# Algoritmos 2 - Soluções para problemas difíceis

Pedro Henrique Meireles de Almeida- 2021031440
UFMG - Universidade Federal de Minas Gerais

December 2023

## 1 Introdução

O problema do caixeiro viajante é um problema que consiste em determinar a rota mais curta possível que passe pelo conjunto de cidades e retornar à cidade de origem, visitando cada cidade exatamente uma vez. As cidades são representadas por coordenadas no plano euclidiano.

O trabalho tem como foco a implementação e análise de três algoritmos para solucionar o problema do caixeiro viajante: o algoritmo exato de branch-and-bound, o algoritmo Twice-Around-the-Tree e o algoritmo de Christofides. O objetivo principal é fazer uma comparação entre essas três abordagens, debatendo sobre qual delas usar em certos cenários. O algoritmo branch-and-bound demorou muito para todos os grafos do dataset, então foi criado 3 novos datasets com 12, 13 e 14 cidades.

## 2 Twice-Around-the-Tree

Através da biblioteca NetworkX em Python, o algoritmo Twice-Around-the-Tree começa encontrando uma árvore geradora mínima no grafo. Em seguida, a função executa uma busca em profundidade na árvore geradora mínima para formar um ciclo euleriano. Isso é feito utilizando a função **nx.dfs_preorder_nodes** do NetworkX, que retorna os nós na ordem de uma travessia DFS. A implementação do algoritmo é bastante simples. Os datasets usados têm métricas euclidianas, então a aproximação do algoritmo é 2.

### 2.1 Complexidade de Tempo

A complexidade de tempo é dominada pelo algoritmo usado para construir a árvore geradora mínima, nesse caso $O(ElogV)$. A construção do ciclo euleriano tem complexidade $O(V)$ e a busca em profundidade $O(V + E)$. Isso resulta em uma complexidade de $O(ElogV)$.

### 2.2 Complexidade de Espaço

O algoritmo somente consome o espaço para armazenar a árvore geradora mínima e o ciclo euleriano. Isso requer espaço proporcional ao número de vértices do grafo.

## 3 Christofides

O algoritmo de Christofides é uma solução eficiente para resolver o Problema do Caixeiro Viajante. Primeiramente, é identificado os vértices com grau ímpar na árvore geradora mínima do grafo, utilizando a biblioteca NetworkX em Python. Em seguida, é feito um pareamento mínimo entre esses vértices utilizando a função **nx.min_weight_matching**. Posteriormente, é criado um grafo que consiste na árvore geradora mínima com as arestas adicionadas pelo pareamento mínimo. Por fim, é calculado o menor caminho passando por todos os vértices do grafo. Isso é alcançado através da obtenção de um circuito euleriano usando a função **nx.eulerian_circuit**. Em seguida, o caminho é percorrido, somando os pesos das arestas para calcular o custo total do percurso.

### 3.1 Complexidade de Tempo

A complexidade do algoritmo de Christofides é determinada pela etapa do pareamento perfeito mínimo. A complexidade dessa etapa pode ser $O(n^3)$ nos piores casos.

### 3.2 Complexidade de Espaço

A complexidade de espaçopara o algoritmo de Christofides, igual ao caso do algoritmo Twice-Around-the-Tree, é determinada pelo espaço utilizado para armazenar a árvore geradora mínima. As estruturas de dados auxiliares também são proporcionais ao número de vértices.

## 4 Branch-and-Bound

O algoritmo Branch and Bound é um algoritmo exato que tenta encontrar a solução ótima. Ele realiza uma busca exaustiva, avaliando todas as possíveis combinações de caminhos para determinar a solução ótima do problema. No entanto, evita certas buscas para evitar explorar ramos que sabidamente não levarão a soluções melhores que a melhor solução atual.

O algoritmo utiliza uma pilha para armazenar os nós a serem avaliados. Cada nó na pilha representa um caminho que está sendo explorado. A seleção dos nós a serem avaliados na próxima iteração leva em conta a qualidade da solução até o momento.

### 4.1 Complexidade de Tempo

No pior caso a complexidade de tempo do algoritmo Branch-and-Bound é $O(n!)$. Isso ocorre porque, mesmo com as podas realizadas para evitar certos ramos da árvore de busca, o algoritmo ainda explora uma quantidade significativa de soluções possíveis, que cresce de forma fatorial em relação ao número de cidades.

## 5 Resultados

### 5.1 Gráficos

A Figura 1 apresenta uma comparação entre os tempos dos algoritmos Twice Around the Tree e Christofides. É claro que o Twice Around the Tree apresenta um tempo de execução significativamente mais rápido em comparação com o Christofides. No entanto, embora o Twice Around the Tree execute mais rapidamente, ele pode não encontrar soluções tão otimizadas quanto o Christofides, que, apesar de ter um tempo de execução maior, tende a fornecer soluções mais próximas da solução ótima. O algoritmo branch-and-bound demorou muito em todas as instâncias maiores que 15.

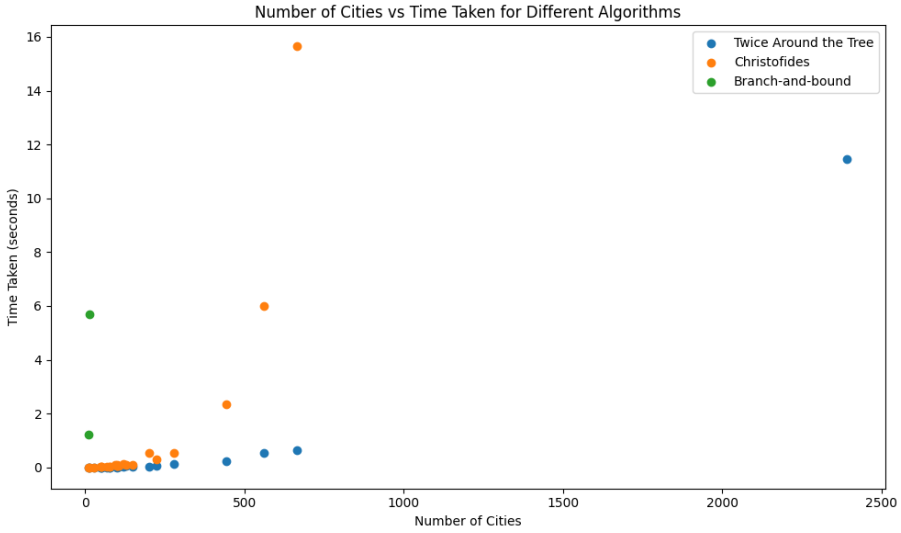


Figure 1: Número de cidades e o tempo em segundos

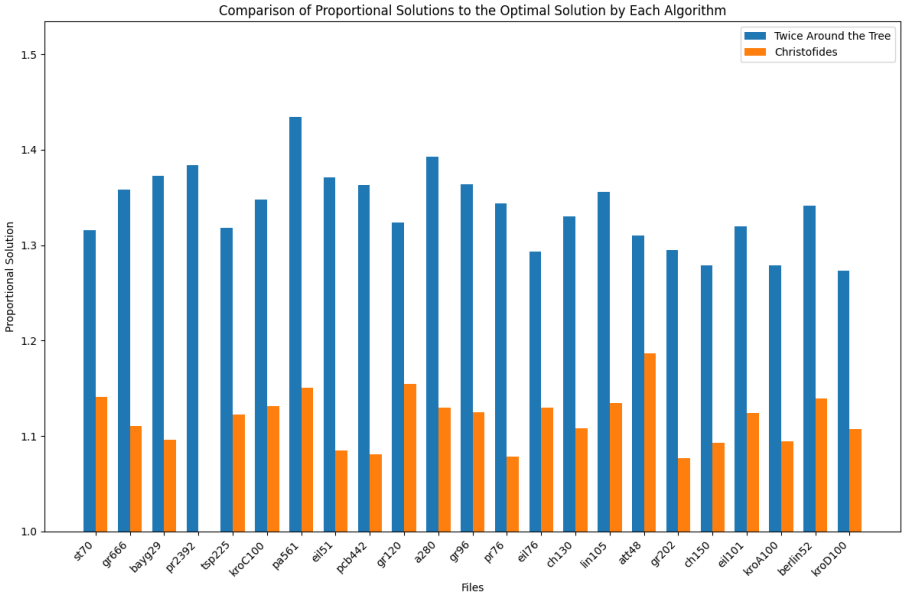


Figure 2: Comparação proporcional entre a solução encontrada do algoritmo e o ótimo

O algoritmo de Christofides, conforme observado na Figura 2, demonstra uma maior aproximação com a solução ótima em comparação com o Twice Around the Tree. Apesar do tempo de execução superior, o Christofides tende a encontrar soluções mais próximas do ótimo.

## 5.2 Comparação

- **Twice Around the Tree:** É rápido e pode fornecer um resultado aceitável. A solução, embora não seja ótima e seja inferior à do algoritmo de Christofides, é limitada a ser no máximo duas vezes pior que o ótimo. Portanto, deve ser usado para conjuntos de dados com mais de 400 cidades quando a velocidade for prioridade.
- **Christofides:** Apesar do tempo de execução mais longo, o algoritmo de Christofides é mais indicado para buscas de soluções mais próximas do ótimo, priorizando a precisão sobre a velocidade de execução. Especialmente eficaz para conjuntos de dados com menos de 300 cidades, onde sua capacidade de encontrar soluções com uma aproximação mais próxima da solução ideal supera a limitação de tempo. Não deverá ser utilizado para conjuntos maiores que 1000, devido a complexidade $O(n^3)$ citada anteriormente.
- **Branch-and-Bound:** Apesar de sempre apresentar a solução ótima, deverá apenas ser usado para problemas de escalas muito pequenas. Devido

à sua natureza exaustiva, ele se torna impraticável a partir de conjuntos com mais de 15 cidades, o que inviabiliza sua aplicação prática para conjuntos de dados extensos.

# 6 Conclusão

A escolha do algoritmo mais apropriado para resolver o problema depende da natureza do conjunto de dados, priorizando entre velocidade de execução, precisão da solução e escalabilidade do algoritmo. Para conjuntos de dados extensos, o Twice Around the Tree pode ser uma escolha eficiente, enquanto o Christofides é ideal para dados menores que demandam precisão e o Branch-and-Bound para problemas de pequena escala que requerem a solução ótima. Nesse contexto, os algoritmos aproximativos se destacam por oferecerem soluções satisfatórias em um tempo razoável. Embora não garantam a solução ideal, esses algoritmos são capazes de encontrar soluções próximas ao ótimo.