Análise do desempenho de algoritmos ótimos e aproximados para o problema do caixeiro viajante (TSP) métrico

João Pedro Fernandes Silva
joaofernandes@dcc.ufmg.br

DCC - UFMG

12 de dezembro de 2022

Resumo

O problema do caixeiro viajante é um dos problemas provados NP-completo mais estudados em ciência da computação. Com diferentes estratégias para encontrar soluções ótimas ou aproximadas, se faz necessário um estudo sobre o desempenho prático destes algoritmos, visando clarear as limitações e particularidades de cada um.

O estudo comparativo utilizando instâncias artificiais do TSP com o algoritmo exato de branch and bound, o algoritmo 2-aproximado twice around the tree e o algoritmo de Christofides 1,5-aproximado, reforça a limitação do algoritmo exato que, em tempo hábil, resolveu apenas instâncias pequenas do problema. Além disso, o estudo mostrou uma ligeira superioridade nas soluções aproximadas encontradas pelo algoritmo de Christofides em relação ao twice around the tree, porém com tempo de execução muito superior.

1 Introdução

Definição 1. *Dado um grafo $G = (V, E)$, um ciclo hamiltoniano em G é um ciclo que contém todos os vértices do grafo.*

O problema do caixeiro viajante (em inglês: *travelling salesman problem*, ou simplesmente *TSP*) consiste de encontrar o ciclo hamiltoniano de menor peso em um grafo ponderado. A versão métrica do problema adiciona a restrição de que os pesos das arestas de G (distância) devem seguir a desigualdade triangular, i.e. $w_{ij} <\leq w_{ik} + wkj \ \forall i, j, k \in V$. O problema é NP-completo [1], o que faz com que estratégias que encontram soluções ótimas para o problema tenham alto custo computacional, de tempo e espaço. Sendo assim, faz-se necessário o uso de outras técnicas para encontrar soluções para este tipo de prolema. Algoritmos aproximativos têm a qualidade da solução garantida, no pior caso, por um fator de aproximação em relação à solução ótima, fazendo com que sejam alternativas polinomiais seguras de soluções. Este artigo apresenta um estudo comparativo entre três algoritmos que encontram soluções para o problema do caixeiro viajante métrico: branch and bound para o TSP métrico, twice around the tree e o algoritmo de Christofides, comparando métricas de qualidade da solução encontrada, tempo de execução do algoritmo e memória alocada.

2 Algoritmos

2.1 Branch and bound

A técnica de branch and bound consiste em explorar a árvore de possibilidades da instância de maneira "guiada", dando prioriade a nós que parecem promissores e impedindo a exploração de ramos desnecessários, a fim de encontrar uma solução ótima para o problema. Para cada possível nó a ser adicionado ao caminho atual é feito o cálculo do limite inferior da solução, impedindo que ramos com soluções piores que aquela já encontrada sejam sejam explorados, o custo de tempo do cálculo do limite inferior para uma solução parcial de um grafo $G = (V, E)$ é $O(|V|)$, uma vez que a função itera entre

todos os nós, somando as médias entre as duas arestas de peso mínimo adejacentes ao nó. Uma vez que um ramo é possível de ser explorado, ele é adicionado à uma fila de prioridades, e assim a árvore de possibilidade é sendo tratada. No pior caso, o algoritmo de branch and bound se assemelha ao de força bruta, tendo que explorar todas as permutações de vértices do grafo, além de calcular a expecitativa da solução para cada uma das permutações, fazendo com que o algoritmo tenha complexidade de tempo $O(2^{|V|} * |V|)$ e de espaço auxiliar $O(2^{|V|})$.

2.2 Twice around the tree

O algoritmo twice around the tree é um algoritmo 2-aproximativo para o TSP. O algoritmo consiste em encontrar a árvore geradora mínima para o grafo de entrada $G = (V, E)$ e realizar um caminhamento em pré-ordem no grafo da árvore geradora mínima ignorando vértices duplicados. Este caminhamento retorna um caminho hamiltoniano em G, e que ao transformá-lo em um ciclo hamiltoniano, gera uma solução que é, no máximo, 2 vezes a solução ótima para o problema do caixeiro viajante para G. O algoritmo tem custo de tempo dominado pela computação da árvore geradora mínima, utilizando o algoritmo de Prim [2] para computá-la, o algoritmo twice around the tree tem custo de tempo $O(|V|^2)$.

2.3 Algoritmo de Christofides

O algoritmo de Christofides [3] é um algoritmo 1,5-aproximativo para o TSP. Consiste em encontrar o emparelhamento mínimo de vértices no subgrafo induzido pelas arestas de grau ímpar da árvore geradora mínima para o grafo de entrada $G = (V, E)$, e adicionar as arestas do emparelhamento à árvore. Após isso realiza-se o caminhamento em pré ordem no grafo gerado, retornando um caminho hamiltoniano em G, e que ao transformá-lo em um ciclo hamiltoniano, gera uma solução que é, no máximo, 1,5 vezes a solução ótima para o problema do caixeiro viajante para G. O algoritmo de Christofides tem custo de tempo dominado pela computação do emparelhamento de peso mínimo, utilizando o algoritmo de Blossom [4] para emparelhamento mínimo, o algoritmo de Christofides tem custo de tempo $O(|V|^3)$.

3 Experimentos

O objetivo da experimentação foi analisar o desempenho prático dos algoritmos e compará-los de forma a entender as limitações e particularidades de cada um. Especificamente, tentar responder os seguintes questionamentos:

- Qual o limite prático do algoritmo de solução exata baseado em branch and bound?
- Para as soluções em que se conhece o ótimo, qual a qualidade das soluções aproximadas encontradas?
- Em relação a qualidade da solução encontrada, há unanimidade entre os algoritmos aproximativos?

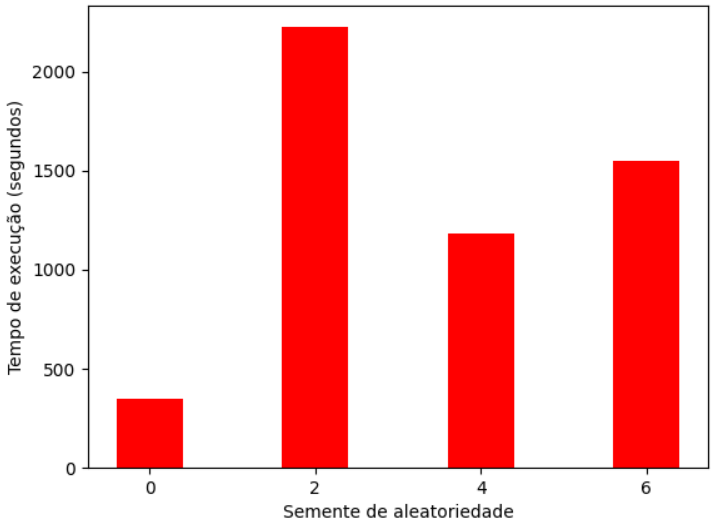
3.1 Configuração

Todos os algoritmos foram implementados na linguagem python [4] utilizando a biblioteca *netowrkx* [5] para auxiliar na representação dos grafos. Foram gerados dois conjuntos de instâncias diferentes, um com grafos completos aleatórios com arestas de peso w com $1 \leq w \leq 1000$ com arestas obedecendo a distância euclidiana entre os vértices, e outro grupo de grafos completos aleatórios com arestas de peso w com $1 \leq w \leq 1000$ com arestas obedecendo a distância Manhattan entre os vértices. Os grafos tinham tamanho 2^i com i variando de 4 a 10. Os experimentos foram realizados em uma máquina com sistema operacional Ubuntu 22.04.1 LTS (Jammy Jellyfish), 8GB de memória RAM e processador Intel Core i5-8250U CPU @ 1.60GHz 1.80GHz.

3.2 Resultados e análise

Tempo de execução. Foi adotado o limite de 30 minutos para a execução de cada um dos algoritmos. Dessa forma, o algoritmo de branch and bound apenas completou sua execução com instâncias de tamanho $i = 4$ (i.e. 2^4 vértices). Porém ao variar a semente de aleatoriedade na geração do grafo, pôde-se observar que houve alta variância no tempo de execução do algoritmo, o que mostra que em instâncias específicas em que o ótimo é encontrado rapidamente, o algoritmo tem um melhor desempenho deixando de explorar mais ramos desnecessários. A média do tempo de execução do branch and bound com 4 sementes de aleatoriedade diferentes, utilizando a distância euclidiana, foi de **22.08 minutos**.

Figura 1: Branch and bound com diferentes sementes



Ao comparar o tempo de execução dos algoritmos aproximativos, o algoritmo de Christofides mostrou uma curva de crescimento de tempo muito maior que o algoritmo twice around the tree. Isso era esperado, uma vez que o algoritmo de Christofides tem complexidade de tempo de maior grau em relação ao twice around the tree. A média total do tempo de execução do algoritmo de Christofides foi de **16,73 segundos**, enquanto o algoritmo twice around the tree teve média de tempo de execução de **0,82 segundos**. Não houveram diferenças significativas entre o tempo de execução dos algoritmos aproximativos usando distância euclidiana ou Manhattan.

Figura 2: Execução com distância euclidiana

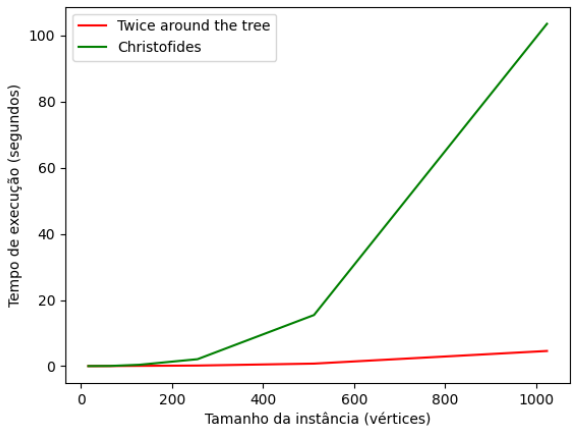
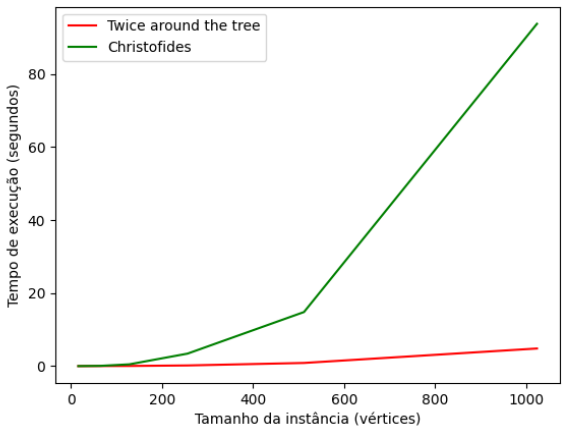


Figura 3: Execução com distância Manhattan



Qualidade da solução. Comparados com a solução ótima, os dois algoritmos aproximativos tiveram desempenho parecido. O algoritmo de Christofides teve média geral de solução **21%** mais

custosa que o ótimo, enquanto o algoritmo twice around the tree teve média de solução **17,22%** mais custosa que o ótimo. Observou-se que o algoritmo twice around the tree, para esta instância de tamanho $i = 4$ e utilizando 4 sementes de aleatoriedade diferentes, obteve desempenho médio geral melhor que o algoritmo de Christofides, que tem um fator de aproximação menor. É importante ressaltar que este fenômeno não vai contra a prova dos fatores de aproximação, uma vez que o fator de um algoritmo aproximativo é um limite superior para a solução retornada, relacionado com a solução ótima.

Figura 4: Execução com distância euclidiana

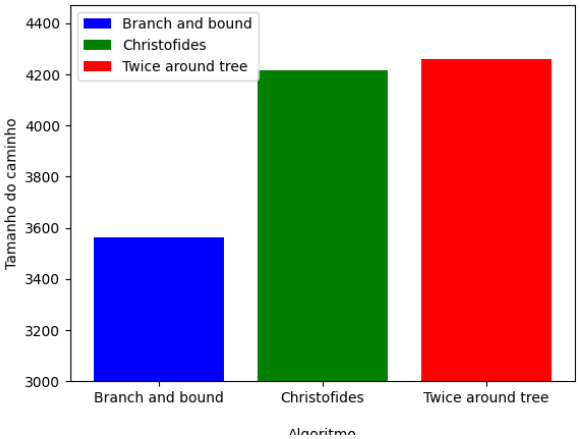
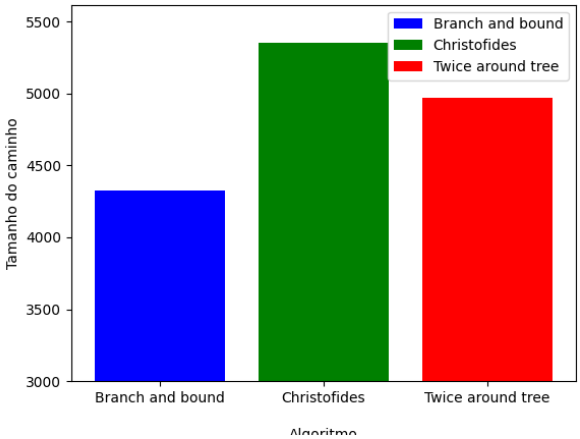


Figura 5: Execução com distância Manhattan



Comparando apenas as soluções aproximadas para as demais instâncias, mesmo com o algoritmo twice around the tree tendo fator de aproximação superior ao algoritmo de Christofides, os dois algoritmos encontraram soluções próximas independente da função de distância utilizada, com o algoritmo de Christofides tendo um desempenho médio geral superior de **4,8%**.

Figura 6: Execução com distância euclidiana

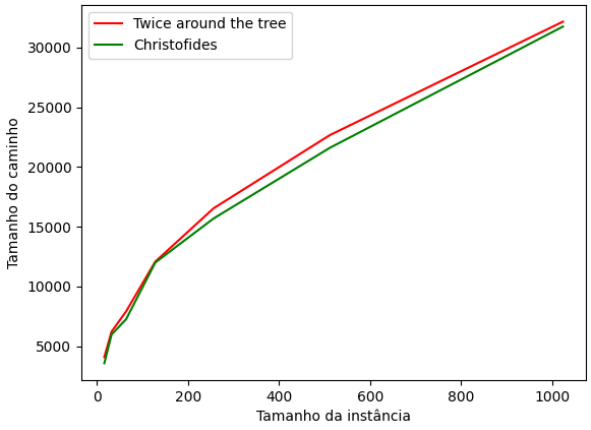
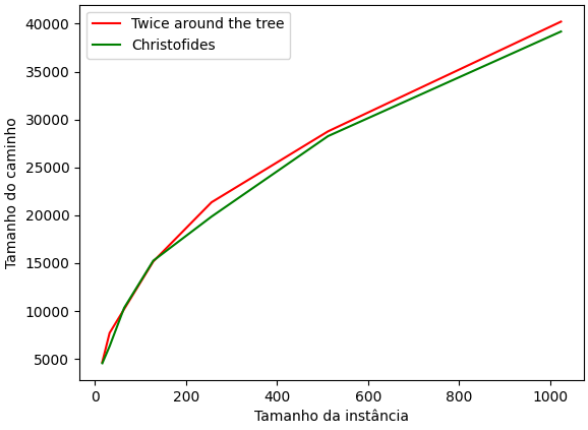


Figura 7: Execução com distância Manhattan



Pico de memória utilizada. O algoritmo de branch and bound, teve média de pico de consumo de memória de **992 MB**, Christofides teve de **0,0678 MB** e twice around the tree teve de **0.0473 MB**. Ja era esperado um consumo muito maior de memória do algoritmo branch and bound: uma vez que os ramos a serem explorados são alocados em uma fila de proridades e há possivelmente $2^{|V|}$ ramos a serem explorados, a quantidade de memória alocada pelo algoritmo exato é bem maior que os algoritmos aproximativos. Este consumo exacerbado de memória que o algoritmo de branch and bound demanda, reforça a inviabilidade do algoritmo mesmo para instâncias pequenas do problema: a tentativa de execução do algoritmo branch and bound em instâncias de tamanho $i = 5$ já fez com que toda a memória do computador fosse consumida, sendo necessária a reinicialização da máquina.

Os algoritmos aproximativos tiveram média de pico de memória utilizada igual (diferença menor que 1%) para as demais instâncias do problema.

4 Conclusão e considerações finais

Esse artigo apresentou um estudo de comparação entre o desempenho de alguns algoritmos para o problema do caixeiro viajante. Os resultados mostraram que, de modo geral, encontrar a solução ótima para o TSP utilizando-se apenas da técnica de branch and bound é inviável pois o algoritmo é, além de lento, muito custoso do ponto de vista do consumo de memória. Os algoritmos aproximativos apresentados retornaram soluções condizentes com os seus respectivos fatores de aproximação e conseguiram retornar soluções para todas as instâncias propostas. Porém o artigo carece de mais experimentos, uma quantidade maior de instâncias e sementes testadas serão analisadas para um estudo futuro.

Referências

[1] Christos H. Papadimitriou, "The Euclidean travelling salesman problem is NP-complete", Theoretical Computer Science, Volume 4, Issue 3, 1977.

[2] R. C. Prim, "Shortest connection networks and some generalizations,"in The Bell System Technical Journal, vol. 36, no. 6, pp. 1389-1401, Nov. 1957.

[3] Christofides Nicos,"Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem", 1976.

[4] Edmonds J., Johnson E.L., Lockhart S.C.: Blossom I: A Computer Code for the Matching Problem. IBM T. J. Watson Research Center, Yorktown Heights, New York, 1969.

[5] Python Software Foundation. Python Language reference, verison 3.10.6. Disponível em: http://www.python.org

[6] NetworkX developers. Networkx library reference, version 2.8.8. Disponível em: https://networkx.org/documentation/stable/