Relatório Simulated Annealing - TSP

Eduardo Schwarz Moreira[1], **Eric Grochowicz**[1]

[1]Departamento de Ciência da Computação
Universidade do Estado de Santa Catarina (UDESC) – Joinville – SC – Brasil

{eduardo.moreira22,eric.g}@edu.udesc.br

Resumo. *Este trabalho tem como objetivo aplicar o algoritmo de Simulated Annealing (SA) na resolução de duas instâncias do Problema do Caixeiro Viajante (TSP), buscando soluções aproximadas próximas da ótima. Foram implementadas três fórmulas de resfriamento distintas, avaliando-se o desempenho de cada uma a partir de múltiplas execuções independentes. Os resultados obtidos permitiram analisar a estabilidade e a eficiência das diferentes estratégias de resfriamento, destacando o impacto dos parâmetros do algoritmo na qualidade das soluções encontradas.*

1. Introdução

O Problema do Caixeiro Viajante (TSP) é um dos problemas clássicos de otimização combinatória, amplamente utilizado como referência para avaliação de algoritmos heurísticos e metaheurísticos. Sua formulação consiste em determinar o caminho de menor custo que percorra um conjunto de cidades, visitando cada uma exatamente uma vez e retornando à cidade de origem. [Walker 2018]

Neste trabalho, buscou-se obter soluções aproximadas próximas da solução ótima para duas instâncias do TSP por meio do algoritmo Simulated Annealing (SA). Essa técnica, inspirada no processo físico de resfriamento de metais, é capaz de escapar de mínimos locais ao aceitar soluções piores de forma probabilística, conforme a temperatura do sistema diminui. [Walker 2018]

Foram implementadas três diferentes fórmulas de resfriamento (*cooling schedules*) e comparados os resultados obtidos com cada uma delas. Para avaliar a estabilidade das soluções, cada configuração foi executada dez vezes, permitindo o cálculo de média e desvio padrão. Assim, buscou-se não apenas avaliar o desempenho do SA, mas também compreender a influência das estratégias de resfriamento sobre a qualidade e consistência dos resultados.

2. Metodologia

Foi desenvolvida uma simulação utilizando a linguagem de programação C++, escolhida tendo em vista a possibilidade de realização de várias iterações da simulação em tempo hábil graças à sua alta performance.

Também, para realizar várias execuções utilizando a mesma fórmula de resfriamento, foi feito código para executar elas em paralelo.

Como programa auxiliar para a visualização, foi desenvolvido um programa em *Python* que gera imagens e vídeos a partir da saída do simulador. Para tal, foi utilizada a biblioteca *matplotlib*.

2.1. Definições Iniciais

2.1.1. Fórmulas de Resfriamento

A temperatura na i-ésima iteração T_i, em função de i, da temperatura inicial T_0 e da temperatura final T_N é dada por:

- Cooling Schedule 0
  - $T_i = T_0 - i(T_0 - T_N)/N$
- Cooling Schedule 1
  - $T_i = T_0(T_N/T_0)^{i/N}$
- Cooling Schedule 5
  - $T_0 = \frac{1}{2}(T_0 - T_N)(1 + \cos(i\pi/N)) + T_N$

As três fórmulas utilizadas foram extraídas de [Luke].

2.1.2. Probabilidade de aceitar uma solução com custo maior

A decisão P de ir de uma solução intermediária com energia E para uma solução vizinha com energia E_{prox} tendo como temperatura atual T é dada por:

$$P(E, E_{prox}, T) = \begin{cases} \text{Verdadeiro} & \text{se } E_{prox} < E \text{ ou uniform}(0,1) \leq \exp(-\frac{E_{prox}-E}{T}) \\ \text{Falso} & \text{se não} \end{cases} \tag{1}$$

Onde uniform(L, R) é um número real aleatório escolhido no intervalo $[L, R]$.

Implementação em C++:

```
 bool shouldAccept(const std::vector<int>& oldTour, const std::
vector<int>& newTour, double temperature) {
    double oldCost = tourCost(oldTour);
    double newCost = tourCost(newTour);
    if (newCost < oldCost) {
        return true;
    }
    double acceptanceProb = std::exp((oldCost - newCost) /
temperature);
    double randomProb = uniform(0.0, 1.0);
    return randomProb < acceptanceProb;
 }
```

Fórmula adaptada de [Proxihox 2025].

2.1.3. Aplicação de ruído na permutação de resposta

Para gerar uma nova solução a partir da atual, foi inicialmente utilizado o método de trocas aleatórias, no qual são escolhidos entre 1 e 5 pares de elementos da permutação para serem trocados entre si.

No entanto, conforme descrito por [Walker 2018], existem outras estratégias de perturbação mais estruturadas, que buscam explorar o espaço de busca de forma mais eficiente. Nesse caso, o método seleciona aleatoriamente entre duas possíveis transformações aplicadas à permutação atual:

- **Reversão**: seleciona-se aleatoriamente um intervalo $[L, R]$ da permutação, e os elementos contidos nesse intervalo têm sua ordem invertida.
- **Transporte**: seleciona-se aleatoriamente um intervalo $[L, R]$, remove-se esse trecho da permutação e insere-se o mesmo em uma nova posição aleatória.

Essas operações têm o objetivo de introduzir diversidade nas soluções geradas, evitando a estagnação em mínimos locais e permitindo uma exploração mais ampla do espaço de soluções.

Implementação em C++:

```cpp
void applyPermutationNoise(std::vector<int>& tour) {
    int number_swaps = uniform(1, 5);
    while (number_swaps--) {
        int i = uniform(0, N - 1);
        int j = uniform(0, N - 1);

        bool reverse = uniform(0, 1);
        if (i > j) std::swap(i, j);
        if (reverse) {
            std::reverse(tour.begin() + i, tour.begin() + j + 1);
        } else {
            std::vector<int> segment(tour.begin() + i, tour.begin() + j + 1);
            tour.erase(tour.begin() + i, tour.begin() + j + 1);
            int k = uniform(0, int(tour.size()));
            tour.insert(tour.begin() + k, segment.begin(), segment.end());
        }
    }
}
```

2.2. Parâmetros

2.2.1. Temperatura inicial e final

Foram obtidos experimentalmente os valores de T_0 e T_N como $0,5$ e 10^{-10}, respectivamente. A temperatura final foi definida como a menor possível a fim de se aproximar mais da solução ótima.

2.2.2. Número de iterações

Foi definido a fim de que as fórmulas de resfriamento já tenham atingido uma temperatura estável quando a iteração N é alcançada. O valor definido foi de $1,5 \cdot 10^6$.

3. Resultados obtidos

3.1. Instância - 51 cidades

Foi executado o Simulated Annealing para resolver a instância do Caixeiro Viajante com 51 cidades (eil51). As três fórmulas de resfriamento estão identificadas por cor no gráfico.

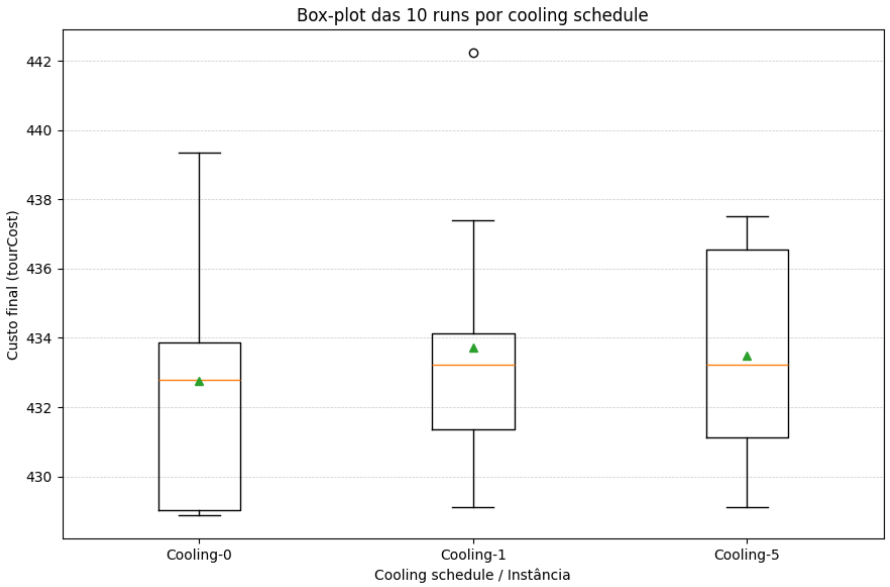

Figura 1. Boxplot das 10 execuções para cada função de resfriamento.

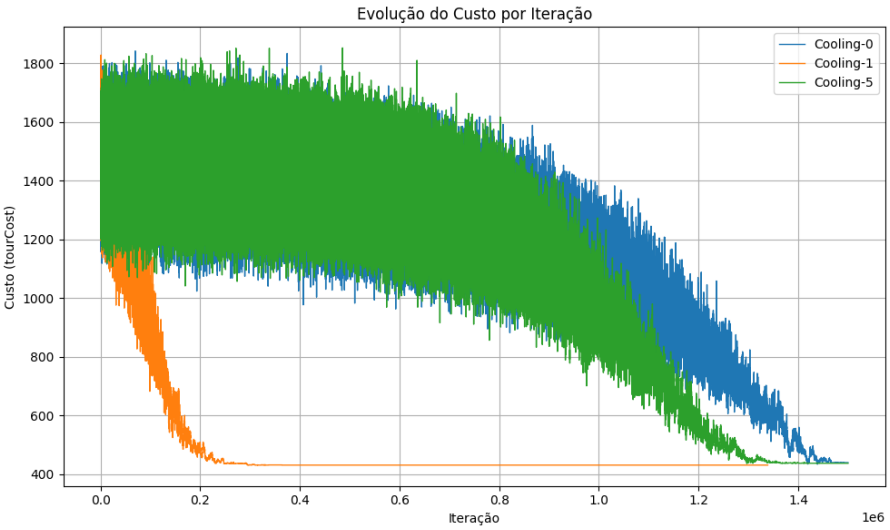

Figura 2. Gráfico das 10 execuções para cada função de resfriamento.

Tabela 1. Resultados das 10 execuções para cada função de resfriamento.

Label	Count	Mean	Std	Mean $\pm$ Std
Cooling-0	10	432.768	4.016	432.768 ± 4.0158
Cooling-1	10	433.729	3.786	433.729 ± 3.7859
Cooling-5	10	433.488	3.140	433.488 ± 3.1401

3.2. Instância - 100 cidades

Foi executado o Simulated Annealing para resolver a instância do Caixeiro Viajante com 100 cidades (kroA100). As três fórmulas de resfriamento estão identificadas por cor no gráfico.

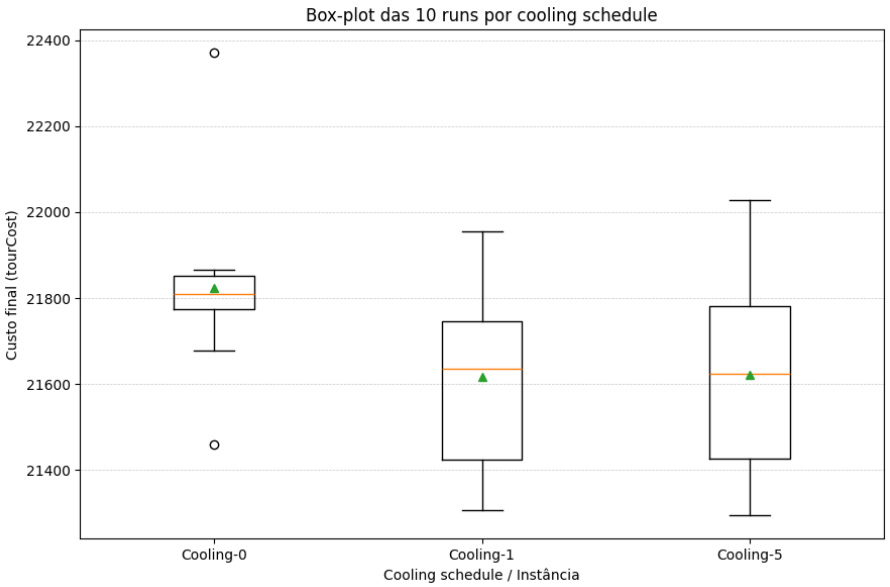

Figura 3. Boxplot das 10 execuções para cada função de resfriamento.

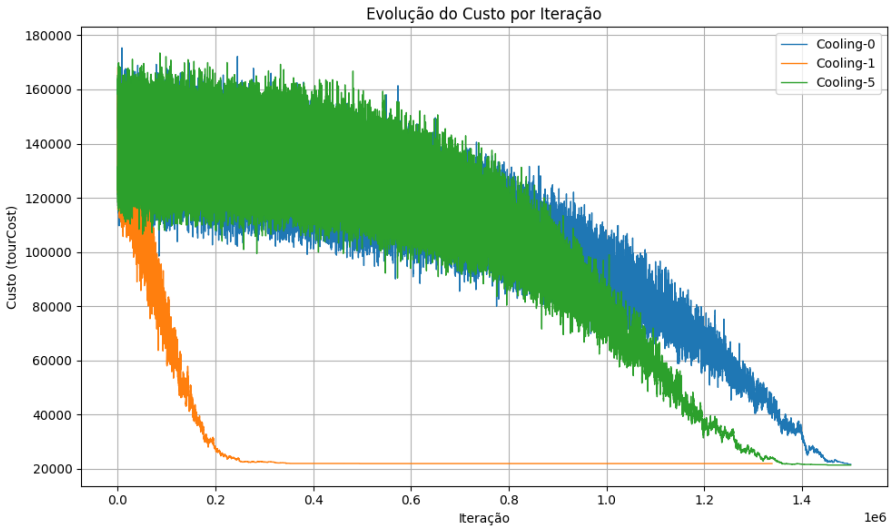

Figura 4. Gráfico das 10 execuções para cada função de resfriamento.

Tabela 2. Resultados das 10 execuções para cada função de resfriamento.

Label	Count	Mean	Std	Mean $\pm$ Std
Cooling-0	10	21824.260000	226.256910	21824.3 ± 226.257
Cooling-1	10	21615.920000	222.452910	21615.9 ± 222.453
Cooling-5	10	21824.260000	230.880451	21620.7 ± 230.88

4. Conclusão

O presente trabalho permitiu observar, na prática, a aplicação do algoritmo Simulated Annealing na resolução de instâncias do Problema do Caixeiro Viajante. Os resultados experimentais confirmaram a capacidade do método em encontrar soluções de boa qualidade em tempo viável, mesmo para instâncias em que métodos exatos, como força bruta ou programação dinâmica, se tornam inviáveis pela escala do problema. foi observada Verificou-se que a escolha dos parâmetros tem influência considerável sobre a otimização do algoritmo. Além disso, as diferentes fórmulas de resfriamento apresentaram variações sutis na qualidade das soluções e na estabilidade entre execuções, demonstrando que a estratégia de resfriamento pode ser ajustada conforme o perfil da instância e o tempo disponível de execução.

Em resumo, o estudo reforça a eficácia do Simulated Annealing como ferramenta heurística para problemas de otimização complexos, bem como a importância da experimentação na calibração de seus parâmetros e funções de resfriamento.

Referências

Luke, B. T. Simulated annealing cooling schedules. `http://www.btluke.com/simanf1.html`.

Proxihox (2025). Simulated annealing. `https://cp-algorithms.com/num_methods/simulated_annealing.html`.

Walker, J. (2018). Simulated annealing: The traveling salesman problem. `https://www.fourmilab.ch/documents/travelling/anneal/`.