Uma análise do método de Enxame de Partículas para a solução do Problema do Caixeiro Viajante

Adélson de Oliveira Carmo Júnior[1], **Renan Rodrigues Sousa**[1], **Rodrigo José Zonzin Esteves**[1], **Wandra Martins Dias**[1]

[1]Departamento de Computação – Universidade Federal de São João del Rei (UFSJ)
São João del Rei – Brazil

Resumo. *A compreensão de fenômenos complexos em diversas áreas tem levado a um enfoque em métodos matemáticos para representar esses sistemas. Métodos de Inteligência Artificial (IA), como a Otimização por Enxame de Partículas (PSO), têm sido utilizados com sucesso para a solução de problemas de otimização. Este trabalho visa resolver o problema do caixeiro-viajante (TSP) por meio do PSO a fim de se analisar os parâmetros que impactam na convergência do algoritmo. Os resultados mostram que o PSO pode ser eficaz, dependendo do ajuste adequado entre o número de cidades, iterações e partículas. Um maior número de partículas tende a melhorar o desempenho, enquanto um aumento no número de cidades reduz a eficiência.*

1. Introdução

A compreensão de fenômenos físicos, sociais, econômicos e logísticos tem sido uma das principais preocupações das sociedades ao longo do tempo. A busca por explicar, predizer e sistematizar esses fenômenos impulsionou uma abordagem cada vez mais quantitativa dos problemas que envolvem a experiência humana. Nesse contexto, a criação de modelos matemáticos que representam com maior precisão os sistemas complexos observados na natureza é talvez um dos estágios mais avançados de representação do conhecimento humano.

Entre outras abordagens, a Teoria dos Grafos é uma área da Matemática que estuda as relações entre um sistema de variáveis contidas em um determinado conjunto [Diestel 2017]. Nesta abordagem, consideram-se dois conjuntos V e E, onde V é um conjunto de elementos denominados vértices e E é formado por pares não ordenados de vértices, *i.e* arestas, $u, v \in V$. Essa abordagem foi utilizada com sucesso para diversas aplicações no mundo real: modelagem de redes de distribuição elétrica [Avella P. and Sforza 2005], segmentação de imagens baseadas em cortes [Felzenszwalb and Huttenlocher 2004], análise do impacto da centralidade dos vértices na propagação de Covid-19 [Yücel et al. 2023]

Além disso, métodos baseados em Inteligência Artificial (IA) têm sido utilizados para aplicações científicas de classificação, predição, clusterização e otimização. Dentre eles, a Otimização por Enxame de Partículas (PSO, na sigla em inglês) é uma abordagem bioinspirada que visa aproximar a solução ótima de um problema, melhorando iterativamente as soluções candidatas com respeito a uma medida de qualidade (fitness).

O problema do caixeiro-viajante (TSP) é um dos problemas clássicos da otimização combinatória [Goldbarg and Goldbarg 2012]. Tal problema consiste em determinar um ciclo hamiltoniano de menor custo em um grafo ponderado $G = (V, E)$. O

TSP é um problema *NP-Difícil*, ou seja, não é possível determinar sua solução analítica em tempo polinomial [Cormen 2012].

Considerando estes aspectos, o objetivo deste trabalho é formular uma solução heurística para o TSP por meio da Otimização Por Exame de Partículas a fim de se analisar quais os parâmetros impactam na convergência do algoritmo para um valor ótimo.

2. Dados e Metodologia

2.1. Dados

Os dados utilizados neste trabalho incluem as coordenadas das cidades fornecidas em um arquivo de texto (instancia.txt). A leitura do arquivo é realizada pela função `read_cities`, que extrai o número de cidades e suas coordenadas, armazenando-as em uma estrutura adequada para posterior processamento.

2.2. Construção do Grafo

A construção do grafo é feita por meio de uma matriz de distâncias euclidianas entre todas as cidades, calculadas a partir de suas respectivas coordenadas. Para isso, a distância entre duas cidades é calculada usando a rotina `calculate_distance`, que aplica a fórmula de distância euclidiana (Equação 1).

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

A função `build_graph` é utilizada para criar o grafo completo.

2.3. Função de Avaliação

A função de *fitness*, `fitness`, é definida para calcular a distância total percorrida em um caminho específico. Esta função é usada para avaliar e comparar diferentes caminhos, com o objetivo de minimizar a distância total percorrida [Russell and Norvig 2013]. A Equação 2 apresenta o cálculo do *fitness*.

$$\textit{fitness}(caminho) = \sum_{i=1}^{n-1} d(cidade_i, cidade_{i+1}) + d(cidade_n, cidade_1) \tag{2}$$

2.4. Operadores Genéticos

Um operador de mutação é implementado pela função `swap_mutation`, que troca duas cidades aleatórias no caminho, introduzindo diversidade na população de soluções. Este operador de mutação é aplicado a uma taxa específica para evitar a convergência prematura do algoritmo.

2.5. Representação das Partículas

Cada partícula representa uma solução potencial para o TSP, que é um caminho percorrendo todas as cidades. Cada partícula possui uma posição (o caminho atual), uma velocidade (a direção e magnitude das mudanças a serem aplicadas) e uma melhor posição conhecida (melhor caminho encontrado pela partícula até o momento). A classe `Particle` é utilizada para representar as partículas.

2.6. Inicialização das Partículas

Um conjunto de partículas é inicializado com caminhos aleatórios usando a função `initialize_particles`. Uma distribuição aleatória é usada para garantir a diversidade inicial das soluções. As partículas são inicializadas com caminhos que percorrem todas as cidades em uma ordem aleatória.

2.7. Criação e Aplicação da Velocidade

A velocidade das partículas é calculada com base na diferença entre suas posições atuais e suas melhores posições conhecidas a partir da função `create_velocity`. A aplicação das velocidades atualiza as posições das partículas, gerando novas soluções potenciais. A velocidade é representada como uma lista de operações de troca (swap), que devem ser aplicadas para transformar a posição atual da partícula em sua melhor posição conhecida. A função `apply_velocity` é usada para aplicar a velocidade.

2.8. Algoritmo PSO

O processo iterativo do PSO envolve a atualização das posições e velocidades das partículas a cada iteração, executado pela função `pso_tsp`. As partículas atualizam suas melhores posições conhecidas e a melhor posição global com base na função de *fitness* [de Sousa Santiago 2019]. O critério de parada é definido pelo número de iterações ou um critério de convergência para interromper a execução do algoritmo.

Os passos do algoritmo PSO são os seguintes:

1. Inicialização das partículas com caminhos aleatórios (`initialize_particles`).
2. Avaliação das partículas usando a função de fitness (`fitness`).
3. Atualização das melhores posições pessoais (pBest) e global (gBest).
4. Cálculo das novas velocidades e aplicação das mesmas para atualizar as posições das partículas (`create_velocity` e `apply_velocity`).
5. Aplicação de mutação em uma taxa específica (`swap_mutation`).
6. Repetição dos passos 2 a 5 até atingir o critério de parada.

3. Resultados e Discussões

Ao executar o código, uma mensagem será impressa contendo o melhor indivíduo e sua configuração de cidades. Além disso, um arquivo de texto nomeado ”saida.txt”será gerado, contendo o melhor indivíduo ao longo das gerações. Como parâmetro adicional, é possível solicitar a geração de um gráfico que ilustra os melhores indivíduos encontrados. A Figura 1 apresenta o a evolução do melhor individuo ao longo das gerações.

Para analisar o comportamento desse algoritmo foram realizados testes para a otimização dos parâmetros com um código específico para essa finalidade. Dessa forma, os resultados apresentados nesta seção são baseados em 100 testes, ordenados do menor ao maior valor. Os parâmetros iniciais foram definidos da seguinte forma: $cidades = 12$, $particulas = 30$ e $iteracoes = 100$.

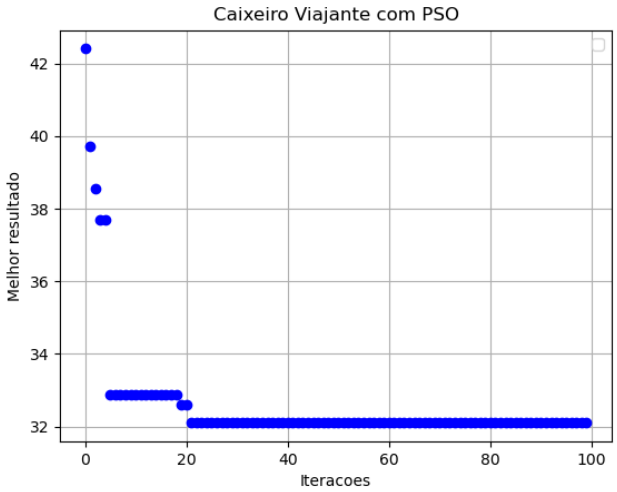

Figura 1. Gráfico da saída

A Figura 2 apresenta o gráfico dos melhores indivíduos em cada teste, evidenciando uma variação significativa nos resultados. A média dos melhores indivíduos é 34,71, com algumas exceções além desses limites. Por outro lado, a Figura 3 apresenta a geração em que o melhor indivíduo foi encontrado ao longo dos testes, revelando uma variação considerável, embora a maioria das descobertas tenha ocorrido por volta da iteração 32.

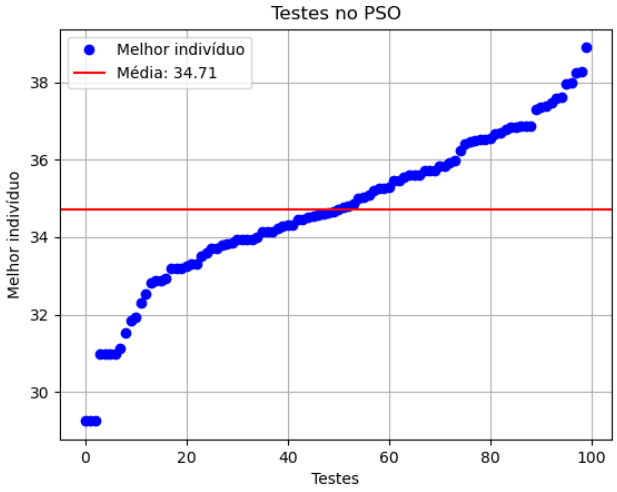

Figura 2. Gráfico dos melhores indivíduos em cada teste

Esses resultados indicam que, na maioria das execuções do algoritmo, o melhor indivíduo não é encontrado, sugerindo que essa abordagem pode não ser a mais eficaz para resolver o problema do caixeiro viajante. Como mostrado na Figura 2, o menor valor foi encontrado apenas três vezes nos 100 testes. Além disso, pelo menos 20% dos testes exigiram mais de 50 iterações para encontrar o melhor resultado, evidenciando a ineficácia deste método.

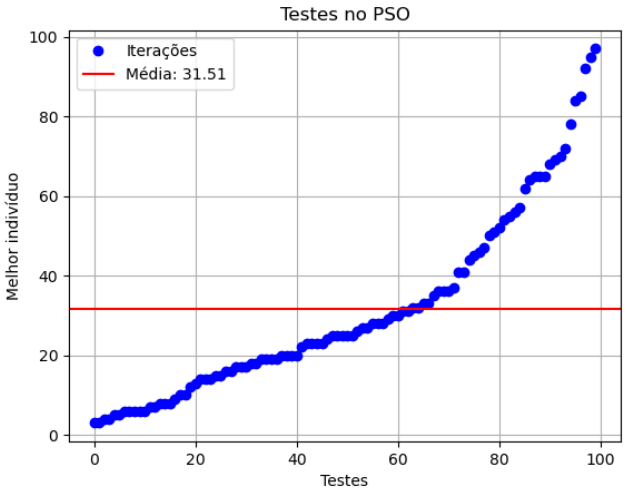

Figura 3. Gráfico do tempo para encontra o melhor indivíduo

Novos testes serão realizados, alterando o número de cidades e de partículas, para avaliar possíveis melhorias ou pioras ao modificar esses parâmetros. O primeiro teste analisará o impacto da quantidade de partículas. Observa-se, nas Figuras 4 e 5, que à medida que o número de partículas aumenta, os valores dos melhores indivíduos ficam menores, assim como o número de iterações necessárias para alcançá-los.

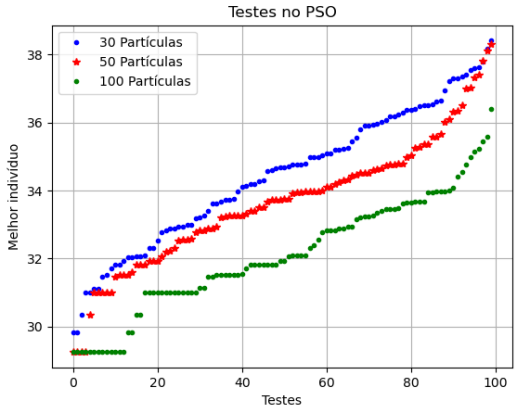

Figura 4. Melhor indivíduo

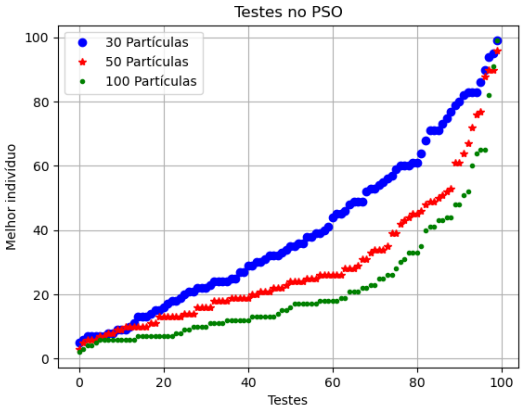

Figura 5. Iterações até encontrar

Para aumentar a eficiência do algoritmo e melhorar a probabilidade de encontrar o menor resultado com mais frequência, é recomendável utilizar um número maior de partículas. Ao aumentar o número de partículas, o algoritmo tem mais oportunidades de explorar o espaço de soluções de maneira mais abrangente, o que pode levar a uma convergência mais rápida e precisa para o menor resultado possível.

Por fim, foi realizada uma análise do impacto do aumento no número de cidades na aplicação. Para isso, foram realizados 100 testes, utilizando 12, 25 e 50 cidades. As figuras 6 e 7 mostram, respectivamente, o melhor resultado encontrado em cada teste e o número de iterações necessárias para encontrá-lo.

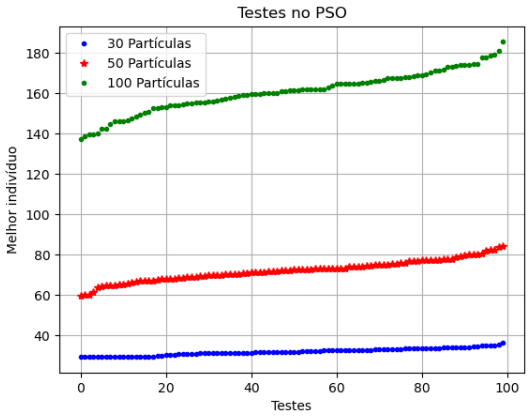

Figura 6. Melhor indivíduo

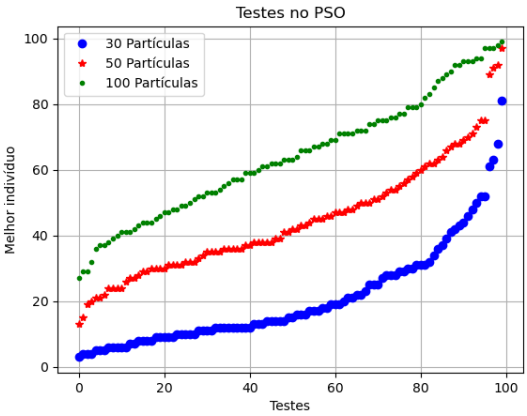

Figura 7. Iterações até encontrar

Como evidenciado nas figuras 7 e 6, o aumento no número de cidades resultou em uma queda significativa na eficiência e no desempenho do algoritmo. Muito provavelmente, essa queda está relacionada ao baixo número de iterações em relação ao aumento do número de cidades. Se o aumento no número de iterações fosse proporcional ao número de cidades, o desempenho do algoritmo não seria tão drasticamente afetado.

4. Conclusão

Este trabalho foi de suma importância para aprimorar o entendimento das técnicas de resolução de problemas NP-completos. O método PSO (Particle Swarm Optimization), embora apresente algumas inconsistências dependendo da forma como é utilizado, pode ser um método satisfatório devido à sua fácil implementação.

Através deste estudo, foi possível observar que o número de cidades, de iterações e de partículas devem ser balanceados adequadamente. Esse trabalho permitiu uma compreensão mais clara do impacto individual de cada um desses fatores. Por exemplo, verificou-se que um maior número de partículas tende a melhorar o desempenho do algoritmo, enquanto um aumento no número de cidades resulta em uma queda na eficiência. Além disso, o número de iterações é influenciado por esses dois critérios, sendo necessária uma adaptação específica para cada caso.

Essas descobertas reforçam a importância de um ajuste equilibrado entre o número de partículas, cidades e iterações para otimizar a performance do PSO. O trabalho evidencia que, com uma implementação cuidadosa e bem ajustada, o PSO pode ser uma ferramenta extremamente eficaz na resolução de problemas complexos, oferecendo uma abordagem acessível e adaptável para diferentes contextos.

Trabalhos futuros podem superar as limitações dos resultados obtidos por meio de uma análise de otimização de parâmetros do PSO mais aprofundada. Além disso, outros métodos baseados em Inteligência Artificial podem ser utilizados para a solução do Problema do Caixeiro Viajante, como Otimização das Colônias de Formigas, Simulated Annealing, Redes Neurais e métodos de Aprendizado por Reforço.

Referências

Avella P., Villacci, D. and Sforza, A. (2005). A steiner arborescence model for the feeder reconfiguration in electric distribution networks. *European Journal of Operational Research*, 164.

Cormen, T. (2012). *Algoritmos - Teoria e Prática*. LTC.

de Sousa Santiago, K. (2019). *Algoritmos Bioinspirados e suas Aplicações*. Novas Edições Acadêmicas.

Diestel, R. (2017). *Graph Theory*. Graduate Texts in Mathematics.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59.

Goldbarg, M. C. and Goldbarg, E. (2012). *Grafos: conceitos, algoritmos e aplicações*. Elsevier.

Russell, S. and Norvig, P. P. (2013). *Inteligência Artificial*. LTC.

Yücel, S. G., Pereira, R. H., Peixoto, P. S., and Camargo, C. Q. (2023). Impact of network centrality and income on slowing infection spread after outbreaks. *Applied Network Science*, 8.