g_{q_v}(\Delta x_t)$, conforme definido na Equação 3.
 2. Calcule a nova energia $E(x_{t+1})$.
 3. Aceite x_{t+1} com base na probabilidade de aceitação $P_{q_A}(x_t \rightarrow x_{t+1})$, conforme definido na Equação 5
- **Resfriamento:** Reduza gradualmente as temperaturas $T_{q_v}(t)$ e $T_A(t)$.

A distribuição de visitação $g_{q_v}(\Delta x_t)$ para um determinado tamanho de passo $\Delta x_t = x_{t+1} - x_t$ é dado por:

$$g_{q_v}(\Delta x_t) \propto \left[1 + (q_v - 1) \frac{(\Delta x_t)^2}{T_{q_v}(t)} \right]^{\frac{1}{1-q_v}} \quad (3)$$

Onde, a temperatura $T_{q_v}(t)$:

$$T_{q_v}(t) = \frac{T_{q_v}(1)}{(1 + t)^{\frac{2}{q_v-1}}} \quad (4)$$

O processo generalizado da probabilidade de aceitação $P_{q_A}(x_t \rightarrow x_{t+1})$ adaptada do algoritmo de Metropolis é a seguinte:

$$P_{q_A}(x_t \rightarrow x_{t+1}) = \begin{cases} 1 & \text{if } E(x_{t+1}) < E(x_t) \\ \left[1 + (q_A - 1) \frac{E(x_{t+1}) - E(x_t)}{T_A(t)} \right]^{\frac{1}{q_A-1}} & \text{if } E(x_{t+1}) \geq E(x_t) \end{cases} \quad (5)$$

Aqui, $T_A(t)$ é a temperatura de aceitação no passo t .

Essa capacidade de adaptar o comportamento do algoritmo de acordo com as características do problema torna o GSA uma ferramenta poderosa para uma gama de aplicações de otimização. Ele combina a eficiência de convergência do FSA com a flexibilidade do CSA, oferecendo um equilíbrio entre exploração e intensificação que pode ser ajustado conforme necessário para resolver problemas complexos de otimização.

Neste trabalho, a implementação da versão básica do GSA com ligeiras modificações consiste em uma etapa de inicialização, onde uma solução inicial é gerada através de uma heurística construtiva, como descrito na Seção 3.1.1 e representada na Linha 3 do Algoritmo 1. Em seguida, a temperatura inicial (T_i) vai decrescendo de um fator ($0 < \alpha < 1$) até atingir uma temperatura final (T_f). Através da distribuição de *Cauchy* uma solução vizinha é gerada (*GetNeighbor()*) e passa por um critério de aceitação (*CalculateProbability()*), como definida na Equação 5.

Algorithm 1: GSA

```

1 Data:  $S, T_i, T_f, \alpha, max_{iter}, q_A, r_f, r_i, max_r, flight$ 
2 begin
3    $S \leftarrow CarouselForfeits(X, W, P, F, D, b)$ 
4    $S_{best} \leftarrow \emptyset$ 
5    $t \leftarrow T_i$ 
6   while  $t \geq T_f$  do
7     for  $iter \in max_{iter}$  do
8        $last_{impr} = last_{impr} + 1$ 
9        $S \leftarrow GetNeighbor(S_{curr}, t, T_i, iter, flight)$ 
10      if  $rand < CalculateProbability(S, S_{curr}, q_A, t)$  then
11         $S_{curr} \leftarrow S$ 
12        if  $cost(S_{curr}) > cost(S_{best})$  then
13           $S_{best} \leftarrow S_{curr}$ 
14           $last_{impr} = 0$ 
15        end
16      end
17    end
18     $t \leftarrow t * \alpha$ 
19    if  $(last_{impr} \geq r_i)$  and  $(events \leq max_r)$  then
20       $t \leftarrow t * r_f$ 
21       $events = events + 1$ 
22    end
23  end
24  return  $S_{best}$ 
25 end

```

Após um determinado número de iterações sem melhorias (r_i) desde que não se tenha ultrapassado o número máximo de eventos de rea aquecimentos (max_r) é realizado um aumento ($r_f > 1$) da temperatura corrente (t) para que se alcance uma maior diversificação e facilite o escape de ótimos locais. A contribuição mais relevante do algoritmo se encontra no método (*GetNeighbor*), que é melhor elucidado na Seção 3.1.2. De modo geral, as ideias discutidas podem ser acompanhadas sequencialmente no Algoritmo 1.

3.1.1. Heurística Construtiva

Na etapa de construção da solução viável que inicializa o GSA, foi utilizada uma heurística construtiva proposta inicialmente por [Cerulli et al., 2020], chamada *CarouselForfeits* e derivada do *Carousel Greedy* (CG) Cerrone et al. [2017], uma abordagem geral para melhorar algoritmos gulosos. Este algoritmo permite reconsiderar e substituir escolhas gulosas iniciais que mais tarde podem revelar-se desvantajosas. No nosso contexto, estas escolhas referem-se à inclusão de itens que poderiam incorrer em um custo alto de penalidade na solução final. Uma visão geral desta abordagem *Carousel Forfeits* é apresentada a seguir:

1. Uma solução inicial, denota por S , é construída usando o algoritmo *GreedyForfeits* Cerulli et al. [2020];

2. Alguns itens são removidos de S para obter S' , onde S' é um subconjunto de S . Especificamente, os últimos $\beta \times |S|$ itens adicionados são eliminados, onde β é um decimal entre 0 e 1.
3. Para $\max_iter_crf \times |S'|$ iterações ($\max_iter_crf \geq 1$, inteiro):
 - (a) O item $i \in S'$ correspondente à escolha mais antiga é removido;
 - (b) Gere um único item ganancioso i^* usando o algoritmo *GreedyForfeits*;
 - (c) $S' \leftarrow S' \cup \{i^*\}$
4. Uma nova solução é derivada resolvendo *GreedyForfeits*, começando com a inicialização do conjunto S' obtido na etapa anterior.

3.1.2. Mecanismo de Perturbação

No contexto de algoritmos de otimização como o *Simulated Annealing* e suas variações, determinar o “tamanho do passo”, ou seja, a magnitude da alteração feita na solução corrente para gerar uma nova solução candidata é fundamental. Neste trabalho, esse cálculo da quantidade de perturbações (adição, remoção e troca de itens) aplicadas à solução corrente é feito utilizando a distribuição de *Cauchy*, devido as suas características de “cauda pesada”, o algoritmo faz movimentos maiores no início (quando a temperatura é alta) e movimentos menores com o passar do tempo (quando a temperatura é baixa), o que contribui no equilíbrio entre exploração e intensificação. Em particular, neste trabalho, o tamanho do passo é calculado de três formas distintas, são elas:

- **Cauchy (GSA-c):** Gera um tamanho de passo baseado unicamente na temperatura corrente e na temperatura inicial ($t_{inicial}$). Ele usa uma escala (γ) que diminui linearmente de uma escala inicial (γ_i) para uma escala final (γ_f) com base na temperatura corrente ($t_{corrente}$), conforme a Equação 6.

$$\gamma = \gamma_i \times \left(\frac{t_{corrente}}{t_{inicial}}\right) + \gamma_f \times \left(1 - \left(\frac{t_{corrente}}{t_{inicial}}\right)\right) \quad (6)$$

O tamanho do passo é então determinado a partir de uma distribuição de *Cauchy* com a escala γ e tomando sua função de densidade de probabilidade (PDF), definida como:

$$f(x; x_0, \gamma) = \frac{1}{\pi \gamma \left[1 + \left(\frac{x - x_0}{\gamma}\right)^2\right]} \quad (7)$$

- **Cauchy-Adaptive (GSA-ca):** Ajusta a escala (γ) com base na temperatura corrente, temperatura inicial e na melhoria relativa (i) da solução atual ($curr$) em comparação com a melhor solução encontrada ($best$), de acordo com a Equação 10.

$$\gamma = \left(\gamma_i \times \left(\frac{t_{corrente}}{t_{inicial}}\right) + \gamma_f \times \left(1 - \left(\frac{t_{corrente}}{t_{inicial}}\right)\right)\right) \times (1 + i), \quad (8)$$

onde

$$i = \max\left(0, \frac{best - curr}{\max(|best|, 1 \times 10^{-10})}\right) \quad (9)$$

O tamanho do passo é então determinado a partir de uma distribuição de *Cauchy* com a escala γ , conforme a Equação 7.

- **Cauchy-Hybrid (GSA-ch):** Combina o ajuste baseado na temperatura, o fator de melhoria adaptativo e um fator baseado em iteração para determinar o tamanho do passo. Esta abordagem tenta, portanto, equilibrar a exploração e a intensificação, considerando o progresso da iteração, a temperatura e a qualidade da solução.

$$\gamma = \left(\gamma_i \times \left(\frac{t_{corrente}}{t_{inicial}} \right) + \gamma_f \times \left(1 - \left(\frac{t_{corrente}}{t_{inicial}} \right) \right) \right) \times (1 - iter) \times (i \times iter), \quad (10)$$

O tamanho do passo é então determinado a partir de uma distribuição de *Cauchy* com a escala γ , conforme a Equação 7.

Com base nas ideias acima, implementamos essas três variações do cálculo do tamanho do passo no mecanismo de perturbação do GSA, conforme indicado na Linha 9 do Algoritmo 1. Nessa mesma linha o parâmetro chamado *flight* é responsável por selecionar as três diferentes estratégias de perturbação.

3.2. GSA-LS

A variação GSA-LS foi implementada como uma chamada de busca local entre as linhas 12 e 13 do Algoritmo 1. Essa busca local explora iterativamente um conjunto de soluções vizinhas, buscando encontrar melhores soluções dada uma solução corrente. Se uma solução melhor é encontrada, a solução corrente é substituída e o processo de busca continua até que seja encontrado um ótimo local, para isso foi definida uma vizinhança de troca de itens.

Seja $\bar{S}$ definido com o conjuntos de itens tal que $i \notin S$, desse modo a busca local visa trocar itens entre $\bar{S}$ and S , verificando se tais trocas melhoram o custo. À medida que o tamanho da instância cresce, trocar itens se torna mais caro computacionalmente, para minimizar este efeito, foi introduzido uma forma eficiente de avaliar a solução, utilizando para isso a variável δ , definido nas Equações 11 e 12.

$$\delta_{ij} = \beta_j - \beta_i + d_{ij}, \quad \forall i \in S, j \in \bar{S} \quad (11)$$

$$\beta_m = p_m - \sum_{j \in S} d_{mj}, \quad \forall m \in I \quad (12)$$

onde β representa o ganho relativo de adicionar um determinado item à solução. No pior caso, a avaliação da solução possui uma complexidade de tempo de $\mathcal{O}(1)$, já para atualizar para atualizar os valores de β a complexidade de pior caso é $\mathcal{O}(n^2)$.

Implementar essa abordagem para avaliar o valor da solução na busca local representa uma significativa melhoria na eficiência computacional. Ela reduz a complexidade do pior caso de $\mathcal{O}(n^4)$ para $\mathcal{O}(n^2)$, o que significa que a execução do algoritmo se torna substancialmente mais rápida, especialmente para conjuntos de dados grandes Souto et al. [2024].

3.3. GSA-CS

A ideia de encontrar regiões promissoras do espaço de busca para aplicar a busca local motivou a hibridização do GSA com o algoritmo de *Clustering Search* (CS) [Oliveira e Lorena, 2007; Chaves et al., 2009; Oliveira et al., 2013; Souto et al., 2021]. O CS foi desenvolvido como uma abordagem para integrar diferentes técnicas de metaheurísticas, com o objetivo de identificar áreas promissoras no espaço de soluções para realizar uma busca local. Para isso, o CS agrupa o espaço de busca em *clusters* definidos por três atributos: $C = (c, v, r)$, onde c é a solução que representa o centro do *cluster* C , v é o número de soluções agrupadas em C , que determina quando

que o *cluster* se torna promissor, e por último, r , uma variável de controle que verifica se a aplicação da busca local está melhorando o centro c .

Neste trabalho, o GSA foi utilizado para gerar soluções para o CS, inicialmente, agrupando as soluções obtidas em diferentes *clusters* até que sejam definidos γ *clusters*. Em seguida, para cada solução gerada, determina-se em qual *cluster* ela deve ser adicionada, utilizando a distância de *Hamming* para medir a proximidade entre a nova solução e o centro c_i de cada *cluster*.

Após essa atribuição, se uma solução possuir um custo melhor do que o centro do *cluster*, ela passa a ser o novo centro e o volume do cluster é atualizado. Se a solução não melhorar o custo do centro, apenas o volume do *cluster* é atualizado. Quando o volume de um *cluster* atinge um número máximo de soluções (v_{max}), é aplicada uma busca local, igual a detalhada na Seção 3.2, no centro do *cluster*.

4. Experimentação Computacional

Esta seção está organizada em três subseções: A primeira subseção é dedicada a descrever os detalhes de implementação e os parâmetros de calibrados para cada algoritmo. A segunda reporta o conjunto instâncias e analisa os principais resultados dos experimentos computacionais relacionadas ao GSA, comparando-o com GSA-LS e GSA-CS, utilizando as três diferentes estratégias de geração de soluções vizinhas, totalizando nove comparações.

4.1. Detalhes de Implementação e Parametrização

Os métodos propostos foram implementados no Sistema Operacional Linux, usando a linguagem de programação Python, com o interpretador na versão 3.10.12. Os experimentos foram executados dez vezes para cada instância em um Intel(R) Core(TM) 11ª Geração i7-11800H executando em uma frequência de 4.60GHz com 16GB RAM.

Antes dos experimentos computacionais, no entanto, era necessário realizar uma parametrização para cada um dos nove métodos individualmente, são eles: GSA, GSA-LS, GSA-CS com cada um deles utilizando os três diferentes mecanismos de perturbação. Para esta parametrização, foi utilizada uma ferramenta de otimização conhecida como *iRace* [López-Ibáñez et al., 2016], seu princípio reside em explorar eficientemente o espaço de parâmetros de um algoritmo para encontrar o conjunto ideal de parâmetros que maximize seu desempenho em um determinado problema. Portanto, para determinar os principais parâmetros para o GSA, como a temperatura inicial (T_i), temperatura final (T_f), fator de decrescimento da temperatura (α), número máximo de iterações (max_{iter}), parâmetro de generalização do GSA (q), fator de reaquecimento (r_f), número de iterações sem melhorias (r_i), e o número máximo de eventos de reaquecimentos (max_r), volume máximo do *cluster* (v_{max}) e *distance threshold* do CS (λ).

Tabela 1: Valores testados e selecionados para cada parâmetro do GSA

Parâmetros	Intervalos	GSA-c	GSA-ca	GSA-ch	GSA-LS-c	GSA-LS-ca	GSA-LS-ch	GSA-CS-c	GSA-CS-ca	GSA-CS-ch
T_i	(100; 200; 300)	100	100	100	300	100	200	300	100	100
T_f	(0.2; 0.3; 0.4)	0.2	0.2	0.2	0.3	0.2	0.3	0.3	0.2	0.3
α	(0.70; 0.85; 0.90)	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
max_{iter}	(50; 100; 150)	50	50	50	50	50	50	50	50	50
q	(1.5; 2.0; 3.0)	3.0	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
r_f	(1.5; 2.0; 2.5)	2.5	2.5	2.5	2.5	2.0	2.5	2.5	2.0	2.5
r_i	(150; 200; 250)	150	250	250	250	200	150	250	200	150
max_r	(10; 15; 20)	15	10	15	10	10	10	10	10	10
v_{max}	(10; 20; 30)	-	-	-	-	-	-	10	30	30
λ	(0.002, 0.005, 0.008)	-	-	-	-	-	-	0.005	0.008	0.005

Os valores testados e selecionados para os parâmetros do GSA, GSA-LS e GSA-CS, levando em consideração os três mecanismos de perturbação para cada um deles podem ser conferidos na Tabela 1

4.2. Resultados Computacionais

Para avaliar o desempenho dos algoritmos propostos, foram realizados testes nas 40 instâncias originais (O) de *benchmark* propostas por [Cerulli et al., 2020], essas instâncias foram organizadas em 4 grupos de 10 instâncias para cada quantidade de itens $n = \{500, 700, 800, 1000\}$, com pares de penalidades formados aleatoriamente $l = 6n$, peso dos itens $w_i \in [3, \dots, 20]$, benefício dos itens $p_i \in [5, \dots, 25]$, custo dos pares de penalidades $d_k \in [2, \dots, 15]$, e a capacidade total da mochila $b = 3n$.

4.2.1. Comparação entre os mecanismos de perturbação

Os algoritmos foram executados com os parâmetros selecionados pelo *iRace* na Seção 4.1. Para as análises, uma importante métrica foi calculada, o *gap* da solução, que é definido como $1 - (Sol_{heurística} / Sol_{CPLEX})$, onde $Sol_{heurística}$ representa o custo da solução da heurística empregada, e Sol_{CPLEX} denota o custo associado à melhor solução encontrada pelo solver CPLEX e reportada por Capobianco et al. [2022], usando um tempo de limite de 3h. Após dez rodadas para cada uma das 40 instâncias, calculou-se a média dos *gap* das soluções com o melhor custo, bem como o tempo de execução para cada conjunto de instância.

Em uma primeira etapa, foi calculado o *gap* levando em consideração cada um dos mecanismos de perturbação para os métodos GSA, GSA-CS, GSA-LS. Este cálculo das médias pode ser conferido na Tabela 2, onde são comparados os três algoritmos com seus três mecanismos de perturbação (Seção 3.1.2), totalizando nove comparações.

Tabela 2: Comparação de *gap* e tempo médio entre os diferentes mecanismos de perturbação

Métrica	Método	Conjunto de Instâncias "O"			
		Numero de itens			
		500	700	800	1000
Gap Médio (%)	GSA-c	4,39	5,26	4,60	4,56
	GSA-ca	4,43	5,27	4,61	4,57
	GSA-ch	4,37	5,23	4,61	4,55
	GSA-LS-c	3,41	4,00	3,50	3,32
	GSA-LS-ca	3,36	4,02	3,52	3,37
	GSA-LS-ch	3,31	4,02	3,50	3,20
	GSA-CS-c	3,58	4,68	4,11	4,11
	GSA-CS-ca	3,61	4,67	4,13	4,12
	GSA-CS-ch	3,60	4,61	4,08	4,06

A análise comparativa dos mecanismos de perturbação revelou que os melhores resultados, destacados pelos menores *gaps* e tempo em negrito, foram obtidos pelos mecanismos GSA-ch, GSA-LS-ch e GSA-CS-ch. Esses resultados indicam que as abordagens mais eficazes são aquelas que implementam um mecanismo de perturbação híbrido (Cauchy-Hybrid), como pode ser conferido na Tabela 2.

4.2.2. Comparação entre os métodos

Uma vez que o mecanismo de perturbação híbrido apresentou os menores *gaps*, foi realizada uma comparação entre os diferentes métodos (GSA-ch, GSA-LS-ch e GSA-CS-ch). De acordo com a Figura 1, o método que comparativamente atingiu o menor *gap* foi o GSA-LS-ch.

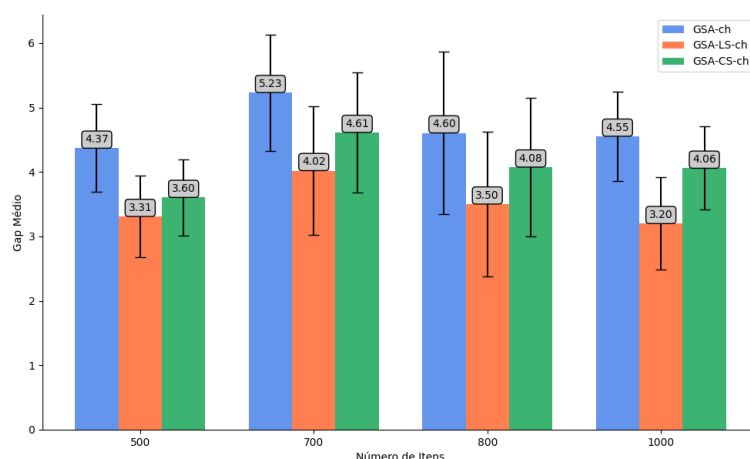


Figura 1: Comparação entre os métodos

Desse modo, o método de melhor desempenho, GSA-LS-ch, foi comparado com a melhor solução encontrada pelo CPLEX em um tempo limite de 3 horas, conforme reportado por Capobianco et al. [2022], para avaliar o comportamento do GSA em diferentes contextos de perturbação e hibridizações com outras heurísticas, como Busca Local e Clustering Search. Conforme mostrado na Figura 2, a melhor versão do GSA, que aplica busca local com um mecanismo de perturbação híbrido, apresentou resultados com uma alta probabilidade de encontrar *gaps* entre 2.0% e 4.0%, com baixa variância entre as soluções encontradas.

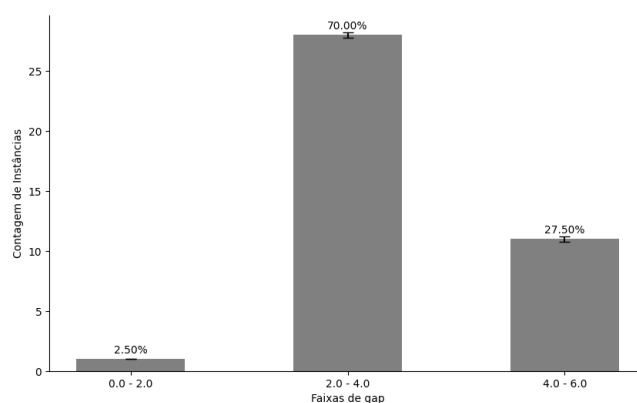


Figura 2: Comparação entre a solução usando o CPLEX e o GSA-LS-ch

A fim de avaliar a significância estatística das diferenças observadas nos valores das melhores soluções obtidas pelos métodos GSA, GSA-LS e GSA-CS, foi conduzido também um teste de hipótese não paramétrico *Wilcoxon Signed Rank Test* [Hettmansperger e McKean, 2010; Pratt, 1959]. Este teste é particularmente adequado quando não se dispõe de informações sobre a distribuição dos dados ou quando os dados não atendem aos pressupostos dos testes paramétricos.

O referido teste de hipótese permite rejeitar a hipótese nula em um nível de significân-

cia denotado por θ , com probabilidade de $(1 - \theta) \times 100\%$. Este estudo considerou um nível de significância de $\theta = 0,05$. As hipóteses formuladas são as seguintes:

- Hipótese Nula (H_0): A mediana das diferenças é zero.
- Hipótese Alternativa (H_1): A mediana das diferenças não é zero.

A hipótese nula (H_0) deste teste afirma que a mediana das diferenças entre os pares é zero, o que implica que não há diferença significativa entre os métodos comparados. A hipótese alternativa (H_1) sugere que a mediana das diferenças não é zero, indicando a existência de uma diferença significativa entre os métodos.

De acordo com a Tabela 2, como o GSA-LS-ch possui o menor valor de *gap* entre os métodos (GSA, GSA-LS e GSA-CS) o teste de significância estatística foi realizado em relação a ele, considerando as duas comparações: GSA-LS-ch e GSA-ch, GSA-LS-ch e GSA-CS-ch. Em termos práticos, se a hipótese nula for rejeitada, podemos concluir que há uma diferença estatisticamente significativa entre os métodos comparados.

Após a realização dos testes de hipótese, levando em consideração cada par de comparação, obtivemos o p-valor $< \theta$, assim a hipótese nula foi rejeitada e há mais diferenças positivas do que diferenças negativas, então a hipótese de que GSA-LS-ch supera tanto o GSA-CS-ch quanto o GSA-ch é verdadeira. Portanto, isso sugere que a heurística de busca local que combina com o mecanismo de perturbação híbrido efetivamente melhoram a qualidade das soluções encontradas.

5. Conclusão e Trabalhos Futuros

Este trabalho apresentou uma análise comparativa de diferentes variações do algoritmo *Generalized Simulated Annealing* (GSA): o GSA original, GSA com Busca Local (GSA-LS) e GSA com *Clustering Search* (GSA-CS). Foram abordados o algoritmo básico do GSA, a heurística construtiva para gerar soluções iniciais e os diversos mecanismos de perturbação utilizados para gerar soluções vizinhas. Cada variação do GSA foi implementada e parametrizada para aprimorar a eficiência e a qualidade das soluções.

Os experimentos computacionais demonstraram que a hibridização heurísticas melhoraram a qualidade das soluções encontradas pelo GSA original. Em particular, o GSA-LS apresentou o melhor desempenho, aproximando-se em 3,20% da melhor solução obtida pelo solver CPLEX dentro de um limite de tempo de 3 horas.

Para trabalhos futuros, propomos investigar a integração de novas heurísticas de otimização dentro do GSA, desenvolver outras técnicas de parametrização automatizada para ajuste dos parâmetros, aplicar o GSA-LS em outros problemas de otimização, e conduzir uma análise mais aprofundada para entender melhor as propriedades convergentes dos algoritmos híbridos implementados. Essas direções de pesquisa podem contribuir para o avanço das técnicas de otimização.

Agradecimentos

Os autores agradecem à UFRJ (23079.227622/2023-70), CNPq (307663/2021-3, 304680/2023-0), FAPERJ (307663/2021-3) e CAPES pelo financiamento parcial desta pesquisa.

Referências

- Capobianco, G., D'Ambrosio, C., Pavone, L., Raiconi, A., Vitale, G., e Sebastiano, F. (2022). A hybrid metaheuristic for the knapsack problem with forfeits. *Soft Computing*, p. 1–14.
- Cerrone, C., Cerulli, R., e Golden, B. (2017). Carousel greedy: A generalized greedy algorithm with applications in optimization. *Computers & Operations Research*, 85:97–112.

- Cerulli, R., D'Ambrosio, C., Raiconi, A., e Vitale, G. (2020). The knapsack problem with forfeits. In *Combinatorial Optimization: 6th International Symposium, ISCO 2020, Montreal, QC, Canada, May 4–6, 2020, Revised Selected Papers 6*, p. 263–272. Springer.
- Chaves, A. A., Lorena, L. A. N., e Miralles, C. (2009). Hybrid metaheuristic for the assembly line worker assignment and balancing problem. In *Hybrid Metaheuristics: 6th International Workshop, HM 2009, Udine, Italy, October 16-17, 2009. Proceedings 6*, p. 1–14. Springer.
- Gendreau, M., Potvin, J.-Y., et al. (2010). *Handbook of metaheuristics*, volume 2. Springer.
- Hettmansperger, T. P. e McKean, J. W. (2010). *Robust nonparametric statistical methods*. CRC Press.
- Karp, R. M. (2010). *Reducibility among combinatorial problems*. Springer.
- 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.
- Oliveira, A. C. M. d., Chaves, A. A., e Lorena, L. A. N. (2013). Clustering search. *Pesquisa operacional*, 33:105–121.
- Oliveira, A. C. e Lorena, L. A. (2007). Hybrid evolutionary algorithms and clustering search. In *HYBRID evolutionary algorithms*, p. 77–99. Springer.
- Pratt, J. W. (1959). Remarks on zeros and ties in the wilcoxon signed rank procedures. *Journal of the American Statistical Association*, 54(287):655–667.
- Souto, G., Mendonca, I., Aritsugi, M., Simonetti, L., e González, P. H. (2024). A q-learning hybrid brkga applied to the knapsack problem with forfeits. In *2024 IEEE Congress on Evolutionary Computation (CEC)*. IEEE.
- Souto, G., Morais, I., Faulhaber, L., Ribeiro, G. M., e González, P. H. (2021). A hybrid brkga approach for the two stage capacitated facility location problem. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, p. 2007–2014. IEEE.
- Szu, H. e Hartley, R. (1987). Fast simulated annealing. *Physics letters A*, 122(3-4):157–162.
- Tsallis, C. (1988). Possible generalization of boltzmann-gibbs statistics. *Journal of statistical physics*, 52:479–487.
- Tsallis, C. e Stariolo, D. A. (1996). Generalized simulated annealing. *Physica A: Statistical Mechanics and its Applications*, 233(1-2):395–406.
- Xiang, Y., Gubian, S., e Martin, F. (2017). Generalized simulated annealing. *Computational Optimization in Engineering-Paradigms and Applications*, p. 25–46.
- Xiang, Y., Sun, D., Fan, W., e Gong, X. (1997). Generalized simulated annealing algorithm and its application to the thomson model. *Physics Letters A*, 233(3):216–220.