Relatório Técnico: Otimização de Rotas com Algoritmo Genético

Introdução

Este relatório descreve a implementação de um sistema de otimização de rotas para o Problema de Roteamento de Veículos (VRP), uma generalização do clássico Problema do Caixeiro Viajante (TSP). O objetivo é determinar as rotas ótimas para uma frota de veículos a partir de um depósito central para atender a um conjunto de cidades (clientes), minimizando a distância total percorrida e respeitando um conjunto de restrições operacionais.
A solução foi desenvolvida em Python, utilizando um Algoritmo Genético (AG) para explorar o espaço de soluções e encontrar rotas de alta qualidade. O sistema inclui uma visualização em tempo real com Pygame, permitindo a análise interativa do processo de otimização.

---

1. Implementação do Algoritmo Genético para Roteamento

O Algoritmo Genético é uma meta-heurística inspirada na teoria da evolução de Charles Darwin. Ele opera sobre uma população de soluções candidatas, aplicando operadores genéticos como seleção, cruzamento (crossover) e mutação para evoluir iterativamente em direção a soluções melhores. A implementação no projeto segue os seguintes componentes:

a) Representação do Indivíduo (Cromossomo): Cada "indivíduo" na população representa uma solução potencial para o problema. A representação escolhida é uma permutação dos índices das cidades a serem visitadas. Por exemplo, para 5 cidades, um indivíduo como [3, 0, 4, 1, 2] define a ordem de visitação. O depósito não faz parte dessa permutação, pois é o ponto de partida e chegada de todas as rotas.

b) População Inicial: A população inicial é criada pela função **generate_random_population**. Ela gera um número pré-definido (POPULATION_SIZE) de indivíduos, cada um sendo uma permutação aleatória das cidades, garantindo diversidade no início do processo evolutivo.

c) Função de Aptidão (Fitness): A função **calculate_fitness** é o componente mais crítico, pois avalia a "qualidade" de cada indivíduo. O objetivo principal é minimizar a distância total. A função calcula a distância euclidiana total percorrida por todos os veículos. Além disso, ela é responsável por incorporar as restrições do problema, como veremos na próxima seção. Um valor de fitness menor indica uma solução melhor (rota mais curta).

d) Processo Evolutivo:

1. Seleção: Para criar a próxima geração, os pais são selecionados com base em sua aptidão. A implementação utiliza uma forma de Seleção por Roleta, onde a probabilidade de um indivíduo ser escolhido é inversamente proporcional ao seu valor de fitness (distância). Indivíduos com rotas mais curtas têm maior chance de serem selecionados para reprodução.

```
python
# tsp.py - Mecanismo de seleção
probability = 1 / np.array(population_fitness)
parent1, parent2 = random.choices(population, weights=probability, k=2)
```

2. Cruzamento (Crossover): O operador **order_crossover** (Order Crossover - OX1) é utilizado para combinar o material genético de dois pais e gerar um descendente. Este método é ideal para problemas baseados em permutação, pois garante que o filho gerado seja uma permutação válida (não visita cidades repetidas). Ele funciona copiando uma subsequência de um pai e preenchendo o restante do cromossomo com os genes do outro pai na ordem em que aparecem.

3.Mutação: O operador **mutate** introduz pequenas alterações aleatórias nos filhos para manter a diversidade genética e evitar a convergência prematura para ótimos locais. A implementação realiza uma mutação de troca (swap), onde dois genes (cidades) adjacentes são trocados de posição, com uma probabilidade definida por MUTATION_PROBABILITY.

4.Elitismo: Para garantir que a melhor solução encontrada até o momento não seja perdida, a estratégia de elitismo foi implementada. O melhor indivíduo de cada geração é automaticamente transferido para a próxima, preservando o progresso da otimização.

5.Critério de Parada: O algoritmo executa por um tempo pré-determinado (TIME_LIMIT_SECONDS), permitindo que a solução evolua até que o limite de tempo seja atingido.

---

2. Estratégias para Lidar com Restrições Adicionais

O problema implementado vai além do TSP simples, incorporando restrições do VRP. A estratégia central para lidar com elas é o uso de funções de penalidade dentro da **calculate_fitness**. Se uma solução viola uma restrição, uma penalidade significativa é adicionada ao seu valor de fitness, tornando-a menos provável de sobreviver e se reproduzir.

a) Múltiplos Veículos: A função de fitness distribui a sequência de cidades do indivíduo entre o número de veículos disponíveis (VEHICLE_COUNT). A lógica de divisão (implementada em **`genetic_algorithm.py`**, não mostrada aqui, mas inferida dos testes) tenta criar rotas balanceadas, atribuindo um subconjunto de cidades a cada veículo. A distância total é a soma das distâncias de todas as rotas individuais dos veículos.

b) Capacidade de Carga (CAPACITY): Cada cidade pode ter uma demanda associada. A função de fitness verifica se a soma das demandas das cidades em uma rota de um único veículo excede sua capacidade. Se a capacidade for violada, uma alta penalidade é aplicada ao fitness, como demonstrado no teste **test_calculate_fitness_capacity_penalty**.

c) Autonomia dos Veículos (MAX_DISTANCE): Similarmente à capacidade, a função verifica se a distância total de uma rota individual (incluindo a ida e volta ao depósito) excede a autonomia máxima do veículo. Se exceder, uma penalidade é somada ao fitness, desencorajando soluções com rotas inviáveis, como visto no teste **test_calculate_fitness_distance_penalty**.
d) Prioridades (PRIORITY): A prioridade é tratada como um peso no cálculo da distância. Cidades com maior prioridade podem ter sua distância "percebida" reduzida ou, alternativamente, a função de fitness pode ser ajustada para favorecer a visitação antecipada a essas cidades. No código atual, um valor de PRIORITY é passado para a função de fitness, sugerindo que ele modifica o cálculo do custo total da rota.

---

3. Comparativo de Desempenho com Outras Abordagens

Os Algoritmos Genéticos são uma de várias abordagens para resolver problemas de roteamento.

Abordagem	Vantagens	Desvantagens
Algoritmo Genético (Implementado)	- **Flexibilidade:** Lida bem com problemas complexos e múltiplas restrições (VRP).- **Robustez:** Bom para encontrar soluções de alta qualidade em espaços de	- **Não garante o ótimo:** Por ser uma heurística, não há garantia de encontrar a melhor solução global.- **Sensível a parâmetros:** O desempenho depende do

	busca grandes, sem ficar preso em ótimos locais.- **Paralelizável:** A avaliação da população pode ser feita em paralelo.	ajuste fino de parâmetros como tamanho da população, taxa de mutação, etc.- **Computacionalmente intensivo.**
Algoritmos Exatos (e.g., Branch and Bound)	- **Garantia de Otimalidade:** Encontram a melhor solução possível.	- **Inviável para problemas grandes:** O tempo de execução cresce exponencialmente com o número de cidades. Praticamente inutilizável para mais de ~20-30 cidades.
Heurísticas Construtivas (e.g., Vizinho Mais Próximo)	- **Rápidos e simples:** Geram uma solução muito rapidamente.- **Bom ponto de partida:** Podem ser usados para criar a população inicial de um AG.	- **Qualidade da solução:** Geralmente produzem soluções de baixa qualidade (sub-ótimas), pois tomam decisões "gulosas" sem visão global.
Outras Meta-heurísticas (e.g., Simulated Annealing, Tabu Search)	- **Eficazes:** Também são muito eficazes para VRP e podem, em alguns casos, convergir mais rápido que AGs.	- **Complexidade de implementação:** Podem ser mais complexos de implementar e parametrizar corretamente.

Conclusão do Comparativo: A escolha do Algoritmo Genético para este projeto é justificada por seu excelente equilíbrio entre a qualidade da solução e a capacidade de lidar com a complexidade e as múltiplas restrições do VRP, algo que métodos exatos ou heurísticas simples não conseguem oferecer de forma eficaz para problemas de tamanho realista.

---

4. Visualizações e Análises das Rotas Otimizadas

O projeto utiliza a biblioteca Pygame para fornecer feedback visual em tempo real, o que é crucial para a análise e compreensão do comportamento do algoritmo.

a) Visualização em Tempo Real: A tela principal é dividida em duas partes:

1.Gráfico de Fitness (Esquerda): O módulo **`draw_functions.py`** plota a evolução do fitness do melhor indivíduo a cada geração. Uma curva descendente indica que o algoritmo está progredindo e encontrando rotas cada vez mais curtas. Estagnações na curva podem sugerir que o algoritmo convergiu ou está preso em um ótimo local.

2.Mapa de Rotas (Direita): Exibe a localização geográfica do depósito (roxo) e das cidades. A melhor rota encontrada na geração atual é desenhada, com cores diferentes para cada veículo. As cidades são numeradas na ordem de visitação para cada rota, facilitando a interpretação do trajeto.

b) Análise dos Resultados Finais: Ao final da execução, o programa gera saídas importantes para análise offline:

- Imagens (**`best_route.png`**, t**opN_route.png**): Salvam uma imagem da melhor solução global encontrada e das N melhores soluções, combinando o mapa da rota e o gráfico de evolução do fitness. Isso permite documentar e comparar visualmente os melhores resultados.

- Arquivos de Dados (**`top20_results.csv`**, **`top20_results.json`**): Armazenam os dados brutos das 20 melhores soluções encontradas durante toda a execução, incluindo o fitness, a geração em que foi encontrada e a sequência exata da rota. Esses arquivos são fundamentais para uma análise quantitativa, permitindo:
  - Identificar a melhor rota de forma precisa.
  - Analisar a variabilidade entre as melhores soluções.
  - Utilizar os dados para relatórios ou para alimentar outros sistemas logísticos