INSTITUTO FEDERAL DO ESPÍRITO SANTO
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA – PPCOMP

MAURICIO JASTROW UHLIG

COMPARAÇÃO DE ALGORITMOS DE OTIMIZAÇÃO NOS PROBLEMAS TSP E MINIMIZAÇÃO DA FUNÇÃO *RASTRIGIN*

Serra
2025

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A otimização é uma área fundamental da computação aplicada, voltada para a busca de soluções eficientes em problemas complexos. Entre os desafios mais conhecidos estão o Problema do Caixeiro-Viajante (TSP) e a minimização da Função de *Rastrigin*, ambos amplamente utilizados como benchmarks para avaliar o desempenho de algoritmos de otimização. O TSP consiste em encontrar a rota mais curta que visita um conjunto de cidades exatamente uma vez, enquanto a Função de *Rastrigin* é uma função com múltiplos mínimos locais, representando um desafio para algoritmos que buscam o mínimo global.

Neste contexto, os algoritmos de busca local, como *Hill-Climbin*g (HC), *Hill-Climbing* com Reinicialização (HC-R), *Simulated Annealing* (SA) e Algoritmos Genéticos (GA), destacam-se por sua capacidade de encontrar soluções satisfatórias em espaços de busca complexos. Cada um desses métodos possui características distintas: o HC é simples e rápido, mas tende a ficar preso em ótimos locais; o HC-R melhora essa limitação ao reiniciar a busca a partir de diferentes pontos; o SA permite movimentos para soluções piores com uma probabilidade controlada, facilitando a exploração do espaço de busca; e o GA utiliza operações de seleção, crossover e mutação para explorar e refinar soluções.

Este trabalho tem como objetivo comparar o desempenho desses algoritmos aplicados ao TSP e à minimização da Função de *Rastrigin*. A análise será conduzida com base em métricas como qualidade das soluções, variabilidade dos resultados e custo computacional. Eses problemas permitem avaliar a eficácia dos algoritmos em diferentes cenários, desde problemas combinatórios, como o TSP, até problemas contínuos, como a Função de *Rastrigin*.

A metodologia adotada inclui a implementação manual dos algoritmos em *Python*, a execução de múltiplas iterações para cada problema e a coleta de dados sobre custos, tempos de execução e acessos à função objetivo. Os resultados serão apresentados por meio de gráficos e tabelas, permitindo uma comparação detalhada do desempenho dos algoritmos. A partir dessa análise, espera-se identificar as vantagens e limitações de cada método, fornecendo *insights* valiosos para a escolha de algoritmos de otimização em problemas reais.

2 FUNDAMENTAÇÃO TEÓRICA

A busca local é uma abordagem eficaz para resolver problemas onde o que importa é o estado final da solução, e não o caminho percorrido até alcançá-lo. Diferente dos algoritmos tradicionais que exploram sistematicamente o espaço de estados mantendo múltiplos caminhos na memória, os algoritmos de busca local focam em um único estado atual, realizando modificações incrementais a partir dele. Essa característica torna a busca local especialmente útil para problemas de otimização em grandes ou infinitos espaços de estados, onde métodos sistemáticos seriam inviáveis. Além disso, a busca local utiliza pouca memória e pode encontrar soluções satisfatórias de forma eficiente. Entre os algoritmos mais conhecidos dessa abordagem estão o *hill climbing*, o *simulated annealing* e os algoritmos genéticos, que serão detalhados a seguir.(RUSSELL; NORVIG, 2010)

2.1 *Hill-Climbing* (HC)

O *Hill-Climbing* é um algoritmo de busca local que começa com um estado selecionado aleatoriamente e move-se iterativamente para o melhor estado vizinho. É um algoritmo guloso, sempre escolhendo o movimento que oferece a melhoria mais imediata. No entanto, o HC tende a ficar preso em máximos (ou mínimos) locais, platôs ou cristas, onde não há estados vizinhos melhores disponíveis. Apesar de sua simplicidade, o HC pode encontrar rapidamente soluções razoáveis em muitos problemas, mas não há garantia de que encontrará o ótimo global.(RUSSELL; NORVIG, 2010)

2.2 *Hill-Climbing* com Reinicialização (HC-R)

O *Hill-Climbing* com Reinicialização aborda as limitações do HC clássico ao executar o algoritmo várias vezes a partir de diferentes estados iniciais gerados aleatoriamente. Essa abordagem aumenta a probabilidade de encontrar o ótimo global, pois o algoritmo não fica confinado a uma única região do espaço de estados. A ideia é que, ao reiniciar a busca a partir de vários pontos, o algoritmo pode escapar de ótimos locais e explorar diferentes áreas do espaço de estados. Esse método é trivialmente completo, o que significa que ele eventualmente encontrará uma solução se ela existir, dado um número suficiente de reinícios. É particularmente eficaz em problemas com menos máximos locais e platôs.(RUSSELL; NORVIG, 2010)

2.3 *Simulated Annealing* (SA)

O *Simulated Annealing* (SA) é um algoritmo de busca local probabilístico que combina a *Hill-Climb* com *Random Walk*. Ele permite movimentos ocasionais para estados piores, o que ajuda o algoritmo a escapar de ótimos locais. A probabilidade de aceitar um movimento pior diminui ao longo do tempo, controlada por um parâmetro de *temperatura*

que é gradualmente reduzido. Inicialmente, em temperaturas altas, o algoritmo tem maior probabilidade de aceitar movimentos piores, permitindo uma exploração ampla do espaço de estados. À medida que a temperatura diminui, o algoritmo torna-se mais seletivo, focando em refinar a solução. O SA pode encontrar ótimos globais com alta probabilidade se o esquema de resfriamento for ajustado adequadamente.(RUSSELL; NORVIG, 2010)

2.4 Algoritmos Genéticos (GA)

Os **Algoritmos Genéticos** são inspirados na seleção natural e na evolução. Eles mantêm uma população de soluções candidatas (indivíduos) e as melhoram iterativamente por meio de seleção, *crossover* e mutação. Cada indivíduo é avaliado usando uma função de aptidão (*fitness*), e os indivíduos mais aptos têm maior probabilidade de serem selecionados para reprodução. Durante o *crossover*, partes de duas soluções parentais são combinadas para criar descendentes, que podem então sofrer mutações aleatórias. Esse processo introduz diversidade e permite que o algoritmo explore diferentes regiões do espaço de estados. Os Algoritmos Genéticos são particularmente eficazes quando a solução pode ser decomposta em componentes significativos (esquemas) que podem ser combinados de maneiras úteis. (RUSSELL; NORVIG, 2010)

3 Metodologia

Este trabalho tem como objetivo comparar o desempenho de algoritmos de otimização aplicados a dois problemas: o Problema do Caixeiro-Viajante (*Travelling Salesman Problem - TSP*) e a Minimização da Função de *Rastrigin*. A seguir, são detalhados os aspectos metodológicos adotados para o desenvolvimento, execução e análise dos resultados.

3.1 Desenvolvimento de Algoritmos e Problemas

Os algoritmos de otimização utilizados neste estudo incluem *Hill-Climbing* Clássico (*HC-C*), *Hill-Climbing* com Reinicialização (*HC-R*), *Simulated Annealing* (*SA*) e Algoritmo Genético (*GA*). Todos foram desenvolvidos manualmente em *Python 3.13*, seguindo as diretrizes que proíbem o uso de bibliotecas prontas para otimização, mas permitindo o uso de ferramentas auxiliares como *NumPy* e *Matplotlib*. A implementação pode ser encontrada no repositório do *GitHub*: <https://github.com/MauricioUhlig/mestrado_trabalho_IA_2>

3.1.1 Problema do Caixeiro-Viajante (*TSP*)

O Problema do Caixeiro-Viajante (TSP) foi representado como uma sequência de cidades, onde cada cidade é visitada exatamente uma vez antes de retornar à cidade inicial. Essa sequência foi modelada como uma permutação de números inteiros, representando a ordem de visitação das cidades. A solução ideal minimiza o custo total do trajeto, que é calculado como a soma das distâncias entre cidades consecutivas na sequência, incluindo o retorno à cidade inicial.

Os dados utilizados neste trabalho foram obtidos do conjunto de instâncias do TSP para o *Western Sahara*, composto por 29 cidades. O trajeto ideal para esta instância possui um custo total de 27.603. O conjunto de dados pode ser acessado no seguinte endereço: <http://www.math.uwaterloo.ca/tsp/world/wi29.tsp>.

Para gerar vizinhos, foi adotada a estratégia de troca de posições entre duas cidades selecionadas aleatoriamente no trajeto. Por exemplo, dado o estado inicial $[1, 2, 3, 4, 5]$, a troca das posições das cidades 2 e 4 pode resultar no estado $[1, 4, 3, 2, 5]$. Essa abordagem garante que todas as cidades permaneçam na solução e evita repetições ou exclusões.

No Algoritmo Genético (GA), a operação de crossover foi realizada utilizando o método Order 1 (OX), projetado especificamente para preservar a integridade das soluções no contexto de permutações. O método OX garante que os filhos herdados dos pais mantenham a sequência relativa de cidades, sem duplicações ou ausências. Por exemplo, para os estados pais $P1 = [1, 3, 5, 2, 6, 4]$ e $P2 = [6, 4, 1, 3, 2, 5]$, o crossover no ponto 2 resultaria nos filhos $C1 = [1, 3, 5, 4, 6, 2]$ e $C2 = [6, 4, 1, 3, 5, 2]$. Este método foi escolhido por sua eficiência em evitar erros comuns de representação em problemas de permutação.

3.1.2 Minimização da Função de Rastrigin

A Função de Rastrigin é uma função matemática frequentemente utilizada como benchmark em problemas de otimização. Sua definição é dada por:

$$f(x, y) = 20 + x^2 - 10\cos(2\pi x) + y^2 - 10\cos(2\pi y)$$

onde $x, y \in [-5.12, 5.12]$. A função apresenta múltiplos mínimos locais, tornando-se desafiadora para algoritmos de otimização que podem facilmente se prender a essas regiões.

Para a geração de vizinhos no espaço de busca, foi adotada uma abordagem baseada em perturbações aleatórias. As coordenadas x e y do ponto atual foram ajustadas ao somar valores retirados de uma distribuição normal com média 0 e desvio padrão 0.2, garantindo que as perturbações fossem pequenas e controladas. Essa estratégia permite explorar o espaço de busca de forma gradual, aumentando as chances de encontrar regiões promissoras para a otimização.

No Algoritmo Genético, o operador de crossover foi implementado por meio de uma média ponderada entre os dois pontos pais (p_1 e p_2). Para cada coordenada x e y, os descendentes foram calculados da seguinte forma: $c_1 = \alpha p_1 + (1 - \alpha)p_2, \quad c_2 = \alpha p_2 + (1 - \alpha)p_1$, onde α é um valor aleatório retirado de uma distribuição uniforme no intervalo $[0, 1]$. Esse método assegura que os descendentes sejam interpolados entre os pais, explorando de maneira eficiente o espaço de busca e evitando que os pontos gerados ultrapassem os limites definidos para a função.

A mutação foi implementada de forma semelhante à geração de vizinhos. Cada indivíduo da população teve suas coordenadas ajustadas por perturbações aleatórias, novamente utilizando uma distribuição normal com média 0 e desvio padrão 0.2. Essa abordagem introduz diversidade na população e aumenta a probabilidade de escapar de mínimos locais.

3.2 Parametrização dos Problemas

Para garantir uma execução consistente e comparável dos algoritmos de otimização, foram definidos parâmetros específicos para cada problema. Estes parâmetros foram ajustados de forma a equilibrar a exploração e a exploração do espaço de busca, considerando a complexidade de cada função objetivo.

3.2.1 Problema do Caixeiro-Viajante (TSP)

Para o TSP, os parâmetros dos algoritmos foram ajustados com base no número de cidades presentes na instância do problema. As configurações utilizadas para cada algoritmo são detalhadas a seguir:

- **Hill Climbing (HC):** O algoritmo foi executado em sua forma padrão, sem parametrizações adicionais, mantendo sua configuração original.
- **Hill Climbing com Reinicialização (HC-R):** O algoritmo foi configurado para executar 50 reinicializações do HC, permitindo uma exploração mais ampla do espaço de busca.
- **Simulated Annealing (SA):** O SA foi parametrizado com 7000 iterações e 70 repetições. A probabilidade de aceitar soluções piores foi inicializada em 1 e reduzida linearmente ao longo de 90% das iterações, chegando a 0 nas iterações finais.
- **Algoritmo Genético (GA):** O GA foi configurado com uma população de 110 indivíduos, 800 gerações e uma probabilidade de mutação de 10%.

3.2.2 Função de Rastrigin

Para a minimização da Função de Rastrigin, os parâmetros dos algoritmos foram definidos de forma a explorar adequadamente o espaço de busca no intervalo $[-5.12, 5.12]$. As configurações utilizadas para cada algoritmo são as seguintes:

- **Hill Climbing (HC):** Assim como no TSP, o HC foi executado em sua forma padrão, sem parametrizações adicionais.
- **Hill Climbing com Reinicialização (HC-R):** O HC-R foi configurado para realizar 450 reinicializações do HC, aumentando a probabilidade de escapar de mínimos locais e explorar diferentes regiões do espaço de busca.
- **Simulated Annealing (SA):** O SA foi parametrizado com 1100 iterações e 1 repetição. A probabilidade de aceitar soluções piores foi inicializada em 1 e reduzida linearmente ao longo de 90% das iterações, chegando a 0 no final.
- **Algoritmo Genético (GA):** O GA foi configurado com uma população inicial de 20 indivíduos, 50 gerações e uma probabilidade de mutação de 10%.

Esses parâmetros foram definidos de forma a permitir que os algoritmos alcancem um desempenho robusto em ambas as instâncias do problema, maximizando a qualidade das soluções encontradas enquanto minimizam o custo computacional.

3.3 Ambiente de Execução

Os experimentos foram conduzidos em um MacBook Pro com processador M3 e 36 GB de memória RAM. Este ambiente foi escolhido para garantir recursos computacionais suficientes para a execução eficiente dos algoritmos, permitindo a realização de testes intensivos e confiáveis.

3.4 Execução e Coleta de Dados

Cada algoritmo foi executado 30 vezes para cada problema, com o objetivo de avaliar a variabilidade dos resultados e reduzir o impacto de fatores aleatórios. Para o TSP, a métrica avaliada foi o custo total do trajeto, enquanto para a Função de Rastrigin foi utilizado o valor da função objetivo.

Além disso, o custo computacional foi analisado considerando o tempo médio de execução e a quantidade de acessos à função objetivo. Esses indicadores foram registrados em todas as execuções para possibilitar uma comparação abrangente entre os algoritmos.

3.5 Análise dos Resultados

Os resultados foram sintetizados e apresentados por meio de gráficos e tabelas. Gráficos boxplot foram gerados para comparar o desempenho dos algoritmos em termos das funções objetivo, destacando valores máximos, mínimos, medianas e percentis 25 e 75.

3.5.1 Evolução da *Fitness*

Adicionalmente, gráficos da evolução da *fitness* ao longo das iterações foram criados, permitindo uma análise detalhada do comportamento dos algoritmos durante o processo de otimização.

3.5.2 Custo Computacional

A análise do custo computacional incluiu a comparação do tempo médio de execução e da quantidade de acessos à função objetivo. Estes indicadores foram utilizados para avaliar a eficiência relativa dos algoritmos, considerando tanto a qualidade das soluções quanto o esforço computacional requerido.

4 Resultados e Discussões

Neste capítulo, são apresentados e discutidos os resultados obtidos durante a execução dos experimentos com os algoritmos de otimização. A análise é conduzida com base nos critérios de desempenho definidos, como qualidade das soluções, variabilidade dos resultados e custo computacional.

4.1 Problema do Caixeiro-Viajante (TSP)

Os resultados obtidos para o Problema do Caixeiro-Viajante (TSP) incluem uma análise detalhada do custo total do trajeto otimizado por cada algoritmo. A Tabela 1 resume os valores máximo, mínimo, mediano e o desvio padrão dos custos obtidos ao final de 30 execuções independentes. Esses valores permitem uma comparação quantitativa do desempenho dos algoritmos, destacando a consistência e a variabilidade dos resultados.

Além disso, a Figura 1 apresenta um boxplot que ilustra a distribuição dos custos para cada algoritmo. No eixo X, estão representados os algoritmos testados, enquanto no eixo Y, é exibido o custo total do trajeto. A linha vermelha em cada boxplot indica o custo mediano, fornecendo uma visão clara da tendência central dos resultados. A análise do boxplot permite identificar a dispersão dos custos, bem como a presença de possíveis outliers, o que é fundamental para avaliar a robustez de cada algoritmo.

Tabela 1 – Estatísticas dos custos para o TSP.

Algoritmo	**Máximo**	**Mínimo**	**Mediana**	**Desvio Padrão**
HC	47155	31161	37551	4642
HC-R	32027	28230	30139	1016
SA	29544	27601	28652	601
GA	30542	27601	28113	791

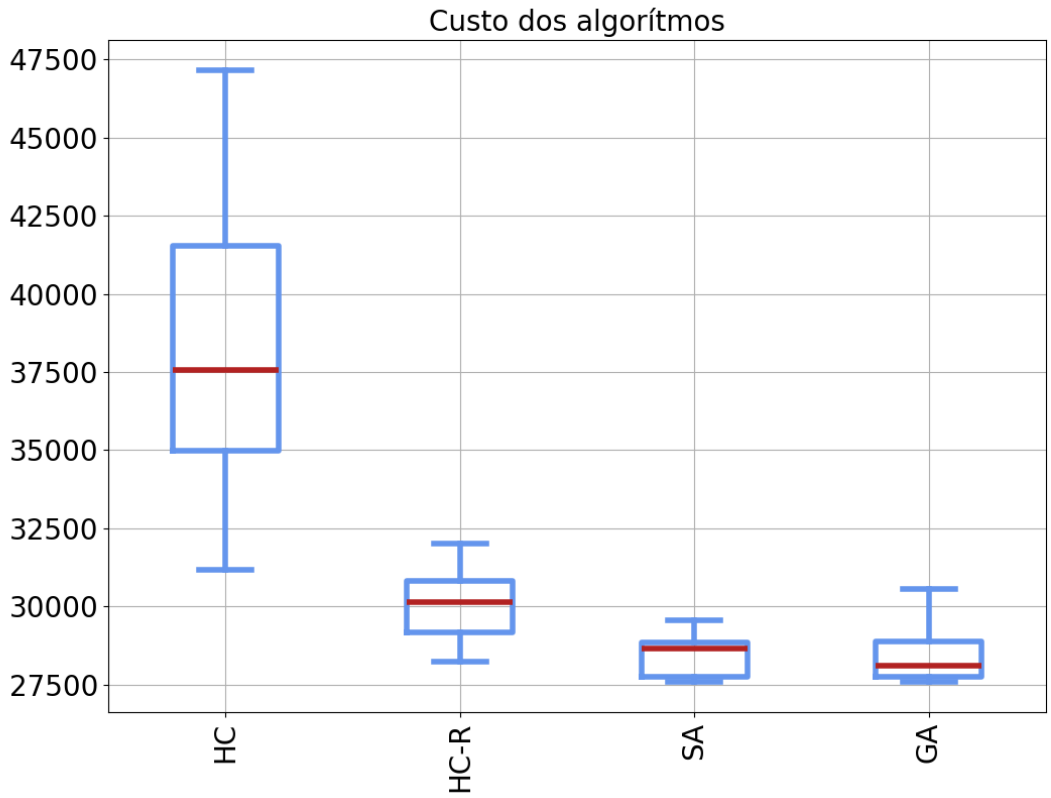

Figura 1 – Boxplot dos custos para o TSP.

Podemos observar, com base na Tabela 1 e na Figura 1, que os algoritmos **Simulated Annealing (SA)** e **Algoritmo Genético (GA)** foram capazes de encontrar o valor mínimo de percurso para o problema do Caixeiro Viajante (TSP). No entanto, o GA apresentou um desempenho superior ao SA ao analisar a mediana dos resultados, indicando que 50% das execuções do GA obtiveram custos inferiores em comparação com 50% das execuções do SA.

Além disso, ao comparar a evolução do *fitness* do SA (Figura 9) com a do GA (Figura 11), nota-se que o GA convergiu para a solução ótima em um número menor de iterações. Essa convergência mais rápida permitiu a utilização de uma parametrização com menos iterações, reduzindo o número de acessos à função objetivo e, consequentemente, tornando o GA mais eficiente em termos computacionais. Para mais detalhes, consulte o Apêndice A.

O algoritmo **Hill Climbing (HC)** obteve o pior desempenho na comparação de custos, um resultado esperado devido à sua natureza gulosa e à alta suscetibilidade aos efeitos da inicialização. Como o HC converge para o primeiro mínimo local encontrado, sua performance depende fortemente da sorte em iniciar em uma região próxima a uma solução ideal. Essa limitação intrínseca do HC faz com que ele frequentemente fique preso em soluções subótimas.

Por outro lado, o **Hill Climbing com Reinicialização (HC-R)** demonstrou um desempenho significativamente melhor, chegando próximo aos resultados obtidos pelos algoritmos SA e GA. Essa melhoria pode ser atribuída à estratégia de reinicialização, que permite ao HC-R explorar diferentes regiões do espaço de busca, aumentando as chances de encontrar soluções melhores. No entanto, o desempenho do HC-R ainda depende, em parte, da sorte associada à inicialização do HC e do número limitado de reinicializações configuradas.

4.1.1 Melhor solução e Evolução da *fitness*

Nesta seção, são apresentadas as melhores soluções de rotas encontradas por cada algoritmo, com as rotas representadas por linhas azuis e as cidades por pontos verdes.

Além disso, as evoluções de *fitness* de cada algoritmo são exibidas. Nessas representações, a linha verde indica o melhor valor de *fitness* total alcançado até o momento, enquanto a linha azul mostra o melhor *fitness* da iteração atual. Ambas as métricas são plotadas em função do número de iterações, onde o eixo Y corresponde ao valor de *fitness* e o eixo X representa o número de iterações.

- **Hill Climb (HC)**

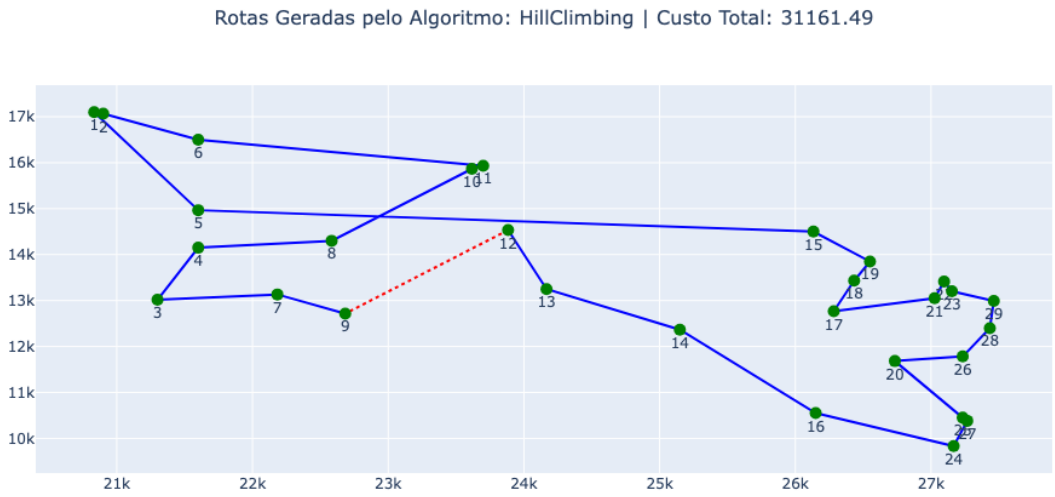

Figura 2 – Melhor solução encontrada pelo algoritmo HC para o problema TSP.

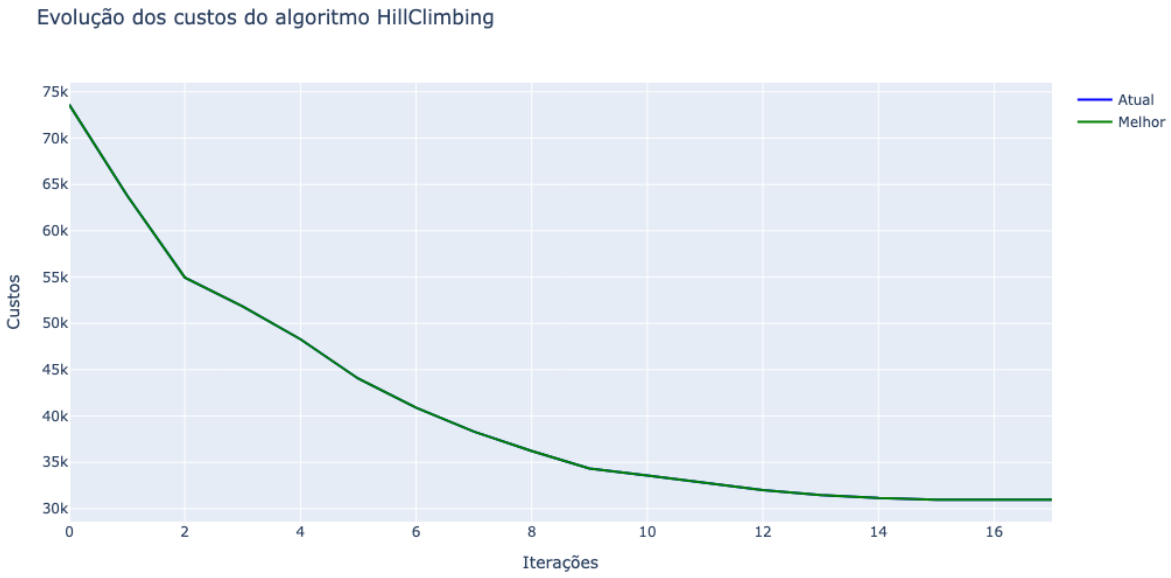

Figura 3 – Evolução da *fitness* da melhor solução do algoritmo HC para o problema TSP.

Não é possível observar a linha azul na Figura 3, pois pela característica do HC, ele sempre procura a melhor solução a cada iteração. Isto faz com que a *Atual* seja igual à *Melhor*. Esta característica também o faz convergir apenas para o **mínimo local** mais próximo ao seu ponto de partida.

- **Hill Climb com Reinicialização (HC-R)**

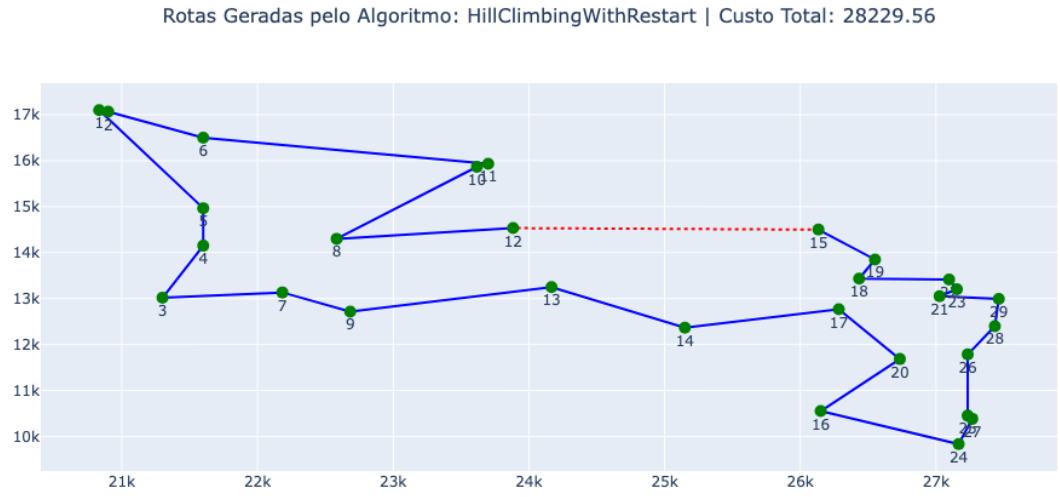

Figura 4 – Melhor solução encontrada pelo algoritmo HC-R para o problema TSP.

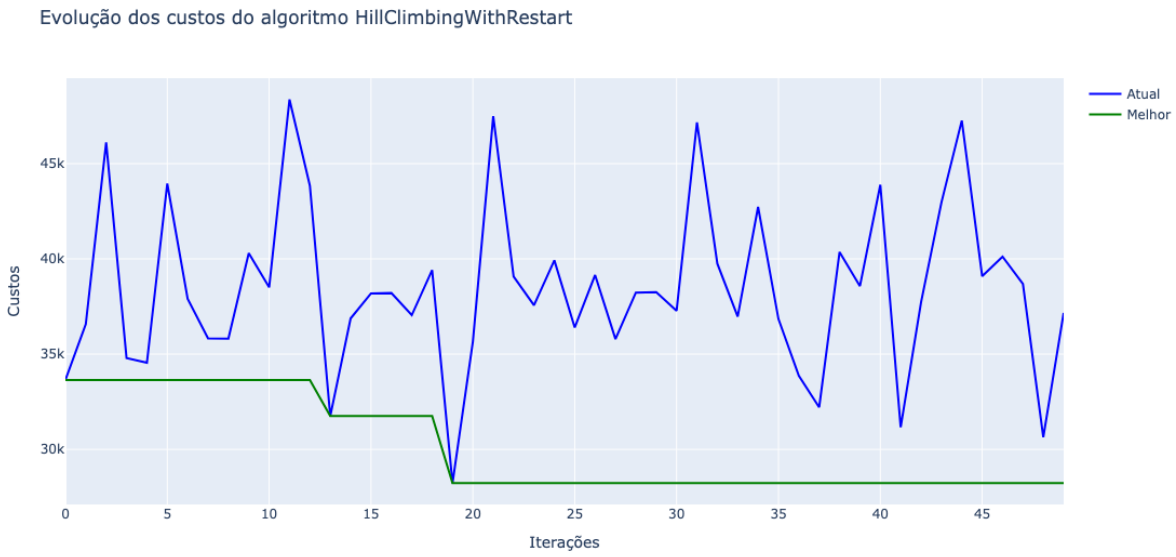

Figura 5 – Evolução da *fitness* da melhor solução do algoritmo HC-R para o problema TSP.

Pode-se observar na Figura 5 o comportamento de *restart* do algorítmo HC-R, pois a linha azul se comporta de forma aleatória, sempre encontrando o mínimo local de uma execução do HC a partir de um ponto aleatório. Mas, a melhor solução (em verde) sempre é atualizada assim que uma iteracao é capaz de encontrar um custo menor.

- **Simulated Annealing (SA)**

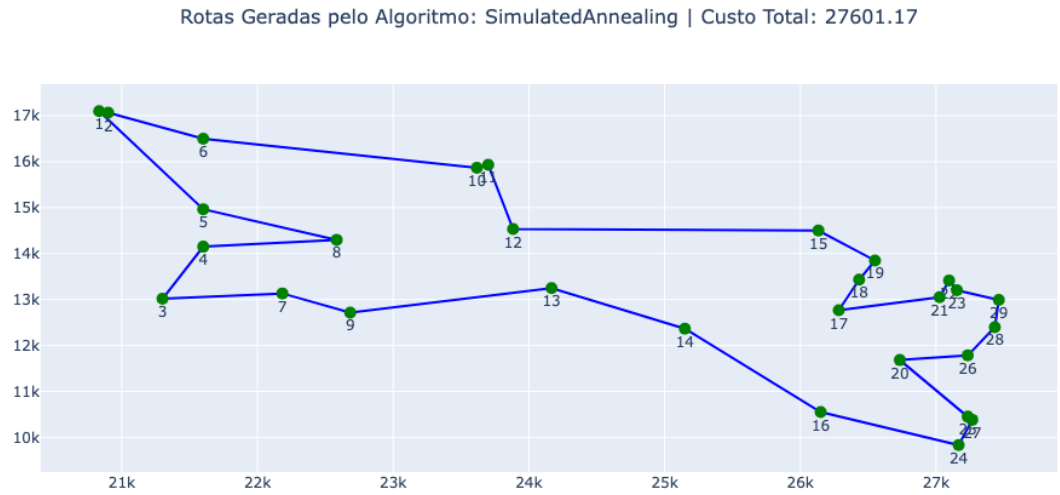

Figura 6 – Melhor solução encontrada pelo algoritmo SA para o problema TSP.

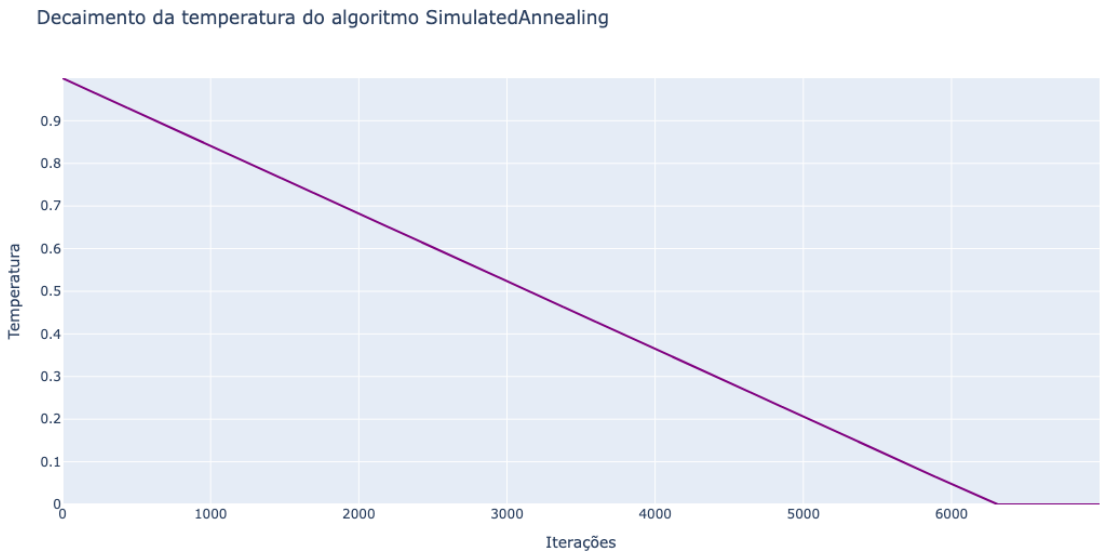

Figura 7 – Evolução da temperatura da melhor solução do algoritmo SA para o problema TSP.

Na Figura 7, observa-se o decaimento linear da temperatura ao longo das iterações, iniciando em 1 e atingindo 0 pouco após 6000 iterações.

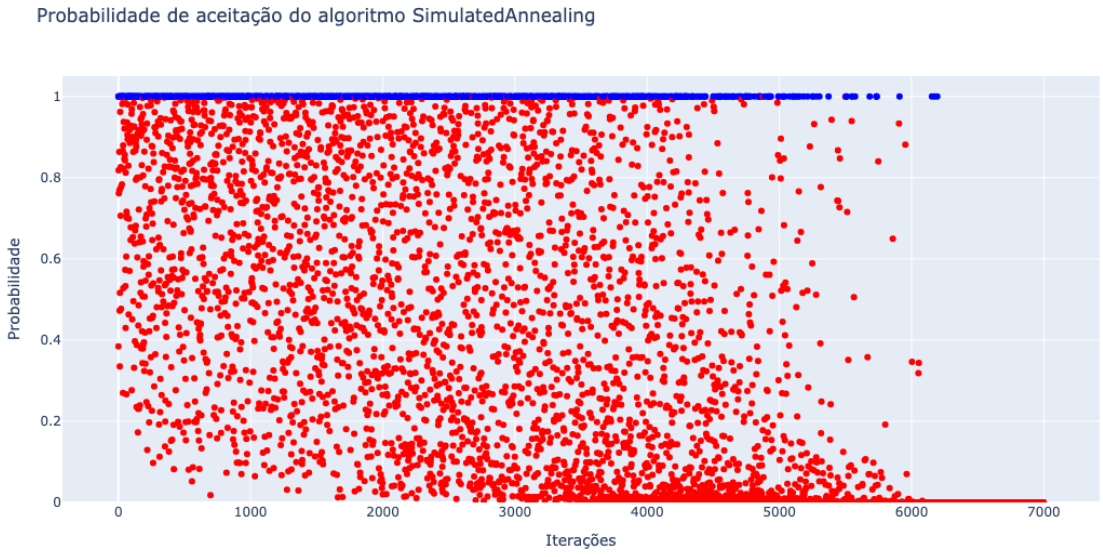

Figura 8 – Probabilidade de aceitação da melhor solução do algoritmo SA para o problema TSP.

A Figura 8 exibe a probabilidade de aceitação (eixo Y) de soluções com custo inferior ao melhor (em vermelho), que decresce ao longo das iterações (eixo X). Em azul, são destacados os casos em que a probabilidade é 1 (100

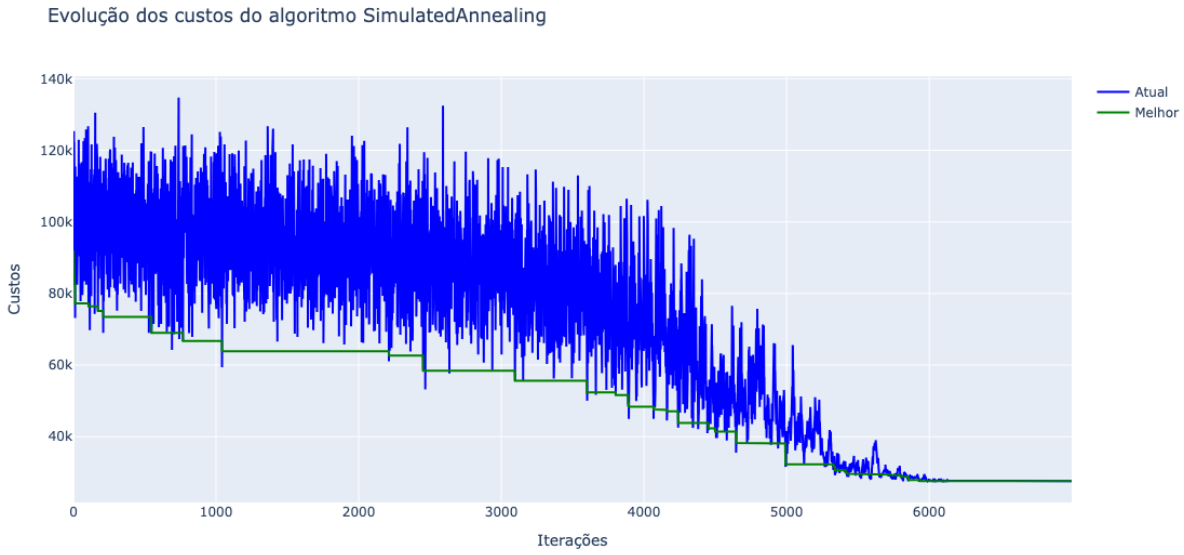

Figura 9 – Evolução da *fitness* da melhor solução do algoritmo SA para o problema TSP.

As Figuras 7, 8 e 9 ilustram o comportamento do algoritmo Simulated Annealing (SA). No início, a temperatura está elevada, o que aumenta a probabilidade de aceitar soluções com custo maior, permitindo que o algoritmo escape de mínimos locais. À medida que a temperatura decai, a probabilidade de aceitar custos superiores diminui gradualmente, enquanto o valor do custo (*fitness*) converge para uma solução otimizada.

- **Algoritmo Genético (GA)**

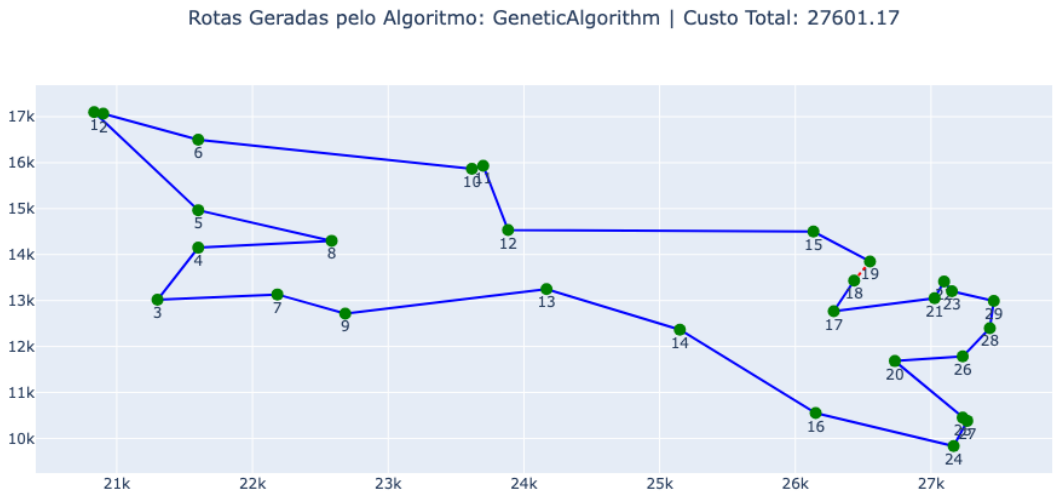

Figura 10 – Melhor solução encontrada pelo algoritmo GA para o problema TSP.

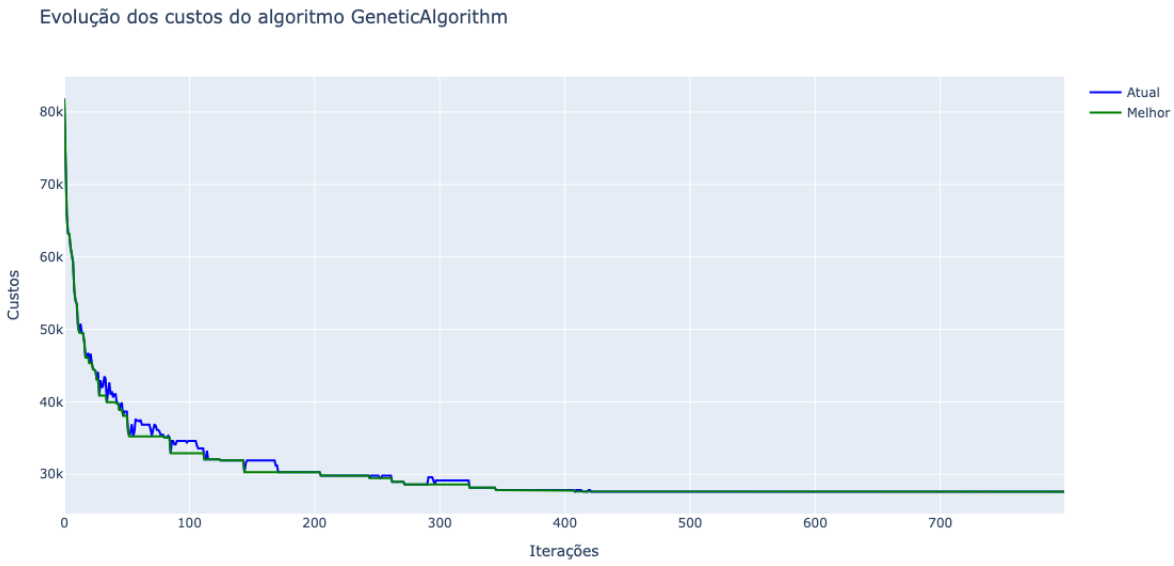

Figura 11 – Evolução da *fitness* da melhor solução do algoritmo GA para o problema TSP.

A Figura 11 demonstra uma rápida convergência do valor de *fitness* no algoritmo genético. Observa-se que, pouco após 400 iterações, o algoritmo já alcança um valor próximo da solução ideal para o problema do TSP apresentado.

4.2 Função de *Rastrigin*

Para a Função de *Rastrigin*, os resultados incluem a análise do valor mínimo da função objetivo encontrado por cada algoritmo. A Tabela 2 apresenta as estatísticas das execuções, e a Figura 12 ilustra a distribuição dos valores com um boxplot.

Tabela 2 – Estatísticas dos valores da função objetivo para a Função de *Rastrigin*.

Algoritmo	**Máximo**	**Mínimo**	**Mediana**	**Desvio Padrão**
HC-C	56.55	2.46	33.06	16.32
HC-R	2.57	0.07	1.52	0.61
SA	1.04	0.99	1.00	0.01
GA	10.04	0.08	1.22	2.53

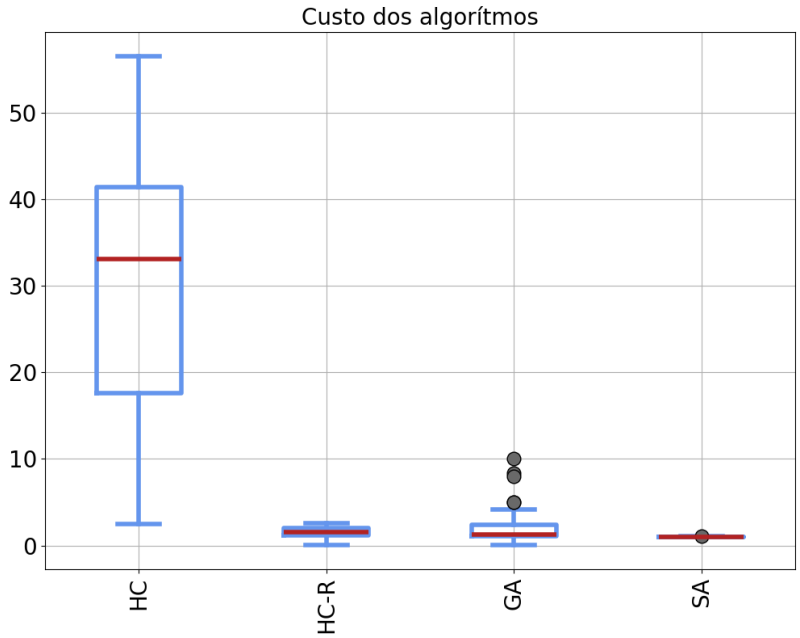

Figura 12 – Boxplot dos valores finais para a Função de *Rastrigin*.

A análise dos resultados, apresentados na Tabela 2 e no Figura 12, revela diferenças significativas no desempenho dos algoritmos HC, HC-R, SA e GA.

O **Simulated Annealing (SA)** destacou-se como o algoritmo com melhor desempenho, apresentando a menor mediana e o menor desvio padrão. Isso indica que o SA foi capaz de encontrar soluções próximas ao ótimo global (0) de forma consistente, com pouca variabilidade entre as execuções. O **Algoritmo Genético (GA)** também obteve resultados competitivos, com uma mediana de 1.22, mas com um desvio padrão maior, sugerindo maior variabilidade nas soluções encontradas.

O **Hill Climbing com Reinicialização (HC-R)** apresentou uma mediana de 1.52, demonstrando uma melhoria significativa em relação ao **Hill Climbing (HC)**, que obteve a pior mediana (33.068431). Essa diferença evidencia a eficácia da estratégia de reinicialização do HC-R, que permitiu escapar de mínimos locais e explorar melhor o espaço de busca.

Considerando os custos computacionais (tempo de execução e acessos à função objetivo, apresentados no Apêndice B), observa-se que SA e GA possuem desempenho semelhante, tornando-os igualmente competitivos. O HC-R, por outro lado, apresenta um tempo de execução mais elevado, embora o número de acessos à função objetivo seja apenas ligeiramente superior ao dos demais. Apesar de seus bons resultados na minimização e da simplicidade de implementação, o custo computacional elevado pode limitar sua atratividade em cenários que demandam eficiência.

4.2.1 Melhor solução e Evolução da *fitness*

Nesta seção, são apresentadas as melhores soluções encontradas por cada algoritmo e a evolução da *fitness* ao longo das iterações. Nos gráficos, a linha verde indica o melhor valor de *fitness* alcançado até o momento, enquanto a linha azul representa o melhor valor de *fitness* da iteração atual. Ambas são plotadas em função do número de iterações, com o eixo X correspondendo ao número de iterações e o eixo Y ao valor de *fitness*.

- **Hill Climb (HC)**: Ponto $[X = -0.9736943161609167, Y = -0.08431592583659131]$

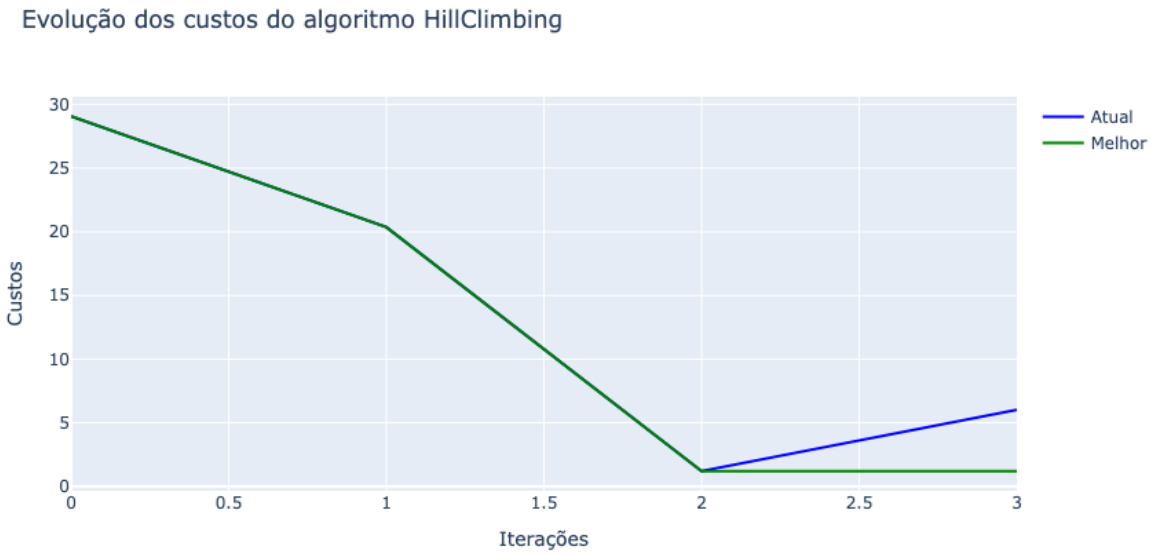

Figura 13 – Evolução da *fitness* da melhor solução do algoritmo HC para o problema *Rastrigin*.

É possível observar na Figura 13 a linha azul se desviando da melhor solução, momento em que o algoritmo atinge seu ponto de parada. Essa é uma característica do algoritmo Hill Climbing (HC): ele busca a melhor solução a cada iteração e, ao não encontrar uma solução melhor, encerra a execução. Além disso, essa abordagem faz com que o HC convirja para o **mínimo local** mais próximo do ponto de partida.

- **Hill Climb com Reinicialização (HC-R)**: Ponto $[X = -0.008829607997035763, Y = -0.017691127961635134]$

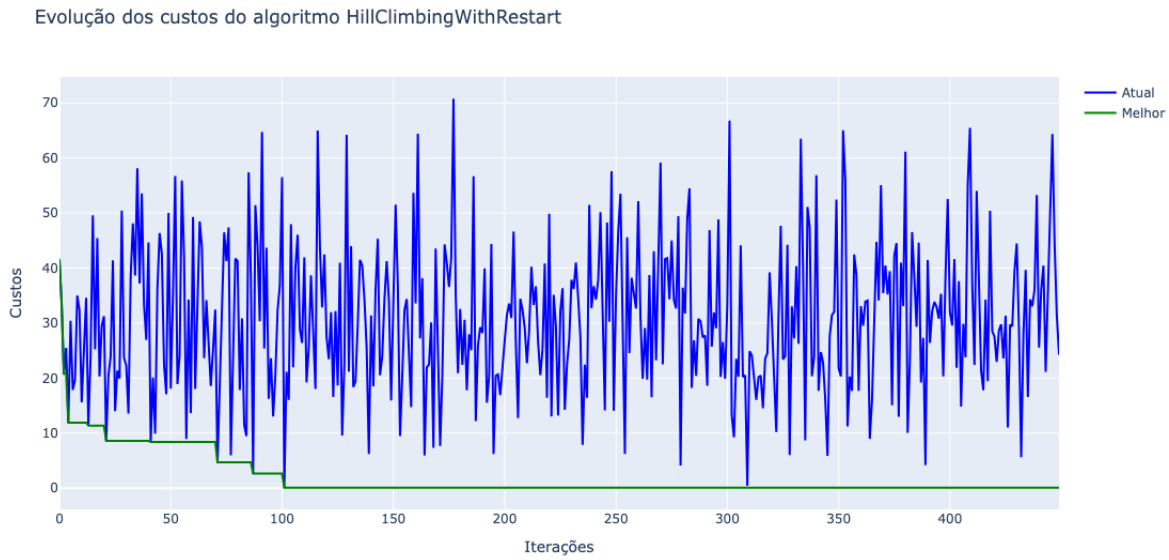

Figura 14 – Evolução da *fitness* da melhor solução do algoritmo HC-R para o problema *Rastrigin*.

Pode-se observar na Figura 14 o comportamento de *restart* do algorítmo HC-R, pois a linha azul se comporta de forma aleatória, sempre encontrando o mínimo local de uma execução do HC a partir de um ponto aleatório. Mas, a melhor solução (em verde) sempre é atualizada assim que uma iteracao é capaz de encontrar um custo menor.

- **Simulated Annealing (SA)**: Ponto $[X = -0.9956241040729951, Y = 0.0004233980189552178]$

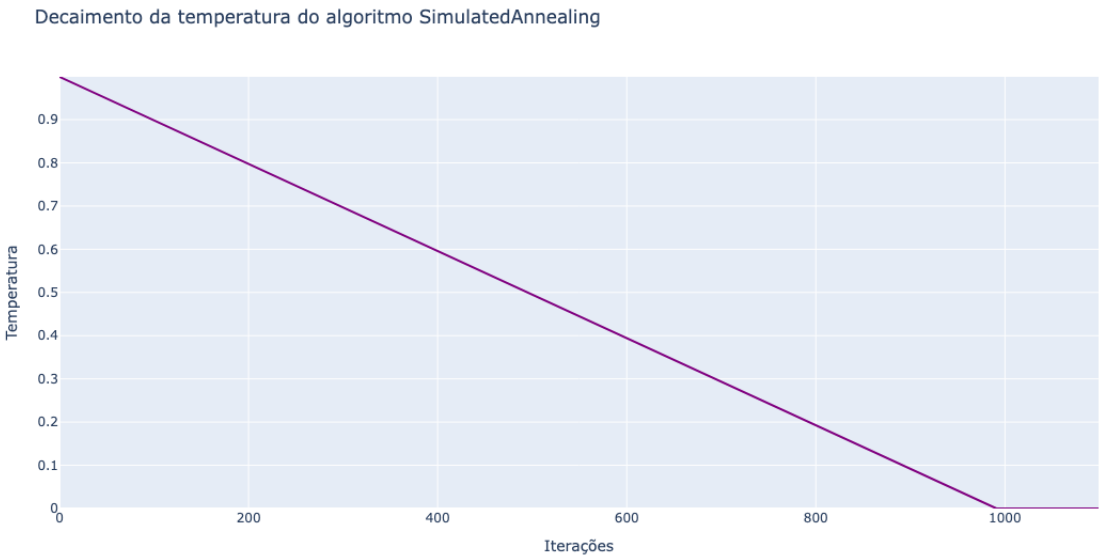

Figura 15 – Evolução da temperatura da melhor solução do algoritmo SA para o problema *Rastrigin*.

Na Figura 15, observa-se o decaimento linear da temperatura ao longo das iterações, iniciando em 1 e atingindo 0 pouco antes de 1000 iterações.

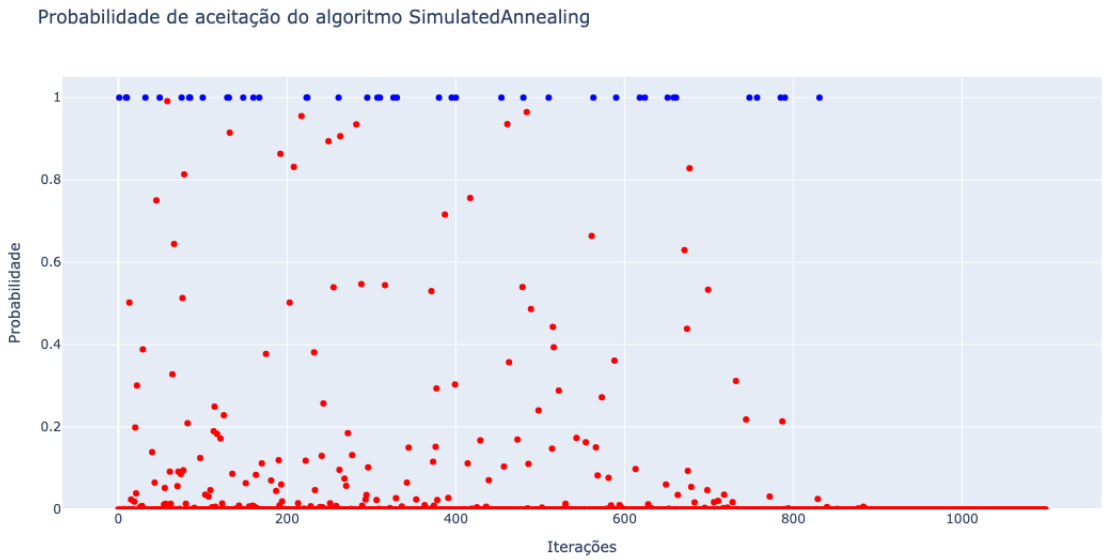

Figura 16 – Probabilidade de aceitação da melhor solução do algoritmo SA para o problema Rastrigin.

A Figura 16 exibe a probabilidade de aceitação (eixo Y) de soluções com custo inferior ao melhor (em vermelho), que decresce ao longo das iterações (eixo X). Em azul, são destacados os casos em que a probabilidade é 1 (100%), ocorrendo quando o novo custo é inferior ao melhor custo.

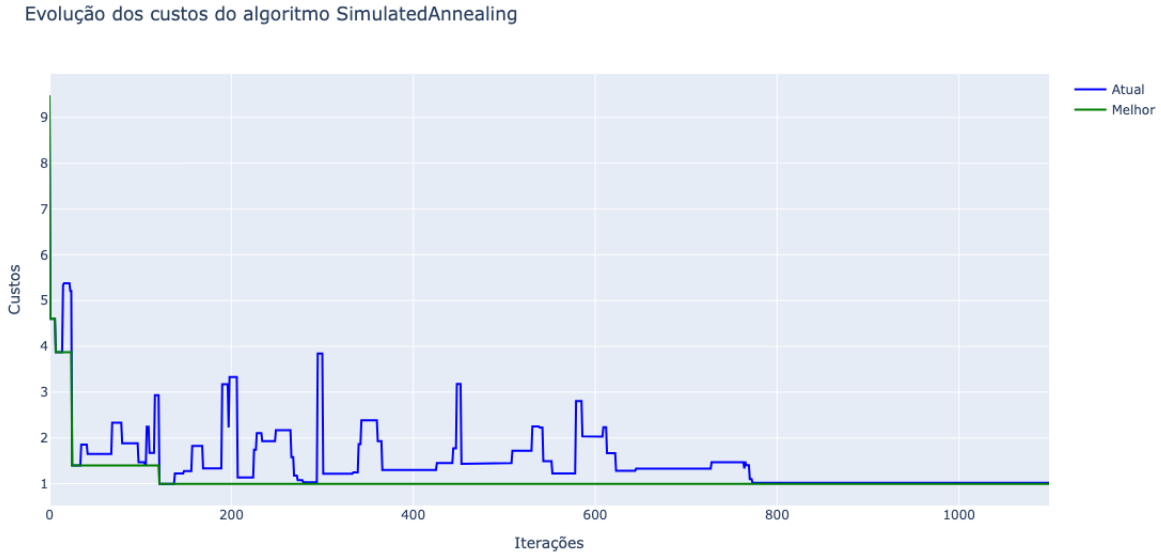

Figura 17 – Evolução da *fitness* da melhor solução do algoritmo SA para o problema *Rastrigin*.

As Figuras 15, 16 e 17 ilustram o comportamento do algoritmo Simulated Annealing (SA). No início, a temperatura está elevada, o que aumenta a probabilidade de aceitar soluções com custo maior, permitindo que o algoritmo escape de mínimos locais. À

medida que a temperatura decai, a probabilidade de aceitar custos superiores diminui gradualmente, enquanto o valor do custo (*fitness*) converge para uma solução otimizada.

- **Algoritmo Genético (GA)**: Ponto $[X = -0.0106596961421816, Y = 0.017239095016591127]$

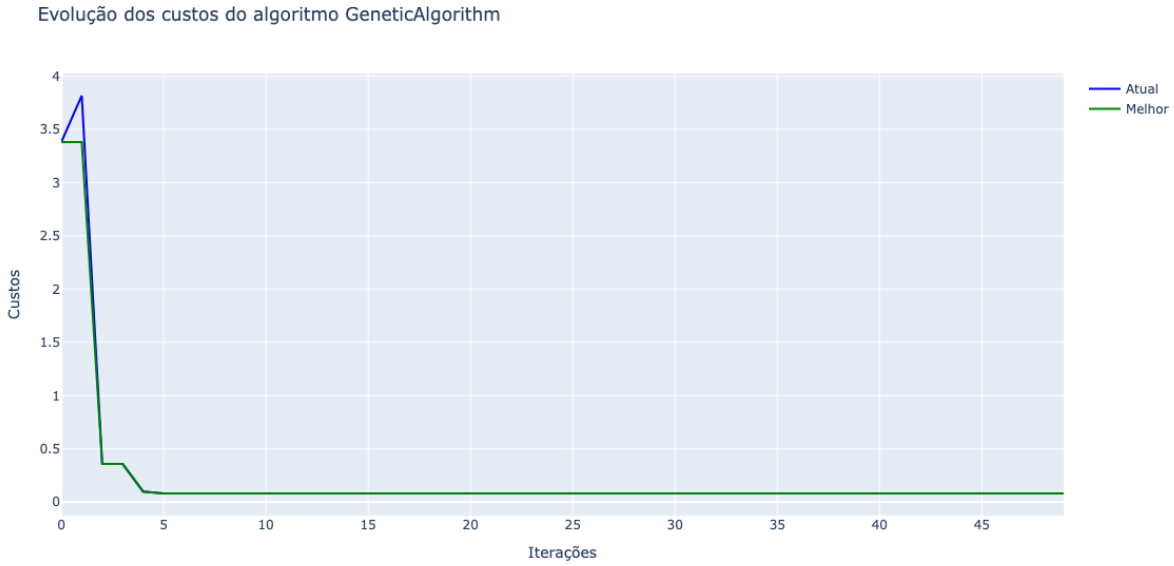

Figura 18 – Evolução da *fitness* da melhor solução do algoritmo GA para o problema *Rastrigin*.

A Figura 18 ilustra a evolução do valor de *fitness* ao longo das iterações do algoritmo genético aplicado à função de *Rastrigin*. Nota-se uma convergência extremamente rápida, em que, após apenas 5 iterações, o algoritmo atinge um valor próximo ao ótimo global. Contudo, embora o algoritmo demonstre eficiência na aproximação inicial, ele não consegue alcançar a solução ideal, que corresponde ao valor 0, indicando uma possível limitação na exploração do espaço de busca em estágios mais avançados da execução.

5 Conclusão

O **Simulated Annealing (SA)** e o **Algoritmo Genético (GA)** são as melhores escolhas para problemas de otimização complexos, como o TSP e a minimização da Função de *Rastrigin*, devido à sua capacidade de escapar de mínimos locais e encontrar soluções próximas ao ótimo global, além do bom desempenho computacional.

O **Hill Climbing com Reinicialização (HC-R)** pode ser uma alternativa viável em cenários onde a simplicidade de implementação é prioritária, mas seu custo computacional mais elevado pode ser um fator limitante.

O **Hill Climbing (HC)** tradicional mostrou-se inadequado para problemas com múltiplos mínimos locais, como o TSP e a Função de *Rastrigin*, devido à sua tendência de convergir para soluções subótimas.

REFERÊNCIAS

RUSSELL, Stuart; NORVIG, Peter. *Artificial Intelligence: A Modern Approach*. 3. ed. [S.l.]: Prentice Hall, 2010.

APÊNDICES

APÊNDICE A – Metadados complementares da execução do TSP

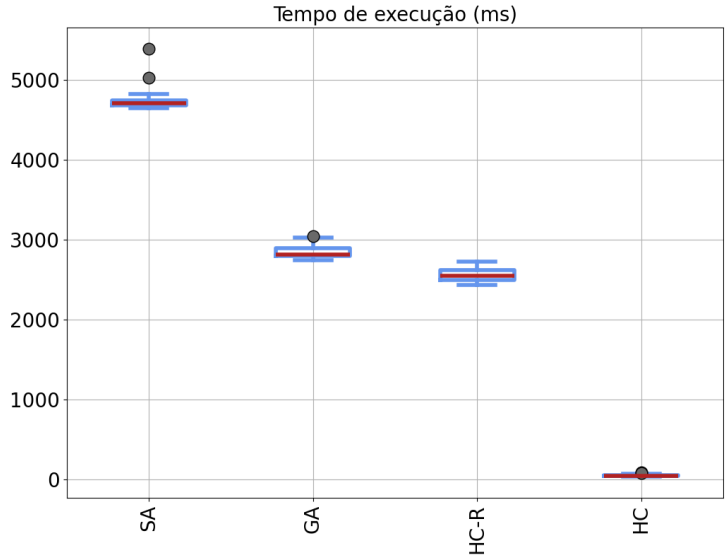

Figura 19 – Boxplot dos tempos de execução dos algoritmos para o problema TSP.

Tabela 3 – Estatísticas dos tempos (em ms) para o TSP.

Algoritmo	Máximo	Mínimo	Mediana	Desvio Padrão
HC	86	37	48	12
HC-R	2719	2433	2542	84
SA	5385	4642	4700	141
GA	3040	2741	2810	84

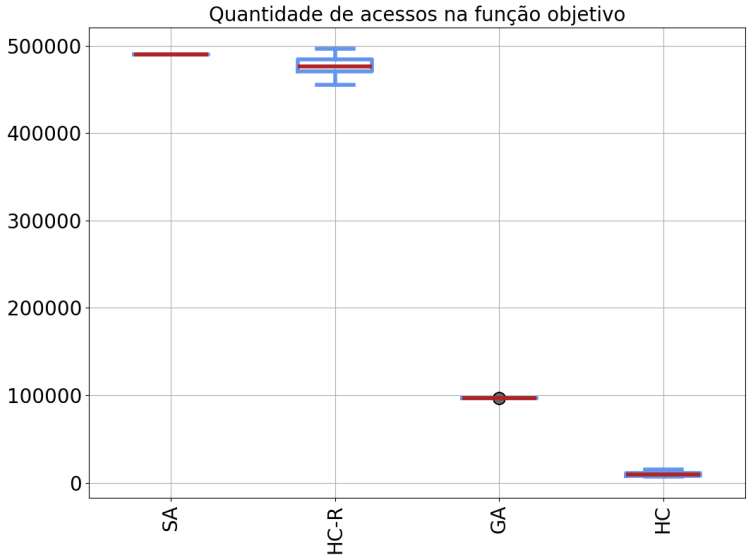

Figura 20 – Boxplot da quantidade de acessos à função objetivo dos algoritmos para o problema TSP.

APÊNDICE B – Metadados complementares da execução do *Rastrigin*

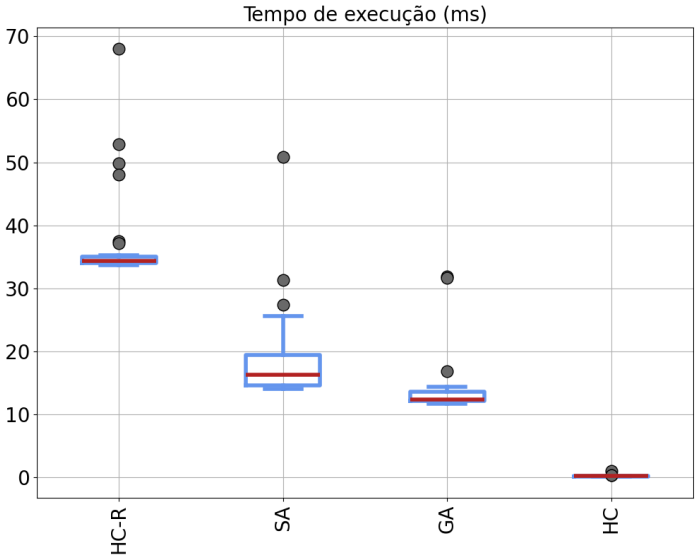

Figura 21 – Boxplot dos tempos de execução dos algoritmos para o problema *Rastrigin*.

Tabela 4 – Estatísticas dos tempos (em ms) para o *Rastrigin*.

Algoritmo	Máximo	Mínimo	Mediana	Desvio Padrão
HC	0.954866	0.134945	0.159979	0.150074
HC-R	68.029165	33.658028	34.373522	7.602483
SA	50.839186	14.060020	16.309023	7.428948
GA	31.838894	11.657953	12.343526	4.928219

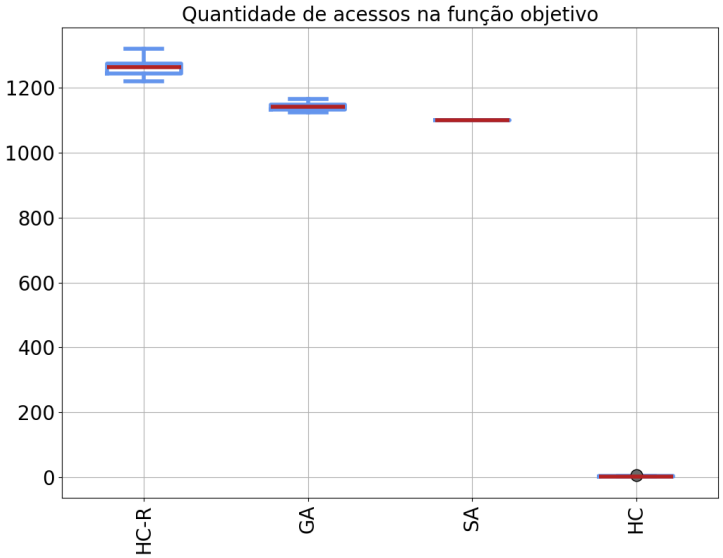

Figura 22 – Boxplot da quantidade de acessos à função objetivo dos algoritmos para o problema *Rastrigin*.