IF Sul de Minas - Campus Poços de Caldas
Trabalho Prático 2 de Projetos e Análise de Algoritmos
Resolução do Problema do Caixeiro Viajante

Aluno: Iago Ananias Silva
Professor: Douglas Castilho

Introdução:

O Problema do Caixeiro Viajante (PCV), também conhecido como Travelling Salesman Problem, é um desafio clássico na área de otimização combinatória. Este problema busca determinar a rota mais curta que um vendedor deve seguir para visitar uma série de cidades, passando por cada uma exatamente uma vez e retornando à cidade de origem. Com origens no contexto prático de otimizar entregas, o PCV é um problema NP-Completo, o que significa que não existe um algoritmo eficiente para resolvê-lo em tempo polinomial.

No âmbito deste trabalho, abordaremos duas abordagens distintas para resolver o PCV: uma utilizando um método exato baseado em tentativa e erro (Algoritmo Ótimo) e outra empregando uma heurística (Algoritmo Genético). A primeira estratégia explora a busca exaustiva por soluções, analisando todas as possíveis combinações de percursos, enquanto a segunda se baseia em princípios inspirados na evolução genética para encontrar soluções aproximadas.

Algoritmo Ótimo (Tentativa e Erro):
Na implementação do Algoritmo Ótimo, adotamos a estratégia de força bruta, gerando todas as permutações possíveis dos vértices e avaliando o custo de cada caminho. Embora eficaz para conjuntos de dados menores, esse método pode se tornar impraticável para grafos de grande porte devido à sua complexidade fatorial.

Algoritmo Heurístico (Algoritmo Genético):
A abordagem heurística utiliza o Algoritmo Genético para buscar soluções aproximadas. Esse método simula processos inspirados na evolução natural, como seleção, crossover e mutação, visando encontrar soluções de boa qualidade em um tempo computacional viável. Apesar de não garantir a solução ótima, é uma alternativa eficiente para problemas complexos como o PCV.

Tentativa e Erro :
A classe CaixeiroViajanteForcaBruta implementa a resolução do Problema do Caixeiro Viajante (PCV) utilizando o método de força bruta. O PCV consiste em encontrar o menor caminho que visita todos os vértices de um grafo exatamente uma vez e retorna ao vértice de origem.

Membros da Classe:

melhorCusto: Armazena o custo do melhor caminho encontrado durante a busca.
melhorCaminho: Armazena a lista de vértices que representa o melhor caminho encontrado.
grafo: Representa o grafo para o qual o problema do caixeiro viajante está sendo resolvido, armazenado como uma matriz de adjacência.
Métodos Importantes:

Construtor:

CaixeiroViajanteForcaBruta(Grafo grafo): Inicializa a instância com o grafo fornecido, define o melhor custo inicial como infinito e o melhor caminho inicial como uma lista vazia.

Método Principal:

resolver(): Inicia o processo de resolução do PCV. Gera todas as permutações dos vértices e calcula o custo de cada caminho, mantendo o registro do melhor caminho encontrado.

Métodos Auxiliares:

permutacao(ArrayList<Vertice> vertices, int indice): Implementa a geração de permutações recursivamente, calculando e comparando os custos dos caminhos.
swap(ArrayList<Vertice> vertices, int i, int j): Troca dois elementos na lista de vértices, auxiliando na geração de permutações.
calcularCusto(ArrayList<Vertice> caminho): Calcula o custo total de um caminho percorrendo todos os vértices.

Funcionamento:

O método resolver() é chamado para iniciar o processo de busca exaustiva de permutações.
Para cada permutação gerada, o custo do caminho é calculado usando o método calcularCusto.
Se o custo atual for menor que o melhor custo conhecido, o melhor caminho e custo são atualizados.
A busca continua até que todas as permutações sejam geradas e avaliadas.

Observações:

O algoritmo de força bruta gera todas as permutações possíveis dos vértices, resultando em uma complexidade fatorial.

Pode ser impraticável para grandes conjuntos de dados devido ao seu alto custo computacional.

Algoritmo Genético:

A classe AlgoritmoGeneticoTSP implementa um Algoritmo Genético para resolver o Problema do Caixeiro Viajante (TSP). Este algoritmo é uma heurística de otimização que simula a evolução genética para encontrar soluções aproximadas para o TSP.

Membros da Classe:

- **matrizAdjacencia**: Representa a matriz de adjacência do grafo associado ao problema do Caixeiro Viajante.
- **tamanhoPopulacao**: Define o tamanho da população no algoritmo genético.
- **taxaCrossover**: Define a taxa de crossover, que influencia a probabilidade de ocorrer a recombinação genética.
- **taxaMutacao**: Define a taxa de mutação, que influencia a probabilidade de ocorrerem mutações genéticas.
- **numeroGeracoes**: Representa o número de gerações (iterações) para as quais o algoritmo genético será executado.

Métodos Importantes:

1. **Construtor:**
    a. **AlgoritmoGeneticoTSP(Grafo grafo, int tamanhoPopulacao, double taxaCrossover, double taxaMutacao, int numeroGeracoes):** Inicializa a instância com os parâmetros necessários para a execução do algoritmo genético.
2. **Método Principal:**
    a. **resolverTSP():** Inicia o processo de resolução do TSP utilizando o algoritmo genético. Gera uma população inicial, evolui a população ao longo de várias gerações e retorna o melhor caminho encontrado.
3. **Métodos Auxiliares:**
    a. **gerarPopulacaoInicial():** Gera uma população inicial de caminhos aleatórios.
    b. **selecionarEvoluir(List<List<Vertice>> populacao):** Realiza a seleção e evolução da população através de crossover e mutação.
    c. **selecionarPai(List<List<Vertice>> populacao):** Realiza um torneio entre indivíduos para selecionar pais.
    d. **crossover(List<Vertice> pai1, List<Vertice> pai2):** Realiza o crossover (recombinação genética) entre dois pais para gerar um filho.
    e. **mutacao(List<Vertice> caminho):** Aplica uma mutação genética a um caminho, trocando a posição de dois genes.

    f. **obterMelhorCaminho(List<List<Vertice>> populacao):** Retorna o melhor caminho dentro de uma população.
4. **Método de Avaliação:**
    a. **calcularCusto(List<Vertice> caminho):** Calcula o custo total de um caminho percorrendo todos os vértices.

Funcionamento:

- A população inicial é gerada aleatoriamente.
- A cada geração, os indivíduos são selecionados para crossover e mutação, resultando em uma nova população.
- O processo é repetido por um número definido de gerações.
- Ao final, o melhor caminho encontrado é retornado como a solução aproximada para o TSP.

Observações:

- A eficácia do algoritmo genético depende da escolha adequada dos parâmetros, como taxa de crossover, taxa de mutação e tamanho da população.
- Este método é uma heurística e não garante a solução ótima, mas é eficaz para problemas complexos como o TSP.

Conclusão:

Em conclusão, abordamos o desafiador Problema do Caixeiro Viajante, apresentando duas estratégias distintas para a sua resolução. O Algoritmo Ótimo demonstra a aplicação do método de tentativa e erro, explorando todas as possíveis soluções para encontrar a rota mais curta. Por outro lado, o Algoritmo Genético, uma heurística inspirada na evolução natural, oferece uma abordagem mais eficiente para conjuntos de dados maiores, proporcionando soluções aproximadas em um tempo computacional razoável.

Ambas as abordagens têm seus méritos e limitações, destacando a importância de escolher a estratégia mais adequada para as características específicas do problema em questão. A resolução do PCV exemplifica a variedade de métodos disponíveis na busca por soluções otimizadas em problemas complexos e relevantes em diversas áreas.