Problema do Caixeiro Viajante
Algoritmos e Heurísticas

Pedro F. Baptista

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

Abstract. *This assignment aims to analyze the Travelling Salesman Problem and the possible approaches to solve it. With this intent, one exact algorithm and two approximation algorithms were implemented. Each instance of the problem was solved by each algorithm and their performance was compared in terms of runtime, space and quality of the solution.*

Resumo. *Este trabalho tem como objetivo estudar o problema do Caixeiro Viajante e as possíveis abordagens para sua resolução. Para tal, foram implementados um algoritmo exato e dois aproximativos e a execução dos três comparada em relação ao tempo, ao espaço e à qualidade da solução.*

1. Conceitos Básicos

1.1. Problema do Caixeiro-Viajante

O problema do caixeiro-viajante (PCV) é um problema clássico de otimização na área de ciência da computação. Neste problema, um caixeiro-viajante recebe um conjunto de cidades, e sua tarefa é visitar cada cidade exatamente uma vez e retorna à cidade de origem, peloo percurso mais curto possível. O objetivo é minimizar a distância total e/ou o custo percorrido pelo caixeiro-viajante.

1.2. Algoritmos Exatos e Aproximativos

Algoritmos exatos são algoritmos que fornecem uma solução ótima para um determinado problema. No problema do caixeiro-viajante, um algoritmo exato garante encontrar o percurso mais curto absoluto que visita todas as cidades exatamente uma vez. Esses algoritmos asseguram que a solução encontrada seja a melhor possível dentro das restrições do problema.

Por outro lado, **algoritmos aproximativos** são algoritmos que fornecem uma solução quase ótima para um problema, muitas vezes com uma garantia sobre a qualidade da solução. No PCV, por exemplo, um algoritmo aproximativo pode não garantir o percurso mais curto, mas fornecerá uma solução próxima da ótima. Esses algoritmos geralmente são mais eficientes computacionalmente do que os algoritmos exatos e são particularmente úteis para resolver problemas nos quais encontrar uma solução exata é impraticável ou consome muito tempo. Algoritmos aproximativos são avaliados conforme o quão distantes estão da solução ótima por um fator C. Isso significa que um algoritmo 2-aproximativo pode produzir uma solução até duas vezes pior que a solução ótima.

2. Implementação

Os algoritmos foram implementados utilizando a linguagem Python 3.10.12 e a biblioteca Networkx, para auxiliar na construção de grafos e nas operações dentro deles.

2.1. Algoritmo Branch-and-Bound

O algoritmo Branch-and-Bound é um algoritmo exato para encontrar soluções ótimas para problemas de otimização combinatória. Ele parte da abordagem de exploração total do espaço de soluções, dividindo-o em subproblemas menores. Porém, para evitar gasto computacional desnecessário, ele faz uso de técnicas de limitação para eliminar subproblemas que certamente não levarão a uma solução ótima. O algoritmo mantém uma estrutura de árvore, onde cada nó representa um subproblema, e os ramos correspondem às diferentes escolhas ou decisões tomadas em cada etapa. Ele poda eficientemente o espaço de busca, concentrando-se em áreas promissoras para encontrar a solução ótima. [Levitin 2011]

2.2. Algoritmo Twice-Around-The-Tree

O algoritmo Twice-Around-The-Tree é um algoritmo 2-aproximativo desenvolvido para resolver o PCV. Primeiramente, ele constrói uma árvore Geradora Mínima do conjunto dado de cidades. Em seguida, percorre a AGM duas vezes, criando um tour visitando cada aresta duas vezes. Para obter um tour viável, o algoritmo ignora cidades repetidas durante a segunda travessia. Com as duas travessias, é possível encontrar um ciclo que retorna para a cidade original, resultando na resposta final. Embora o tour produzido por este algoritmo não seja ótimo, ele fornece uma aproximação até duas vezes pior para o problema. [Cormen 2009]

2.3. Algoritmo de Christofides

O algoritmo de Christofides é um outro algoritmo aproximativo para resolver o PCV proposto por Nicos Christofides em 1976. Ele combina técnicas da teoria dos grafos, como encontrar uma árvore geradora mínima e um emparelhamento perfeito mínimo para construir uma solução aproximada. O algoritmo constrói uma árvore geradora mínima, encontra um emparelhamento perfeito mínimo vértices da árvore com grau ímpar e combina a árvore geradora com o emparelhamento. Nesse grafo resultante, é encontrado um circuito euleriano que é transformado em um circuito hamiltoniano, que é a solução final. O algoritmo garante um fator de aproximação de 1.5 e, portanto, é mais próximo da solução ótima que o algoritmo Twice-Around-The-Tree. [Cormen 2009]

3. Método

Para testar os algoritmos implementados e comparar suas performances, foram utilizados os conjuntos de testes da biblioteca TSP disponibilizada pela Universidade de Heidelberg [Reinelt 2018]. Foram selecionados os seguintes conjuntos de testes: *berlin52*, *rat99*, *rd400*, *pr1002*, *u2152*, *pcb3038* e *d15112*. Esses conjuntos de testes obedeciam à restrição de métrica, que permite a aplicação dos algoritmos aproximativos.

Cada teste foi feito aplicando os três algoritmos e medindo o tempo de execução, espaço alocado (conforme a complexidade espacial do algoritmo) e o quão distante os algoritmos aproximativos estavam da resposta ótima.

4. Resultados

Durante a execução dos testes, rapidamente ficou claro que o algoritmo branch-and-bound é muito custoso para ser executado fora de uma máquina com capacidade alta e/ou sem estratégias de paralelização. Até para o menor conjunto de testes (berlin52, com 52 cidades)

o algoritmo demorou acima do limite estipulado de 30 minutos. Como foram disponibilizadas as soluções ótimas para os problemas, as comparações foram feitas a partir delas. Para os testes maiores, o algoritmo de Christofides também não conseguiu executar dentro do tempo limite. Para o último teste, nenhum algoritmo conseguiu executar dentro do tempo limite.

4.1. Tempo Gasto

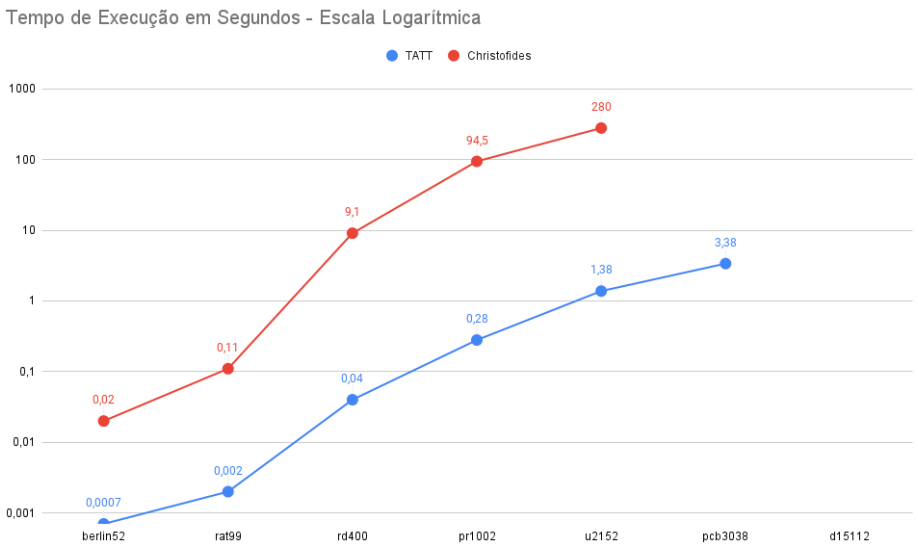
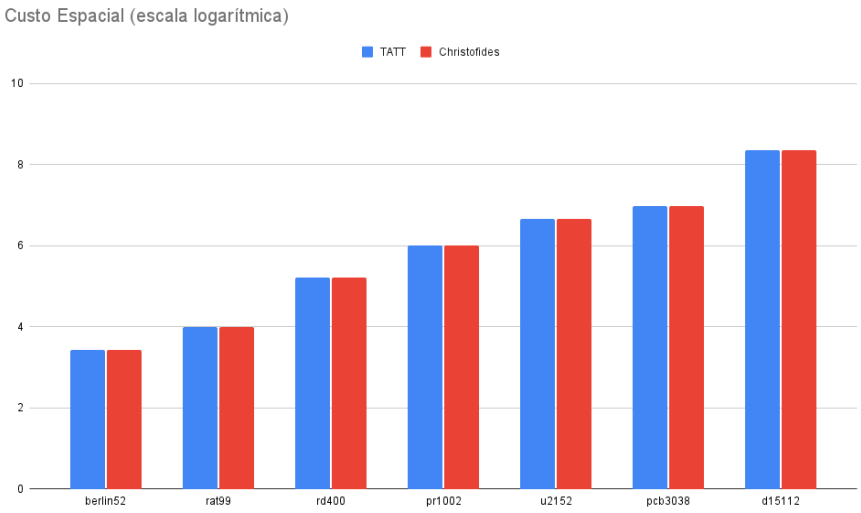


Figure 1. Tempo de Execução em Segundos - Escala Logarítmica

4.2. Espaço Gasto

Ambos os algoritmos foram implementados utilizando matrizes de adjacência para representar os grafos. Dessa maneira, a complexidade espacial de ambos é $O(n^2)$ e o gráfico reflete essa igualdade.



4.3. Proximidade da Solução

A métrica para a proximidade é a razão da solução encontrada pela original.

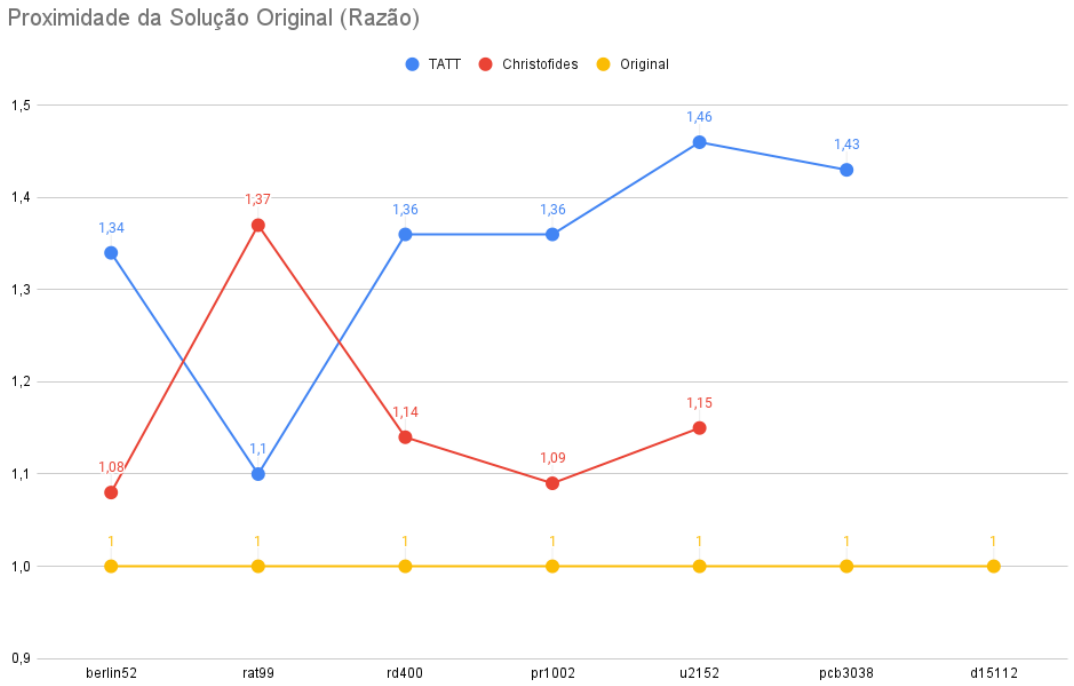


Figure 2. Proximidade da solução, em razão

5. Conclusão

Como foi possível enxergar através dos resultados, os algoritmos aproximativos chegam em respostas relativamente satisfatórias quando comparadas com a solução original. Considerando o imenso custo computacional de um algoritmo exato, a essa diferença é um preço que vale a pena ser pago. Se compararmos os dois algoritmos aproximativos entre si, é possível ver que o algoritmo de Christofides tem um custo de tempo que cresce muito mais rapidamente que o Twice-Around-The-Tree e sua perda de exatidão não é tão significativa.

References

Cormen, T. (2009). *Introduction to Algorithms*. MIT Press (MA).

Levitin, A. (2011). *Introduction to the Design and Analysis of Algorithms*.

Reinelt, G. (2018). TSPLIB 95. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/`. [Online; acesso em 05 de Dezembro de 2023].