Uso da Otimização por Enxame de Partículas para solucionar o problema do Caixeiro Viajante

Davi dos Reis[1], **Gabriel de Paula**[1], **Guilherme Francis**[1], **Wasterman Apolinário**[1]

[1]Universidade Federal de São João Del Rei (UFSJ)
São João del-Rei – MG – Brasil

{davireisjesus, gabriel.meira.2004,

guilherme2036, wastermanavila}@aluno.ufsj.edu.br

Resumo. *O Problema do Caixeiro Viajante (TSP) é um desafio computacional que visa determinar a rota mais curta para um caixeiro viajante visitar cada cidade uma vez e retornar à cidade de origem. Devido à sua NP-completude, não há um algoritmo determinístico que resolva o problema em tempo polinomial. Por isso, métodos de aproximação, como heurísticas e algoritmos de otimização, são usados para encontrar soluções próximas do ótimo. Este artigo explora a aplicação da Otimização por Enxame de Partículas (PSO) ao TSP. O PSO otimiza iterativamente o problema, navegando por soluções candidatas, chamadas partículas, usando fórmulas matemáticas para ajustar suas posições e velocidades. O PSO se mostra uma abordagem promissora para aproximar soluções de forma eficiente para o TSP.*

Abstract. *The Traveling Salesman Problem (TSP) is a computational challenge that aims to determine the shortest route for a salesman to visit each city once and return to the origin. Due to its NP-completeness, no deterministic algorithm can solve the problem in polynomial time. Thus, approximation methods, such as heuristics and optimization algorithms, are used to find near-optimal solutions. This article explores the application of Particle Swarm Optimization (PSO) to the TSP. PSO iteratively optimizes the problem by navigating through candidate solutions, called particles, using mathematical formulas to adjust their positions and velocities. PSO proves to be a promising approach for efficiently approximating solutions to the TSP.*

1. Introdução

Como se sabe, o Travelling Salesman Problema (TSP) é um problema clássico do mundo computacional pelo fato de ser NP completo, ou seja, não existe um algoritmo determinístico que resolva o problema em tempo polinomial. A formulação básica do TSP é a seguinte: dado um conjunto de cidades e a distância entre cada par que possua um conexão, o objetivo é encontrar a rota mais curta que permite ao caixeiro viajante visitar cada cidade exatamente uma vez e retornar à cidade de origem.

Dado essa dificuldade de resolução, utiliza-se métodos que se aproximem, ao máximo, da solução ótima como, por exemplo, heurísticas e otimizadores existentes. Um exemplo é a Otimização por Enxame de Partículas (PSO), que propõe otimizar um problema iterativamente elegendo candidatos (partículas) que se movem por um determinado espaço de busca de acordo com sua posição e velocidade.

A aplicação do PSO ao TSP envolve adaptações específicas que serão descritas neste artigo, permitindo que o algoritmo aproveite seu potencial para explorar o espaço de busca de forma eficaz e encontrar rotas próximas de serem ótimas para o caixeiro viajante. Portanto, investigar sua aplicação específica ao TSP não apenas contribui para o avanço do conhecimento na área, mas também oferece uma potencial ferramenta prática para resolver problemas reais de roteamento e logística, onde ter rotas eficientes é crucial.

2. Referencial Teórico

O objetivo é encontrar a rota mais curta possível que permita visitar todas as cidades e retornar ao ponto de partida. Matematicamente, o TSP pode ser representado por um grafo $G = (V, E)$, onde V é o conjunto de vértices representando as cidades e E é o conjunto de arestas com pesos correspondendo às distâncias entre as cidades. Considere as seguintes cidades: A, B, C e D, assim como mostra a Figura 1.

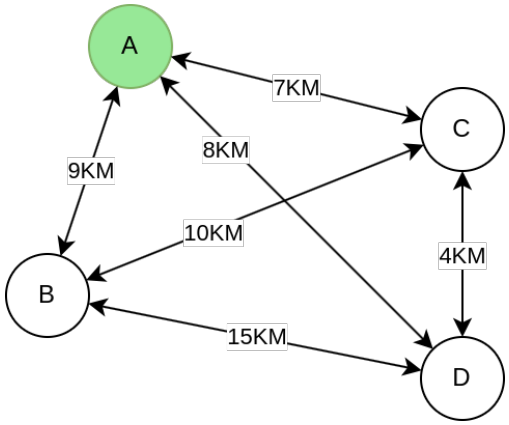

Figura 1. Exemplo de disposição das cidades

A matriz de adjacência da Tabela 1 exibe a distância de todas as arestas do grafo acima, note que a matriz é simétrica em relação à diagonal principal, já que a conexão entre as cidades pode ser feita em qualquer sentido.

Tabela 1. Matriz de adjacência do exemplo

×	**A**	**B**	**C**	**D**
A	0	9	7	8
B	9	0	10	15
C	7	10	0	4
D	8	15	4	0

As possíveis soluções (rotas) e seus respectivos custos (distância total percorrida) podem ser visualizadas na Tabela 2. Cada caminho é uma solução, sendo a função de avaliação definida pela soma total da distância percorrida.

Tabela 2. Soluções do exemplo e seus custos

Início					**Custo**
A	B (9)	C (10)	D (4)	A (8)	31
A	B (9)	D (15)	C (4)	A (7)	35
A	C (7)	B (10)	D (15)	A (8)	40
A	C (7)	D (4)	B (15)	A (9)	35
A	D (8)	B (15)	C (10)	A (7)	40
A	D (8)	C (4)	B (10)	A (9)	31

Já de antemão é possível perceber que a quantidade de caminhos possíveis, em um grafo completo, cresce de forma significativa, existindo ao todo $(n-1)!$ soluções. Atesta-se, portanto, a inviabilidade da utilização de algoritmos de força bruta para a busca. Dessa forma, o PSO surge como uma boa forma de otimização para o problema.

Apesar de não garantir que a solução ótima será encontrada, o PSO quando usado corretamente pode produzir resultados satisfatórios na busca do caminho mínimo, tendo seu funcionamento dividido em alguns passos:

1. Inicialização: As partículas são distribuídas aleatoriamente no espaço de busca, com posições e velocidades iniciais atribuídas.
2. Avaliação: A aptidão de cada partícula é avaliada usando uma função de custo específica do problema.
3. Atualização de Velocidade e Posição: A velocidade de cada partícula é atualizada com base em três componentes: inércia (mantém a partícula se movendo na mesma direção), componente cognitiva (atrai a partícula para sua melhor posição individual), componente social (atrai a partícula para a melhor posição global).
4. Iteração: Este processo de avaliação e atualização continua até que um critério de parada seja atendido, como um número máximo de iterações ou a convergência das partículas para uma solução.

A sequência de etapas é simples e de fácil implementação, sendo abordada com mais detalhes na metodologia, principalmente a adaptação do otimizador ao problema específico proposto. É importante ressaltar que, devido ao processo de inicialização usar caminhos aleatórios, o algoritmo não é

Com base no funcionamento evidenciado e nas adaptações necessárias do otimizador, utiliza-se o PSO para encontrar o máximo ou mínimo de uma função. Função que é definida em um espaço vetorial multidimensional da seguinte forma: Suponha que temos uma função f que produz um valor real a partir de um parâmetro vetor x (como uma coordenada em um plano) e $f(x)$ pode assumir virtualmente qualquer valor no espaço (por exemplo, $f(x)$ é a altitude e podemos encontrar uma para qualquer ponto no plano), então podemos aplicar o PSO, que retornará o parâmetro x que encontrou e que produz o mínimo ou o máximo $f(x)$.

3. Metodologia

É necessário, antes de iniciar a programação, planejar o funcionamento do programa em cada uma das etapas, a fim de garantir a qualidade do software do início ao fim do projeto, facilitando o processo de depuração e manutenção. O programa foi dividido em quatro etapas principais, ilustradas pelo fluxograma da Figura 2.

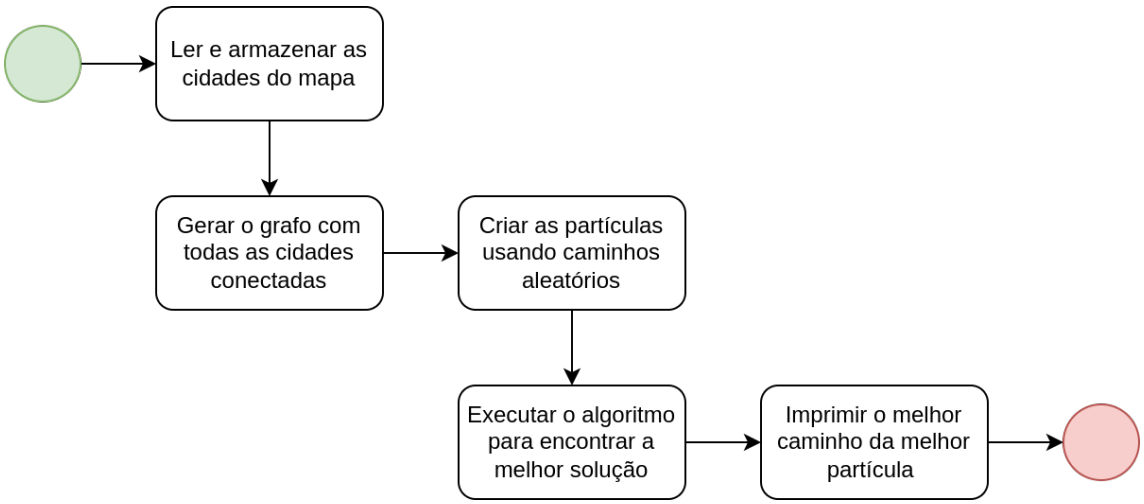

Figura 2. Fluxograma geral do programa

Essa divisão proporciona uma visão clara e estruturada do processo, no qual cada etapa representa um conjunto de tarefas e funcionalidades específicas a serem implementadas permitindo uma abordagem mais organizada e eficiente para o desenvolvimento.

3.1. Partícula e Velocidade

Visando garantir uma organização eficiente e uma maior escalabilidade do software, foram criadas estruturas de dados destinadas unicamente para armazenar os dados de cada partícula. Isso permite que as informações possam ser facilmente acessadas e manipuladas durante toda a execução do programa.

Como descrito anteriormente, cada partícula é iniciada com um caminho aleatório e, com o cálculo de sua velocidade, é capaz de ir para caminhos melhores (ou não). Dito isso, cada partícula deve saber qual é o caminho atual e ser capaz de avaliá-lo (distância total) para definir qual foi a melhor solução já encontrada, resultando na classe:

```
class Particle {
    // ...
    Path actualPath, personalBestPath;
    Velocity velocity;
    // ...
}
```

Por sua natureza discreta, o problema necessita de uma adaptação no conceito de velocidade, fugindo da definição vetorial usada em outros contextos. A abordagem adotada foi a de que seja formada por operadores de troca [Goldbarg et al. 2008].

```
class Velocity {
    // ...
    vector<SwapOperator> swapOperators;
    // ...
}
```

Um operador de troca pode ser entendido como uma possibilidade de mudar a posição de um vértice no caminho. Sua probabilidade é definida por um coeficiente no intervalo $[0, 1]$, quanto mais próximo de 1, maiores são as chances de que essa troca ocorra. A cada iteração do algoritmo, a partícula tem sua velocidade limpa para dar espaço para novos operadores.

```
typedef struct {
    Vertex vertex;
    int indexOrigin, indexCorrect;
    double coefficient;  // 0-1 scale
} SwapOperator;
```

3.2. Algoritmo

De forma prática, o fluxo de execução é representado pela Figura 3. A cada iteração o programa encontra o menor custo de caminho armazenado por cada partícula e, para cada partícula, atualiza suas velocidades com base em constantes para então realizar as trocas.

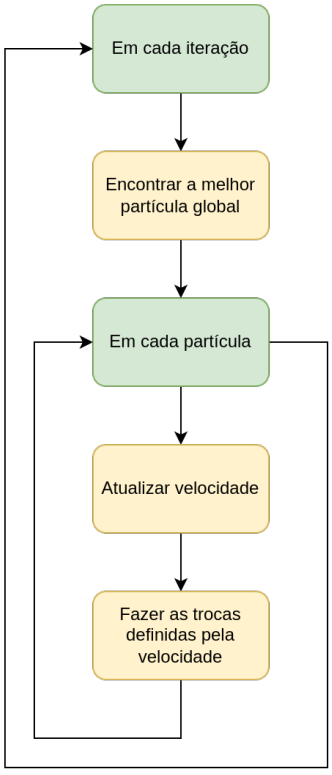

Figura 3. Fluxograma de uma iteração

As trocas podem ocorrer levando em consideração os fatores cognitivos (melhor solução pessoal) e sociais (melhor solução global). A cognição prepara a partícula para evitar soluções piores em um escopo local. A socialização das partículas é essencial para a fazer com que elas tentem convergir na direção de uma solução global.

O processo iterativo possui complexidade $O(ip|V|^2)$, onde i é a quantidade de iterações, p é o tamanho da população e $|V|$ é a quantidade de vértices. Além disso, duas constantes são definidas para controlar a probabilidade das trocas, c_1 e c_2, dos fatores cognitivo e social, respectivamente. Valores altos em c_1 e baixos em c_2 fazem com que a partícula seja mais inflexível, ou seja, diminui a aleatoriedade da solução, já o contrário faz com que ela se torne mais influenciável pelas outras.

4. Resultados

Encontrar um balanceamento entre os dois coeficientes, conjuntamente com a quantidade de iterações e população, é uma tarefa complexa e que demanda uma bateria de testes para tal, dado que só é possível determinar um bom balanceamento realizando uma comprovação empírica. Para esse propósito, foi implementada uma busca em grade (*Grid Search*).

4.1. Implementação do *Grid Search*

O *Grid Search* tem como objetivo encontrar a melhor combinação de parâmetros para uma determinada tarefa através da exploração exaustiva de todas as possíveis combinações definidas pelo usuário. Para o problema em questão, foram selecionados os seguintes parâmetros com ênfase na análise detalhada, e não na otimização de desempenho:

```
vector<int> populationSizes = {10,20,40,60,80};
vector<int> iterationsList = {10,20,40,60,80};
vector<double> c1List = {0.5,0.8,1.5};
vector<double> c2List = {0.5,0.8,1.5};
```

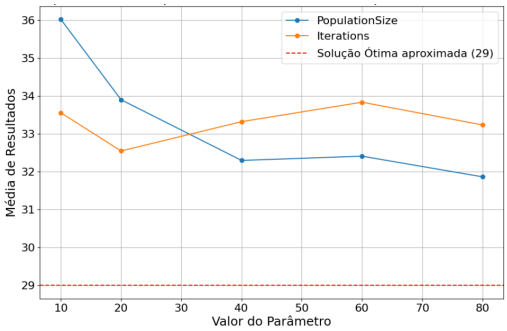

Figura 4. Impacto dos Parâmetros para "instância.txt"

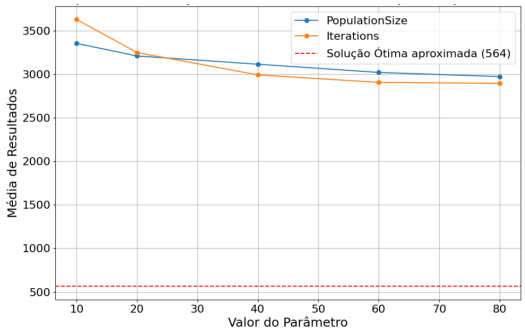

Figura 5. Impacto dos Parâmetros para "xqf131.txt"

Os gráficos das Figuras 4 e 5 mostram a média dos resultados obtidos ao combinar os parâmetros em destaque (Tamanho da População e Iterações) com outras possibilidades, em dois grafos distintos. Após a implementação do *Grid Search* e a análise dos resultados obtidos nas figuras, pode-se inferir que, com o aumento do valor dos parâmetros, principalmente do tamanho da população (*PopulationSize*), as soluções tendem a se aproximar da solução ótima.

Tabela 3. Melhores parâmetros encontrados em cada instância

Instância	p	i	c_1	c_2	Resultado	Solução ótima
“xqf131.txt”	80	60	0.8	0.8	2429.4	564
“instancia.txt”	80	60	0.8	0.5	29.2487	29.2487

A Tabela 3 apresenta a combinação de parâmetros que obteve o melhor resultado para cada um dos grafos. Vale destacar que, para o grafo “instancia”, que contém apenas 12 cidades, várias combinações alcançaram a solução ótima. Em contraste, para o grafo “xqf131”, que possui 131 cidades, a distância entre o resultado e a solução ótima é maior. Isso indica que o método demonstrou maior eficácia para grafos menores. No entanto, isso não diminui sua utilidade para grafos maiores. Em casos onde outras estratégias se mostram inviáveis, a heurística de PSO ainda pode fornecer soluções aceitáveis em um tempo de computação relativamente curto.

4.2. Representação de Soluções

O entendimento do problema do TSP pode ser bem complexo, principalmente com uma quantidade de cidades extremamente grande. Assim, visualizar graficamente as soluções, apesar de ser uma tarefa opcional, é de grande relevância para fins de documentação e análise dos resultados obtidos em cada execução do algoritmo, facilitando a interpretação desses resultados gerados e a demonstração prática da eficácia do método de resolução.

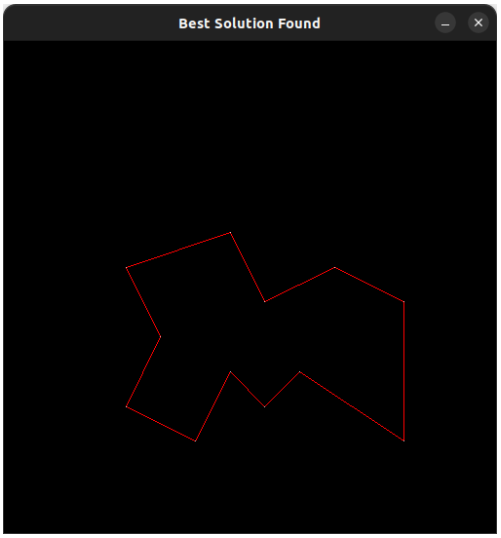

Figura 6. Visualização gráfica de uma solução

Para gerar a representação de uma solução, foi utilizada a biblioteca **SDL** (Simple DirectMedia Layer). Essa biblioteca foi escolhida por sua praticidade na criação de gráficos. Utilizando SDL, os pontos (representando as cidades) foram marcados e as linhas vermelhas foram traçadas para indicar o melhor caminho encontrado.

5. Considerações Finais

Ao analisar o funcionamento do PSO sobre o problema do TSP, podemos inferir que, apesar de não necessariamente resultar em uma saída ótima, as melhorias que a proposta

provê trazem bons resultados em tempo polinomial, diferente da resolução padrão estabelecida para o problema. Sendo assim, podemos determinar que a meta-heurística é sim uma boa escolha de solução.

O algoritmo possui pontos de destaque relevantes, como a eficiência na exploração do espaço de busca por mostrar-se capaz de explorar eficientemente o espaço de busca, identificando rotas de custo reduzido ao longo das iterações e permitindo aleatoriedade para procurar caminhos diversos e evitar ótimos locais. Flexibilidade e adaptabilidade por fornecer a possibilidade de adaptar o funcionamento do otimizador à certos problemas, o que permite a sua utilização em diversas tarefas.

Além disso, fornece resultados promissores, pois a aproximação da solução ótima em tempo polinomial evidencia a efetividade do algoritmo em resolver o problema. Isso é essencial, já que possibilita obter esses resultados em casos reais, como cálculo de rotas feitos por empresas, que são problemas do cotidiano e, costumeiramente, são semelhantes ao problema do TSP.

Referências

Alibrahim, H. and Ludwig, S. A. (2021). Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1551–1559.

Goldbarg, E. F. G., Goldbarg, M. C., and Souza, G. R. (2008). *Particle Swarm Optimization Algorithm for the Traveling Salesman Problem*. InTech.

S.G.T, F. and r, E. C. (2011). *APLICAÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE NA OTIMIZAÇÃO DE ROTEIROS*. ITPI.

Wang, K.-P., Huang, L., Zhou, C.-G., and Pang, W. (2003). Particle swarm optimization for traveling salesman problem. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*, volume 3, pages 1583–1585 Vol.3.