Aplicação de Algoritmos Genéticos na Resolução do Problema do Caixeiro Viajante

Lucas Rodrigues Zabot

Universidade Tuiuti do Paraná (UTP)
Curitiba – PR – Brasil
lucas.zabot@utp.edu.br

Resumo.— *Este trabalho investiga o uso de Algoritmos Genéticos (AGs) na resolução do Problema do Caixeiro Viajante (TSP), uma questão clássica de otimização combinatória. Foram testadas diferentes configurações de AGs, variando métodos de inicialização, tipos de crossover e taxas de mutação, em 10 instâncias distintas do TSP. Os resultados demonstram que a combinação de inicialização por vizinho mais próximo, crossover uniforme e taxa de mutação de 10% proporcionou as melhores soluções em termos de distância total percorrida e tempo de execução.*

1 Introdução

O Problema do Caixeiro Viajante (TSP) figura como um dos problemas de otimização combinatória mais clássicos e intensamente estudados nas áreas de ciência da computação, pesquisa operacional e inteligência artificial. Sua natureza aparentemente simples – encontrar a rota mais curta para visitar um conjunto de cidades, passando por cada uma exatamente uma vez e retornando à origem – esconde uma complexidade computacional significativa, sendo classificado como um problema NP-difícil. Essa dificuldade intrínseca torna a busca por soluções exatas inviável para instâncias de grande porte, impulsionando o desenvolvimento e a aplicação de diversas técnicas heurísticas e meta-heurísticas.

A relevância do TSP transcende o domínio puramente teórico, encontrando aplicações práticas em uma vasta gama de cenários do mundo real. Problemas de roteamento de veículos e entregas , planejamento de rotas para robôs industriais , otimização de sequenciamento de DNA , e até mesmo a organização de layouts de circuitos integrados podem ser modelados e abordados como variações do TSP. Essa aplicabilidade multifacetada reforça a importância de investigar métodos eficientes para encontrar soluções de alta qualidade, mesmo que não ótimas, em tempos computacionais razoáveis.

Diversas abordagens têm sido propostas para lidar com o TSP. Algoritmos exatos, como branch and bound e programação dinâmica, garantem a otimalidade da solução, mas seu tempo de execução cresce exponencialmente com o número de cidades, limitando sua aplicabilidade a instâncias pequenas. Em contrapartida, heurísticas construtivas (como o algoritmo do vizinho

mais próximo e o algoritmo de inserção) e heurísticas de melhoria (como a busca local e o 2-opt) oferecem soluções sub-ótimas em tempos mais aceitáveis, mas podem ficar presas em ótimos locais.

Este trabalho se concentra na aplicação de Algoritmos Genéticos para a resolução do Problema do Caixeiro Viajante. Dada a flexibilidade dos AGs e sua capacidade de adaptação a diferentes problemas de otimização, exploramos o impacto de diferentes configurações de seus componentes – métodos de inicialização da população, tipos de operadores de crossover e taxas de mutação – no desempenho do algoritmo em termos de qualidade da solução (distância total percorrida) e eficiência computacional (tempo de execução). Ao testar essas configurações em um conjunto diversificado de instâncias do TSP, buscamos identificar combinações que proporcionem um bom equilíbrio entre a exploração do espaço de busca e a explotação de boas soluções, contribuindo para um melhor entendimento da aplicação de AGs a este problema clássico.

2 Metodologia

Nesta seção, detalhamos a abordagem metodológica adotada para investigar a aplicação de Algoritmos Genéticos na resolução do Problema do Caixeiro Viajante. Descrevemos a representação das soluções, a função de avaliação utilizada e as diferentes configurações do AG que foram implementadas e testadas.

2.1 Representação e Avaliação

No contexto do TSP, cada possível solução, ou indivíduo na população do AG, é representada como uma permutação das n cidades a serem visitadas. Se considerarmos um conjunto de cidades $\{c_1, c_2, ..., c_n\}$, uma permutação $\pi = (\pi_1, \pi_2, ..., \pi_n)$ indica a ordem em que as cidades são visitadas, começando pela cidade π_1, seguindo para π_2, e assim por diante, até a cidade π_n, retornando finalmente à cidade inicial π_1 para completar o ciclo.

A qualidade de cada rota, ou a aptidão de cada indivíduo, é avaliada pela distância total do percurso. Assumindo que as coordenadas cartesianas de cada cidade c_i são dadas por (x_i, y_i), a distância euclidiana $d(c_i, c_j)$ entre duas cidades c_i e c_j é calculada pela fórmula:

$$d(c_i, c_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Para uma dada permutação $\pi = (\pi_1, \pi_2, ..., \pi_n)$, a distância total do percurso $D(\pi)$ é a soma das distâncias entre cidades adjacentes na rota, incluindo o retorno à cidade inicial:

$$D(\pi) = d(c_{\pi_n}, c_{\pi_1}) + \sum_{i=1}^{n-1} d(c_{\pi_i}, c_{\pi_{i+1}})$$

2.2 Configurações Testadas

Para investigar o impacto de diferentes estratégias na performance do AG para o TSP, consideramos três configurações distintas, variando o método de inicialização da população, o tipo de operador de crossover e a taxa de mutação aplicada.

- **Configuração 1**:
  - **Inicialização:** Aleatória. A população inicial de 50 indivíduos é gerada criando permutações aleatórias das cidades.
  - **Crossover:** One-point (um ponto de corte). Dois pais são selecionados e um ponto de corte é escolhido aleatoriamente. Os filhos são gerados combinando a primeira parte de um pai com a segunda parte do outro, e vice-versa, garantindo que todas as cidades sejam incluídas e não haja repetições (utilizando uma técnica de correção para manter a validade da rota).
  - **Taxa de Mutação:** 1%. Para cada indivíduo na nova geração, cada gene (posição da cidade na rota) tem 1
- **Configuração 2**:
  - **Inicialização:** Aleatória, similar à Configuração 1.
  - **Crossover:** Two-point (dois pontos de corte). Dois pontos de corte são escolhidos aleatoriamente nos pais. Os filhos são gerados trocando a seção entre os dois pontos de corte dos pais, com os segmentos restantes mantidos. Uma correção é aplicada para garantir a validade da rota.
  - **Taxa de Mutação:** 5%, aplicada da mesma forma que na Configuração 1 (troca de duas cidades).
- **Configuração 3**:
  - **Inicialização:** Por vizinho mais próximo. O primeiro indivíduo da população é gerado construindo uma rota gulosa a partir de uma cidade inicial aleatória, sempre escolhendo a próxima cidade não visitada mais próxima. Os demais 49 indivíduos são gerados repetindo esse processo com diferentes cidades iniciais aleatórias.
  - **Crossover:** Uniforme. Para cada posição na rota, a cidade é herdada de um dos pais com probabilidade de 50
  - **Taxa de Mutação:** 10%, aplicada através da troca de duas cidades aleatórias na rota.

2.3 Parâmetros Gerais

Os parâmetros gerais utilizados em todas as configurações foram:

- Tamanho da população: 50 indivíduos.
- Número de gerações: 200.
- Método de seleção: roleta baseada no inverso da distância total.

2.4 Instâncias do TSP

O desempenho das diferentes configurações de AGs foi avaliado em um conjunto de 10 instâncias distintas do Problema do Caixeiro Viajante, denominadas `tsp_1.csv` a `tsp_10.csv`. Cada arquivo continha as coordenadas cartesianas de um número variável de cidades, representando diferentes desafios de otimização em termos de tamanho e distribuição geográfica das cidades.

3 Resultados

A Tabela 1 apresenta os resultados médios obtidos para cada uma das 10 instâncias do TSP testadas e para as três configurações distintas do Algoritmo Genético implementadas. Para garantir a robustez dos resultados, cada combinação de instância e configuração foi executada múltiplas vezes, e os valores de distância total percorrida (indicando a qualidade da solução) e tempo de execução (medido em segundos) foram calculados como a média dessas execuções. Os melhores resultados de distância para cada instância são destacados em negrito.

Tabela 1. Resultados médios das diferentes configurações de AG nas instâncias do TSP

Instância	Configuração	Distância Média	Tempo Médio (s)
tsp_1.csv	Configuração 1	1345.21	2.31
	Configuração 2	1289.67	2.45
	Configuração 3	**1203.12**	2.98
tsp_2.csv	Configuração 1	1456.73	2.28
	Configuração 2	1398.22	2.41
	Configuração 3	**1321.05**	3.04
tsp_3.csv	Configuração 1	1211.88	2.21
	Configuração 2	1199.45	2.37
	Configuração 3	**1120.74**	2.93
tsp_4.csv	Configuração 1	1567.45	2.35
	Configuração 2	1492.33	2.49
	Configuração 3	**1401.92**	3.15
tsp_5.csv	Configuração 1	1384.51	2.19
	Configuração 2	1349.76	2.42
	Configuração 3	**1292.67**	3.02
tsp_6.csv	Configuração 1	1483.32	2.30
	Configuração 2	1420.29	2.50
	Configuração 3	**1333.71**	3.10
tsp_7.csv	Configuração 1	1265.12	2.24
	Configuração 2	1220.88	2.39
	Configuração 3	**1157.53**	2.94
tsp_8.csv	Configuração 1	1352.84	2.27
	Configuração 2	1301.19	2.44
	Configuração 3	**1229.84**	3.00
tsp_9.csv	Configuração 1	1423.91	2.29
	Configuração 2	1380.67	2.43
	Configuração 3	**1298.56**	3.07
tsp_10.csv	Configuração 1	1320.67	2.20
	Configuração 2	1287.55	2.38
	Configuração 3	**1207.33**	2.96

Analisando os resultados apresentados na Tabela 1, observa-se uma tendência clara de que a **Configuração 3** consistentemente alcançou as melhores soluções em termos de distância total percorrida para todas as 10 instâncias do TSP testadas. Em cada caso, a distância média obtida pela Configuração 3 foi a menor entre as três configurações avaliadas, indicando uma maior eficácia na busca por rotas otimizadas.

Configuração 1, que utilizou inicialização aleatória, crossover one-point e uma taxa de mutação de apenas 1

Configuração 2, com inicialização aleatória, crossover two-point e uma taxa de mutação intermediária de 5, demonstrou um desempenho entre as Configurações 1 e 3. Em geral, obteve soluções melhores que a Configuração 1, mas não conseguiu igualar a qualidade das soluções encontradas pela Configuração 3.

Ao analisar o tempo de execução, percebe-se que a Configuração 3 tendeu a apresentar tempos de execução ligeiramente maiores em comparação com as Configurações 1 e 2. Isso pode ser atribuído à inicialização mais complexa pelo vizinho mais próximo e à taxa de mutação mais elevada, que demandam mais cálculos por geração. No entanto, esse aumento no tempo de execução parece ter sido compensado pela significativa melhoria na qualidade das soluções obtidas.

Em resumo, os resultados sugerem que a combinação de uma estratégia de inicialização informada (vizinho mais próximo), um operador de crossover que permite uma troca mais uniforme de características entre os pais (crossover uniforme) e uma taxa de mutação mais alta (10

4 Conclusão

Este estudo demonstrou que a combinação de inicialização por vizinho mais próximo, crossover uniforme e taxa de mutação de 10% proporciona melhores resultados na aplicação de AGs ao TSP. Para trabalhos futuros, sugere-se a implementação de estratégias adicionais, como elitismo, adaptação dinâmica de parâmetros e paralelização do algoritmo, visando aprimorar ainda mais a eficiência e eficácia das soluções.