UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE CRATEÚS
CURSO DE CIÊNCIAS DA COMPUTAÇÃO
DISCIPLINA: Algoritmos em Grafos.
DOCENTE: Rafael Martins Barros.

DISCENTES/MATRÍCULAS:
Carlos Eduardo De Almeida Coutinho - 554710
Luis Eduardo Martins Barbosa - 507860
Tarcísio Soares Oliveira Alcanfor - 541983
Victor Lopes Mendes - 509736

RELATÓRIO DE ALGORITMOS EM GRAFOS

CRATEÚS
2025

Sumário

CRATEÚS
2025

1. Introdução

Este relatório detalha a implementação desenvolvida para o Trabalho Final da disciplina de Algoritmos em Grafos. A atividade, que corresponde à nota da Terceira Unidade, visa a implementação de uma solução aproximada para o Problema do Caixeiro Viajante (TSP), mais específicamente o problema do Caixeiro Viajante Métrico (Δ-TSP), pois os pesos das arestas satisfazem a desigualdade triangular.

Foi utilizado a linguagem Python para a implementação, junto ao ambiente Google Colab. Conforme as regras do projeto, não foram utilizadas bibliotecas externas para as etapas de representação de grafos, cálculo da árvore geradora mínima e determinação de ciclos eulerianos. No entanto, para a etapa de emparelhamento perfeito de custo mínimo — um dos passos cruciais do algoritmo — foi utilizada a biblioteca networkx. A plotagem dos grafos para visualização dos resultados também foi realizada com o auxílio desta biblioteca.

O desenvolvimento seguiu rigorosamente as etapas do Algoritmo de Christofides, desde a criação de uma árvore geradora mínima até a conversão de um ciclo euleriano em um ciclo hamiltoniano por meio de atalhos. Este documento apresentará as decisões de implementação, os testes realizados e uma análise dos resultados obtidos.

2. Objetivos

2.1 Objetivo Geral

O objetivo principal desse trabalho é a solução aproximada do Problema do Caixeiro Viajante (TSP) em grafos completos com pesos que satisfazem a desigualdade triangular, utilizando o algoritmo de Christofides.

2.2 Objetivos Específicos

Para alcançar o objetivo geral, os seguintes objetivos foram definidos:

- Implementar as seis etapas fundamentais do Algoritmo de Christofides:
    - **Construção de uma árvore geradora mínima**
    - **Identificação de vértices de grau ímpar**
    - **Construção de um emparelhamento perfeito de custo mínimo**
    - **União da árvore geradora mínima com o emparelhamento**
    - **Determinação de um ciclo euleriano**
    - **Atalho do ciclo euleriano para obter um ciclo hamiltoniano**
- O programa desenvolvido deverá processar grafos completos com pesos satisfazendo a desigualdade triangular, fornecidos via arquivo de texto, e exibir como saída o ciclo hamiltoniano encontrado e seu custo total. Além da implementação, o trabalho visa demonstrar os conceitos de algoritmos em grafos e as decisões de projeto envolvidas no desenvolvimento da solução.
- A implementação dessas etapas pode ser feita em qualquer linguagem de programação, sem o uso de bibliotecas prontas para a representação e/ou manipulação de grafos, cálculo de árvore geradora mínima ou determinação de ciclos eulerianos e utilizando bibliotecas prontas,a escolha, para encontrar o emparelhamento mínimo.

3.Metodologia:

Essa seção detalha ferramentas utilizadas, a arquitetura do software desenvolvido e os métodos empregados para implementar a solução do Problema do Caixeiro Viajante utilizando o Algoritmo de Christofides.

3.1 Ferramentas e Ambiente de Desenvolvimento:

O projeto foi integralmente desenvolvido na linguagem de programação **Python**, utilizando o ambiente de desenvolvimento colaborativo **Google Colaboratory (Colab)**. A escolha se deu pela facilidade de configuração, acesso a bibliotecas e capacidade de compartilhar o ambiente de execução.

Foram utilizadas as seguintes bibliotecas externas, em conformidade com as regras do trabalho:

- **NetworkX:** empregada exclusivamente para as tarefas permitidas:
  - Cálculo do emparelhamento perfeito de custo mínimo (`nx.min_weight_matching`), conforme incentivado pelo professor.
  - Criação de estruturas de grafo para a visualização dos resultados.
- **Matplotlib.pyplot:** utilizada para a plotagem e exibição gráfica do problema e de sua solução final.
- **Heapq:** Biblioteca padrão do Python, utilizada para implementar a fila de prioridade (heap de mínimo) necessária para a otimização do Algoritmo de Prim.

3.2 Estrutura de código:

O código foi estruturado conforme pedido no arquivo disponibilizado no SIGAA. Contendo funções para leitura do arquivo txt e conversão da matriz de antecedência, construção da árvore geradora mínima, identificação de vértices de grau ímpar, construção de um emparelhamento mínimo, determinação do ciclo euleriano, atalho do ciclo euleriano para obter um ciclo hamiltoniano. E, também, uma **main**, que

faz a leitura do arquivo, as funções e mostra os resultados de todos os passos, além de uma visualização do grafo e seu caminho pelo algoritmo de Christofides. A seguir, descreve-se cada componente principal:

- **`ler_matriz_do_arquivo(nome_arquivo)`**: Esta função é responsável por abrir o arquivo, analisar seu conteúdo e convertê-lo em uma estrutura de dados de matriz (lista de listas em Python) para manipulação pelas demais funções.
- **`calcular_arvoreagm(matrizadj, num_vertices)`**: Implementa a primeira etapa do algoritmo. Implementa a primeira etapa do algoritmo. A implementação foi otimizada com uma fila de prioridade (heap de mínimo) para selecionar a aresta de menor peso a cada iteração, garantindo eficiência. Esta função foi desenvolvida sem o auxílio de bibliotecas de grafos, conforme exigido.
- **`encontrar_vertices_grau_impar(arestas_agm, num_vertices)`**: Esta função percorre a AGM gerada na etapa anterior para identificar todos os vértices que possuem um grau ímpar. O resultado é uma lista contendo os identificadores desses vértices.
- **`encontrar_emparelhamento_minimo(matriz_adj, vertices_impares)`:** Para esta etapa, um subgrafo induzido apenas pelos vértices de grau ímpar é criado. Utilizando a função nx.min_weight_matching da biblioteca NetworkX, é encontrado o emparelhamento perfeito de custo mínimo que conecta esses vértices em pares.
- **`encontrar_ciclo_euleriano(multigrafo_arestas, num_vertices)`**: As arestas da AGM e do emparelhamento mínimo são unidas para formar um multigrafo H, no qual todos os vértices garantidamente possuem grau par. Em seguida, o **Algoritmo de Hierholzer** foi implementado para encontrar um ciclo euleriano nesse multigrafo. A implementação foi feita manualmente, utilizando uma estrutura de pilha para gerenciar o percurso, atendendo às restrições do projeto.
- **`fazer_atalho_e_calcular_custo(ciclo_euleriano, matriz_adj)`**: Na etapa final, o ciclo euleriano é transformado em um ciclo hamiltoniano. A função percorre a sequência de vértices do ciclo euleriano, utilizando um conjunto para registrar os nós já visitados. Ao encontrar um vértice repetido, um "atalho" é criado, pulando-o e garantindo que cada vértice seja visitado apenas uma vez. Ao final, a função calcula o custo total do ciclo hamiltoniano resultante.

E por fim nossa main, que lê o nome do arquivo e o passa para as outras funções para formatar os dados conforme o proposto no documento do trabalho e que se utiliza de vários prints para mostrar os dados obtidos nessas funções.

3.3 Métodos:

O desenvolvimento seguiu uma abordagem incremental e modular. Algumas funções foram implementadas e testada isoladamente, utilizando pequenos grafos de exemplo cujos resultados podiam ser verificados manualmente, enquanto outras foram verificadas em conjunto utilizando instâncias oferecidas pelo professor. Após a validação de cada componente, individual ou não, as funções foram integradas em um script principal que executa a sequência completa do Algoritmo de Christofides e exibe os resultados parciais e finais no console, além de gerar uma visualização gráfica do ciclo encontrado.

3.4 Divisão de Tarefas:

A divisão de tarefas foi feita entre os discentes Carlos Eduardo Almeida, Luis Eduardo Martins, Tarcísio Alcanfor e Victor Lopes. O Carlos Eduardo cuidou do código com a ajuda do Luis Eduardo, com Luis Eduardo fazendo os 2 primeiros passos que mais tarde foram aprimorados por Carlos Eduardo, que também fez os outros passos. Já o relatório foi feito em conjunto entre Tarcísio, Victor e Luis Eduardo.

4. Resultados e Discussões:

Os testes foram feitos seguindo grafos simples e principalmente as instâncias compartilhadas pelo professor. A análise detalhada das respostas paras as instâncias foram a principal métrica de validação para o algoritmo, pois eram o mais próximo do que será cobrado pelo professor.

Sendo assim tivemos duas instâncias de teste principais, `bayg29.tsp.txt e si175.tsp.txt`, iremos abordar cada uma e seus resultados.

bayg29.tsp:

Esse foi o primeiro arquivo testado e também, dentre os dois, o mais simples. Notamos que o peso da Árvore Gerado Mínima foi exatamente igual (1319.0), o que demonstra que o algoritmo de Prim está correto.

Outro ponto em que pudemos comparar foi o Ciclo Hamiltoniano da Solução final, que apesar de a nossa resposta estar em uma ordem diferente
`([20, 4, 25, 28, 2, 8, 5, 11, 27, 0, 23, 26, 15, 7, 22, 6, 24, 18, 14, 10, 21, 16, 13, 17, 3, 9, 12, 19, 1, 20])`
comparado a resposta disponibilizada pelo professor
`([0, 23, 26, 15, 7, 22, 6, 24, 18, 14, 10, 21, 16, 13, 17, 3, 9, 12, 19, 1, 20, 4, 25, 28, 2, 8, 5, 11, 27, 0])`
temos essencialmente o ciclo, diferindo somente no vértice inicial.

Por fim, ao analisar o custo total, chegamos a 1716.0 que é exatamente o custo compartilhado no arquivo com a solução. Representando apenas algo em torno de 6%/7% de diferença ao valort da solução ótima (1610).

si175.tsp:

Esse arquivo foi um pouco mais díficil de analisar o passo a passo devido ao seu tamanho de 175 vértices fica inviável analisar o Ciclo Hamiltoniano da Solução Final.

Analisando o peso da AGM obtivemos 20762.0 que é o mesmo valor da solução compartilhada pelo professor. Por fim, o custo Total da Solução foi 21966.0, diferindo do custo apresentado nos resultados do professor (21954.0) em 12 unidades. Apesar dessa diferença, o algoritmo do professor é de 2,5% pior que o valor ótimo(21407.0) enquanto o nosso é, aproximadamente, 2,6% pior, o que ainda esta no limite do algoritmo de Christofides que é de 1,5x pior, ou

seja, até 50% pior que o ótimo. Tal diferença pode ser somente detalhes de implementação ou como o algoritmo construiu suas estruturas.

Assim, com base nos testes utilizando as instâncias podemos concluir que o algoritmo encontra corretamente o Ciclo Hamiltoniano da Solução Final, por consequência todos os outros passos estão corretos.

5. Conclusão:

Em suma, este trabalho possibilitou a implementação de uma solução aproximada para o Problema do Caixeiro Viajante Métrico (Δ-TSP), utilizando como base o renomado Algoritmo de Christofides. As etapas fundamentais do algoritmo, desde a construção da Árvore Geradora Mínima e a identificação de vértices de grau ímpar, até o emparelhamento perfeito de custo mínimo (com o auxílio de uma biblioteca externa, conforme permitido), a formação do multigrafo euleriano, a determinação do ciclo euleriano e o posterior atalho para o ciclo hamiltoniano, foram implementadas com sucesso. Os resultados demonstram a capacidade do algoritmo em fornecer um ciclo hamiltoniano com seu custo total, atendendo aos requisitos do problema.

A execução do projeto permitiu um aprofundamento significativo nos conceitos de algoritmos em grafos, especialmente no que tange às Árvores Geradoras Mínimas e Ciclos Eulerianos, cujo desenvolvimento foi feito sem o auxílio de bibliotecas prontas. Os desafios inerentes à implementação de cada etapa, especialmente a integração entre elas, proporcionaram um valioso aprendizado prático sobre a complexidade e a aplicação de algoritmos em problemas de otimização combinatória. Este trabalho reforça a importância de algoritmos de aproximação para problemas NP-difíceis, como o TSP, oferecendo soluções viáveis e com garantia de desempenho em cenários práticos. Como futuras melhorias, sugere-se a exploração de outras técnicas de otimização local para refinar o ciclo hamiltoniano obtido.

6. Referências:

As referências utilizadas durante o desenvolvimento foram, por parte, os materiais e slides apresentados em sala de aula, juntamente com atividades práticas.

CRATEÚS
2025