Universidade Federal de Lavras - UFLA
Departamento de Ciências da Computação
GCC218 - Algoritmo em Grafos

Gabriel Venancio Avelar
Julia Aparecida de Faria Morais

Relatório – Trabalho Prático

Lavras
2025

Sumário

1 Introdução

A empresa de transportes urbanos busca otimizar as rotas de suas linhas de ônibus para reduzir custos e melhorar a eficiência operacional. O objetivo é garantir que os motoristas passem pelos pontos de parada da forma mais eficiente possível, considerando que a principal prioridade é minimizar a distância máxima entre dois pontos quaisquer do percurso, ao invés de simplesmente minimizar o custo total do trajeto. Este problema é uma variação do Problema do Caixeiro Viajante (TSP), com uma restrição adicional: minimizar a maior distância entre dois pontos consecutivos da rota.

Neste trabalho, detalhamos uma solução que combina busca binária e o algoritmo Lin-Kernighan para encontrar rotas eficientes. A implementação foi desenvolvida em C++ e utiliza técnicas de grafos e otimização combinatória, focando em escalabilidade e adaptabilidade a diferentes tipos de cálculo de distância (Euclidiana ou Geográfica).

2 Formação do Problema

2.1 Descrição formal

Entrada:

- n pontos de parada com coordenadas geográficas.
- Tipo de cálculo de distância: Euclidiana (EUC_2D) ou Geográfica (GEO, baseada na fórmula de Haversine).

Saída:

- Na saída padrão o valor da minimização da maior aresta entre dois pontos.
- Arquivo de saída no (formato .txt) com sequência de pontos que forma um ciclo (rota fechada).

Objetivo:

- Minimizar a maior distância entre dois pontos consecutivos na rota.

2.2 Complexidade

O problema é NP-difícil, herdando a complexidade do TSP tradicional. A restrição adicional exige estratégias heurísticas para viabilizar soluções em tempo prático.

.

3 Descrição da Solução

A solução é implementada em um único arquivo (main.cpp), organizado em módulos funcionais:

3.1 main.cpp

O arquivo main.cpp implementa a solução proposta para o problema de otimização de rotas. Ele é responsável por coordenar a leitura dos dados de entrada, a execução dos algoritmos de otimização e a geração da solução final. A seguir, descrevemos as principais funcionalidades e interações presentes neste arquivo.

3.1.1 Funcionamento do Algoritmo

1. Leitura de Dados

Funções:

- leArquivoEntrada(string arquivoEntrada):
    - **Objetivo**: Ler arquivos no formato TSPLIB, extraindo coordenadas e definindo o tipo de cálculo de distância (EUC_2D ou GEO).
    - **Funcionamento**:
        - Identifica seções como DIMENSION e NODE_COORD_SECTION para carregar pontos geográficos.
        - Chama calculaDistanciaEuc() ou calculaDistanciaGeo() c onforme especificado.
- leEntradaPadrao():
    - **Objetivo**: Permitir entrada manual de dados via terminal para testes rápidos.
    - **Funcionamento**:
        - Solicita dimensão, tipo de distância e coordenadas dos pontos.
        - Constrói a matriz de distâncias após a entrada.

---

2. Cálculo de Distâncias

Funções:

- calculaDistanciaEuc():
    - **Objetivo**: Calcular distâncias Euclidianas entre pontos, arredondadas para inteiro.
    - **Fórmula**: dist = $\sqrt{[(x_2 - x_1)^2 + (y_2 - y_1)^2]}$

- calculaDistanciaGeo():
    - **Objetivo**: Calcular distâncias geográficas usando a fórmula de Haversine (em km).
    - **Fórmula**: $a = \sin^2(\Delta lat/2) + \cos(lat_1) * \cos(lat_2) * \sin^2(\Delta lon/2)$

      $dist = 2 * 6371 * atan2(\sqrt{a}, \sqrt{(1-a)})$

---

3. Inicialização da Rota

Função: inicializaRota()

- **Objetivo**: Gerar uma rota inicial sequencial.
- **Funcionamento**:
    - Cria um ciclo onde cada ponto conecta ao próximo, e o último retorna ao início.
    - Serve como ponto de partida para o algoritmo de otimização.

---

4. Algoritmo de Busca Binária

Função: minimizaMaiorAresta(int numeroDeInteracoes)

- **Objetivo**: Encontrar a menor aresta máxima viável usando busca binária.
- **Passos**:
    1. **Limites**:
        - **Inferior**: Calculado por encontraLimiteInferior() (maior segundo menor custo de cada nó).
        - **Superior**: Calculado por encontraLimiteSuperior() (heurística gulosa).
    2. **Busca Binária**:
        - Define um valor intermediário (mid).
        - Modifica a matriz de distâncias: arestas > mid são zeradas.
        - Executa MelhoraRota() para tentar construir uma rota válida.
    3. **Atualização de Limites**:
        - Se uma rota válida é encontrada (custo total = 0), reduz o limite superior.
        - Caso contrário, aumenta o limite inferior.

5. Algoritmo Lin-Kernighan

Funções:

- linKernighan(int inicio):
  - **Objetivo**: Refinar a rota através de trocas locais de arestas.
  - **Mecanismo**:
    - Identifica pares de arestas que podem ser trocados para reduzir a distância total.
    - Mantém registros de arestas removidas (arestasRemovidas) e adicionadas (arestasAdicionadas).
    - Inverte segmentos da rota quando há ganho de custo.
- MelhoraRota(int numeroDeInteracoes):
  - **Objetivo**: Aplicar o Lin-Kernighan iterativamente em todos os pontos da rota.
  - **Funcionamento**:
    - Executa linKernighan(j) para cada ponto j.
    - Interrompe se não houver melhoria após uma iteração.

6. Validação e Saída

Funções:

- validaRota():
  - **Objetivo**: Verificar se a rota é um ciclo Hamiltoniano válido.
  - **Método**:
    - Rastreia visitas aos nós para garantir que cada ponto é visitado uma vez.
    - Confirma que a rota termina no ponto inicial.
- salvaSolucaoEmArquivo(string arquivoSaida):
  - **Objetivo**: Salvar a sequência ótima de pontos em um arquivo .txt.
  - **Formato**: Lista de índices separados por espaços (ex.: 0 1 2 3 0).

3.1.2 Como as Funções Resolvem o Problema

- Busca Binária: Reduz o espaço de busca para a maior aresta, permitindo foco em soluções viáveis.
- Lin-Kernighan: Refina rotas localmente, garantindo que a solução seja melhorada iterativamente.
- Validação Contínua: Assegura que todas as rotas geradas são ciclos válidos, mantendo a integridade do resultado.

- Flexibilidade: Suporte a distâncias Euclidianas e geográficas torna a solução adaptável a cenários urbanos reais.

Este conjunto de funções trabalha de forma coordenada para equilibrar eficiência computacional e qualidade da solução, resolvendo o problema de otimização dentro de limites práticos.

4 Resultados Obtidos

Instancia (arq)	Instancia TSP	Solução Inicial (SI)	Solução Final (SF)	Desvio (%) SI →SF	Desvio (%) SF →SO	Tempo (s)	Configuração do Computador
01.ins	535	19540	4238	78.3112	N/A	8.45966	Processador: Intel Core i5-9300H (2.40 GHz) RAM: 8 GB (7.84 GB utilizável) Sistema: 64 bits, x64 Windows 11 Home
02.ins	1291	3682	1294	64.8561	N/A	23.8677	
03.ins	1655	3898	1525	60.8774	N/A	12.2157	
04.ins	2103	5204	1133	78.2283	N/A	114.575	
05.ins	1400	2882	626	78.279	N/A	56.4591	
06.ins	1577	1879	465	75.2528	N/A	24.434	
07.ins	1379	2523	172	93.1827	N/A	54.4035	
08.ins	1173	3264	292	91.0539	N/A	16.3196	
09.ins	783	614	52	91.5309	N/A	9.07375	
10.ins	1817	279	234	16.129	N/A	10.9099	

5 Conclusão

Este trabalho abordou o problema de otimização de rotas urbanas com foco na minimização da maior distância entre pontos consecutivos, uma variação do *Traveling Salesman Problem* (TSP) com restrição de aresta crítica. A solução proposta integrou técnicas de otimização combinatória e meta-heurísticas.

6 Bibliografia

LA RUSIC, John. **Solving the Traveling Salesman Problem Using the Lin-Kernighan Heuristic**. 2014. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade de New Bruns wick,
Fredericton, 2014. Disponível em: <https://www.cs.unb.ca/tech-reports/honours-theses/John.LaRusic-4997.pdf>.

SILVA, João M. **Aplicação do Problema do Caixeiro Viajante numa Empresa de Distribuição: A Heurística de Lin-Kernighan**. 2020. Dissertação (Mestrado em Engenharia Informática) – Universidade de Coimbra, Coimbra, 2020. Disponível em: <https://estudogeral.uc.pt/bitstream/10316/84465/1/Aplicação%20do%20Problema%20do%20Caixeiro%20Viajante%20numa%20empresa%20de%20distribuição%20-%20A%20heurística%20de%20Lin-Kernighan.pdf>.