Trabalho prático 2
Algoritmos II

Igor Eduardo Martins Braga - 2022425671

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais - UFMG

igoreduardobraga@ufmg.br

Abstract. *The main goal of this work is to analyze the differences between three implementations of algorithms for solving the traveling salesman problem: Branch-and-Bound, twice-around-the-tree, and Christofides. This analysis takes into account the execution time, memory usage, and distance from the optimal solution.*

Resumo. *O objetivo principal deste trabalho é comparar três formas diferentes de implementar algoritmos para resolver o problema do caixeiro viajante: Branch-and-Bound, twice-around-the-tree e Christofides. Nessa comparação, vamos avaliar o tempo que cada algoritmo leva para ser executado, a quantidade de memória utilizada e quão próxima a solução encontrada está da solução ótima.*

1. Introdução

O problema do caixeiro viajante é um desafio bem conhecido no campo de otimização e pesquisa operacional. Seu objetivo é descobrir o caminho mais curto que um vendedor pode percorrer para visitar várias cidades e voltar ao ponto de partida, sem visitar a mesma cidade mais de uma vez. Este problema é importante tanto teoricamente quanto na prática, e tem aplicações em áreas como logística e planejamento de itinerários.

Nesse trabalho, utilizou-se diferentes métodos para testar a aplicação do problema do caixeiro viajante na prática. Os três algoritmos usados foram:

Branch and Bound: Este é um método geral usado para encontrar soluções ótimas de vários problemas de otimização. No contexto do problema do caixeiro viajante, ele explora sistematicamente todas as possíveis rotas (ou ”ramificações”) e elimina aquelas que claramente não levam à solução ótima (o ”limitante”). Este processo de ramificação e limitação continua até que a melhor solução seja encontrada.

Twice Around the Tree: Este método envolve a criação de uma árvore geradora mínima do conjunto de cidades e, em seguida, a criação de um circuito que passe duas vezes por cada aresta da árvore. Embora não garanta a obtenção da rota ótima, este método é relativamente simples e pode fornecer uma boa aproximação da solução ideal em um tempo computacionalmente razoável.

Algoritmo de Christofides: Este método é uma heurística que encontra uma rota cujo comprimento não é mais do que 50% mais longo do que o ótimo. O algoritmo começa com uma árvore geradora mínima, encontra um emparelhamento mínimo de custo entre as cidades com grau ímpar na árvore e combina esses dois componentes para formar um circuito fechado.

2. Implementações

2.1. Branch and Bound

Para implementar o método de Branch and Bound para resolver o problema do caixeiro viajante, define-se uma classe Node que representa um nó na árvore de pesquisa. Cada nó armazena informações como o caminho até o momento, a matriz de custos reduzida, o custo acumulado, o vértice atual e o nível do nó na árvore.

O método `reduce_matrix` é usado para reduzir a matriz de custos e calcular o custo limite inferior para um caminho parcial. Isso é feito subtraindo o mínimo não infinito de cada linha e coluna da matriz de custos. O custo resultante da redução é adicionado ao custo do caminho.

O algoritmo começa com a criação de um nó raiz que representa o estado inicial com a matriz de adjacências completa. Este nó é inserido em uma fila de prioridades (pq), que é ordenada com base no custo.

O processo continua em um loop enquanto a fila de prioridades não estiver vazia. Em cada iteração, é extraído o nó com menor custo. Se o nó representa um estado onde todos os vértices foram visitados (exceto o retorno ao vértice inicial), o caminho é finalizado e o custo total é retornado.

Caso contrário, o algoritmo gera sucessores do nó atual. Para cada vértice adjacente não visitado, um novo nó é criado com uma matriz de custos atualizada e um custo acumulado. A matriz é atualizada tornando infinitos os custos das arestas que não podem mais ser usadas no caminho atual. Esses novos nós são então adicionados à fila de prioridades.

O algoritmo continua até que o caminho ótimo seja encontrado, retornando o custo total desse caminho.

Esse é o método com maior complexidade entre os três tendo uma complexidade de O(n!). Apesar de ser exponencial, o algoritmo tenta podar partes da árvore de busca para evitar a exploração de todas as permutações, tornando-o mais eficiente do que uma busca ingênua.

2.2. Twice Around the Tree

Para implementar o método de Twice Around the Tree em grafos, inicialmente constrói-se uma Árvore Geradora Mínima (Minimum Spanning Tree, MST) do grafo dado, utilizando a biblioteca NetworkX. Após a construção da MST, realiza-se uma busca em profundidade (Depth-First Search, DFS) partindo de um nó especificado, e obtém-se o caminho em pré-ordem. Este caminho é então seguido duas vezes para formar um ciclo Hamiltoniano aproximado, retornando ao ponto de partida para completar o ciclo. Finalmente, a função `calculate_weight`, que calcula o peso total do caminho percorrido no grafo, é utilizada para determinar o custo total do ciclo.

A construção da MST pode ser feita em O(n$\hat{2}$) usando o algoritmo de Prim ou Kruskal, e percorrer a árvore duas vezes adiciona complexidade O(n), resultando em uma complexidade total de O(n$\hat{2}$). Embora não seja uma solução ótima, é mais rápida do que abordagens exatas.

2.3. Christofides

Para implementar o método de Christofides, seguiu-se esses passos:

1. **Construção da Árvore Geradora Mínima (MST):** Utilizando a biblioteca NetworkX, o algoritmo cria uma MST do grafo de entrada, que serve como base para o caminho a ser seguido, minimizando o peso total das arestas.
2. **Criação de um Grafo Euleriano:** A função `create_eulerian_graph` identifica todos os nós com grau ímpar na MST e aplica um emparelhamento de peso mínimo neles, dentro do grafo original. Essas novas arestas são adicionadas à MST para formar um grafo euleriano.
3. **Encontrar o Circuito Euleriano:** A função `find_shortest_path` encontra um circuito euleriano no grafo, removendo repetições de vértices para criar um caminho que visita cada vértice pelo menos uma vez.
4. **Cálculo do Peso Total:** O peso total do caminho encontrado é calculado usando a função `calculate_weigth`.
   A construção da MST ($O(n\hat{2})$), encontrar um circuito de Euler ($O(n\hat{2})$), e ajustar para obter um ciclo hamiltoniano mínimo ($O(n\hat{3})$) resulta em uma complexidade total de aproximadamente $O(n\hat{3})$. Embora seja mais eficiente do que a busca exaustiva, ainda é uma heurística com complexidade cúbica.

3. Apresentação dos experimentos e análise dos resultados

OBS: O custo é o peso total do grafo percorrido por cada algoritmo

Para realizar testes, utilizaram-se 78 conjuntos de dados, consistindo em grafos de diversos tamanhos. Implementou-se um procedimento na função principal do programa para iterar por esses conjuntos de dados. Este procedimento organiza os datasets em ordem crescente de tamanho. Além disso, dependendo do nome do algoritmo especificado no comando de execução, o programa executa o algoritmo correspondente em todas as 78 instâncias. A decisão de ordenar os conjuntos de dados por tamanho visa facilitar a análise do aumento do uso de memória e do tempo de processamento, que são registrados em um arquivo CSV. Isso permite uma avaliação mais clara e intuitiva do desempenho e eficácia de cada método.

Os resultados da nossa análise indicam que, conforme o tamanho do grafo aumenta, o tempo de execução e o uso de memória também crescem. Isso era esperado. Uma constatação crucial é que, no método de Christofides, o aumento tanto no tempo quanto no uso de memória é mais acentuado em comparação com o método Twice Around the Tree. Esta diferença notável é atribuída à maior complexidade da implementação personalizada do método de Christofides.

Ao comparar os dois algoritmos, a diferença mais marcante está na eficiência deles ao processar conjuntos de dados de grande escala. Como mostrado na figura 1, o tempo de execução do algoritmo de Christofides cresce de forma significativa em relação ao Twice Around the Tree para conjuntos de dados maiores.

Portanto, com base nesta análise e nas implementações testadas, concluímos que o método Twice Around the Tree é mais eficiente e adequado para conjuntos de dados de grande dimensão.

Table 1. Tabela Twice Around the Tree

Nome do Teste	Custo	Tempo de Execução (s)	Memória (MB)	Distância do ótimo
eil51.tsp	584	2.449954	50.968750	158
st70.tsp	888	2.452049	51.847656	213
eil76.tsp	696	2.439678	52.960938	158
berlin52.tsp	10114	2.445827	53.464844	2572
eil101.tsp	830	2.435184	55.718750	201
...	...	...	...	...
fnl4461.tsp	255829	100.8748	4248.875	73263

Essa tabela mostra brevemente os resultados obtidos a partir do método de Twice Around the Tree. Para analisar ela completamente acesse o arquivo twice_completo.csv do repositório.

Table 2. Tabela Christofides

Nome do Teste	Custo	Tempo de Execução (s)	Memória (MB)	Distância do ótimo
eil51.tsp	462	2.523001	51.992188	36
st70.tsp	770	1.358029	52.898438	95
eil76.tsp	608	1.416868	54.179688	70
berlin52.tsp	8591	2.228441	55.210938	1049
eil101.tsp	707	1.884336	56.835938	78
...	...	...	...	...
fnl4461.tsp	205834	6891.566	4428.5625	23268

Essa tabela mostra brevemente os resultados obtidos a partir do método de Christofides. Para analisar ela completamente acesse o arquivo chritofides_completo.csv do repositório.

Além disso, é crucial ressaltar que a proximidade da solução ideal difere entre os dois algoritmos. O método de Christofides alcança uma solução mais próxima do ideal em comparação ao Twice Around the Tree. Isso pode ser observado pelo primeiro gráfico da figura 1, onde o método de Christofides obteve custos menores do que o método de Twice Around the Tree (o resultado do ótimo está abaixo do resultado do de Christofides). Isso era esperado, dado que os fatores de aproximação desses algoritmos são, respectivamente, 1.5 e 2. Esta observação confirma o que discutimos em sala de aula.

Por fim, é importante mencionar que o método de branch and bound mostrou um desempenho insatisfatório. Não foi possível executá-lo em nenhum conjunto de dados, pois sempre excedeu o limite de tempo estabelecido de 30 minutos. Isso destaca a alta complexidade desse método e sua aplicabilidade limitada na prática, visto que ele exige longos períodos de espera para produzir um resultado satisfatório, especialmente em grafos de grande porte. Em alguns casos, pode levar de horas a anos para completar. Portanto, seu uso é recomendado apenas quando é imprescindível obter o resultado ótimo, já que ele sempre consegue encontrá-lo.

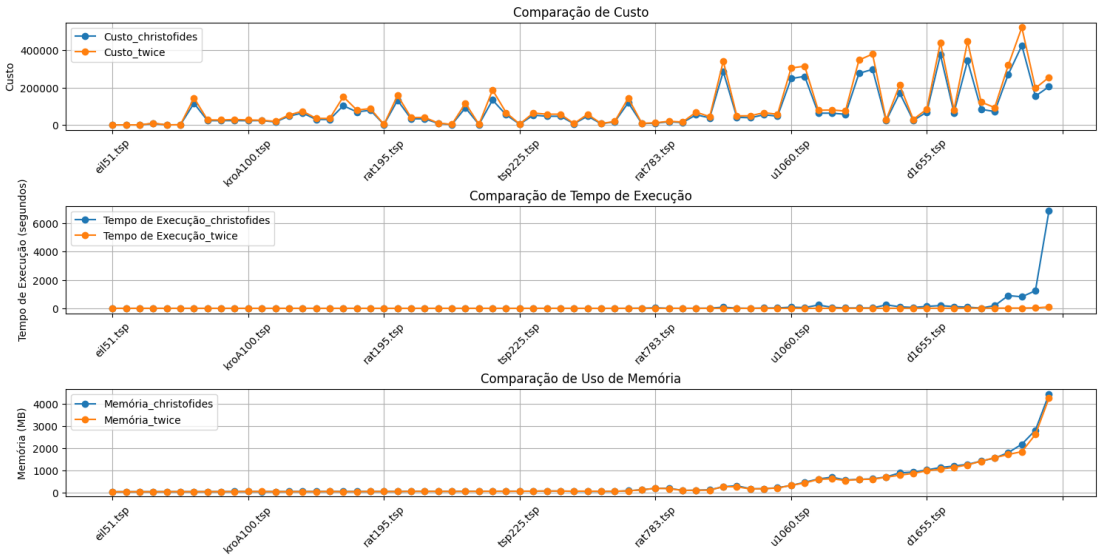

Figure 1. Comparação dos algoritmos Twice Around the Tree e Christofides

4. Conclusão

Com base no que foi desenvolvido durante o trabalho, pôde-se observar grandes diferenças entre os métodos utilizados, destacando-se principalmente em termos de eficiência e proximidade da solução ideal. O método Twice Around the Tree demonstrou maior eficiência em termos de tempo de execução e uso de memória, especialmente para conjuntos de dados de grande escala, enquanto o método de Christofides, apesar de mais exigente em recursos, alcançou soluções mais próximas do ideal. O método de branch and bound, por sua vez, mostrou-se impraticável em termos práticos devido à sua alta complexidade e longos tempos de execução, sendo recomendado apenas em situações onde é estritamente necessário obter a solução ótima. Estas observações reforçam a importância de escolher o algoritmo apropriado com base no objetivo a ser alcançado.

5. References

CHANG, B. Traveling Salesman Problem and Approximation Algorithms. 10 Feb. 2019. Available at: https://bochang.me/blog/posts/tsp/. Accessed on: 30/11/2023.