Soluções para problemas difíceis
Problema do Caixeiro Viajante

Vinícius Alexandre da Silva

[1]Instituto de Ciências Exatas – Universidade Federal de Minas Gerais (UFMG)
Av. Pres. Antônio Carlos, 6627, Pampulha –
31.270-901 – Belo Horizonte – MG – Brasil

vini-alex@ufmg.br

Resumo. *Nesse trabalho serão feitas comparações entre três algoritmos para solucionar o Problema do Caixeiro Viajante. Em particular serão avaliados um de solução exata, Branch and Bound, e dois aproximados, Twice Around the Tree e de Christofides.*

1. Introdução

O Problema do Caixeiro Viajante[TSP 2024] (em inglês *Traveling Salesman Problem*) é um problema clássico da computação. Nele tenta-se determinar a menor rota entre uma sequência de cidades (visitando cada uma única vez), retornando à cidade de origem.

O PCV é sabido como um problema *NP-Completo*[TSP 2024], isto é, ainda não é conhecida um algoritmo determinístico com solução exata que possa resolvê-lo em tempo polinomial. Portanto diversas tentativas de otimização foram feitas para reduzir seu custo computacional.

2. Conceitos Básicos

A entrada do PCV é um grafo ponderado não-direcionado completo, ou seja, um grafo em que existe arestas não-direcionadas, com pesos que neste caso representa a distância euclidiana entre todos os pares de vértices. A saída é uma lista com a sequência de vértices e o peso total do ciclo[TSP 2024].

3. Algoritmos

Para comparação, foram usados 3 algoritmos.

3.1. Branch and Bound

Branch and Bound[Wiener 2003] (*B&B*, *BB*) é uma técnica com resultados ótimos para resolver problemas NP-difíceis em que se tenta fazer cortes de ramos árvore de busca a fim de reduzir o tempo de processamento em relação à força-bruta.

No caso específico do PCV, o corte pode ser feito se o peso do caminho sendo percorrido já é maior que a estimativa de melhor peso.

Como ele é um algoritmo que retorna a resposta ótima, no pior caso ele tem a mesma complexidade da força-bruta, $O(n!)$.

3.2. Twice Around the Tree

Twice Around the Tree[kartik 2022] utiliza da árvore geradora mínima e da desigualdade triangular como base para calcular uma solução aproximada.

A árvore geradora mínima[kartik 2024] (*MST*) é uma árvore que conecta todos os vértices e possui peso mínimo. A partir da *MST* se faz um caminhamento por todos os vértices até retornar ao início. Por fim remove-se todos as repetições de vértices do caminho para se ter o resultado final.

A desigualdade triangular[kartik 2022] (que se aplica neste caso pelo uso de distâncias euclidianas) diz que tendo vértices i, j, k, $dist(i, k) \leq dist(i, j) + dist(j, k)$. Com isso temos a garantia que o resultado produzido por esse algoritmo é no máximo 2 vezes o resultado ótimo.

Quanto a sua complexidade, usa-se o algoritmo de *Prim* para a *MST*, que possui $O(n^2)$ e $O(n)$ para o caminhamento e criação do ciclo.

3.3. Christofides

O algoritmo de *Christofides*[NetworkX.org 2024] também encontra soluções aproximadas do PCV, em casos de distâncias geométricas (simétricas e obedecem a desigualdade triangular).

Possui um fator de aproximação de $3/2$ da solução ótima, seu nome vem do autor Nicos Christofides, que o publicou em 1976.

O algoritmo pode ser descrito pelo seguinte pseudo-código:

1. Cria-se a *MST*
2. Separe os vértices de grau ímpar
3. Faça o *matching* perfeito dos vértices de grau ímpar
4. Faça a união da *MST* com o *match* criando um Multigrafo
5. Forme um *circuito Euleriano*
6. Remova as repetições criando o *circuito Hamiltoniano*[TSP 2024]

Ele é dominado pela complexidade da etapa de *matching*, ou seja, $O(n^3)$.

4. Implementação

Os experimentos foram feitos numa máquina equipada com AMD Ryzen 5700x (8 cores) de 4 GHz, 32 GB de RAM, Python 3.10 e sistema operacional Ubuntu. O script foi programado para abortar quando o tempo de processamento atinja 30 minutos.

As instâncias de entrada foram baixadas de *TSPLIB*[Heidelberg 2018] e outras foram criadas a partir dos exemplos para se ter uma maior variedade de tamanhos de entrada, tendo 35 grafos com 5 a 1400 vértices.

Devido ao limite de 30 minutos, instâncias do *Branch and Bound* com $|V| > 10$ foram todas abortadas, o que limitou o número de valores reportados. Quanto aos outros dois algoritmos, todas as instâncias finalizaram dentro do limite, com a instância de 1400 vértices demorando um pouco mais de 29 minutos com *Christofides*.

5. Análise Comparativa

5.1. Comparação Temporal

Na (Figure 1), temos os tempos de processamento dos três algoritmos em todas as instâncias válidas, podemos ver a diferença assintótica entre os eles, com o *Branch and Bound* levando cerca de 56 minutos, o que me levou a abortar todas as instâncias maiores.

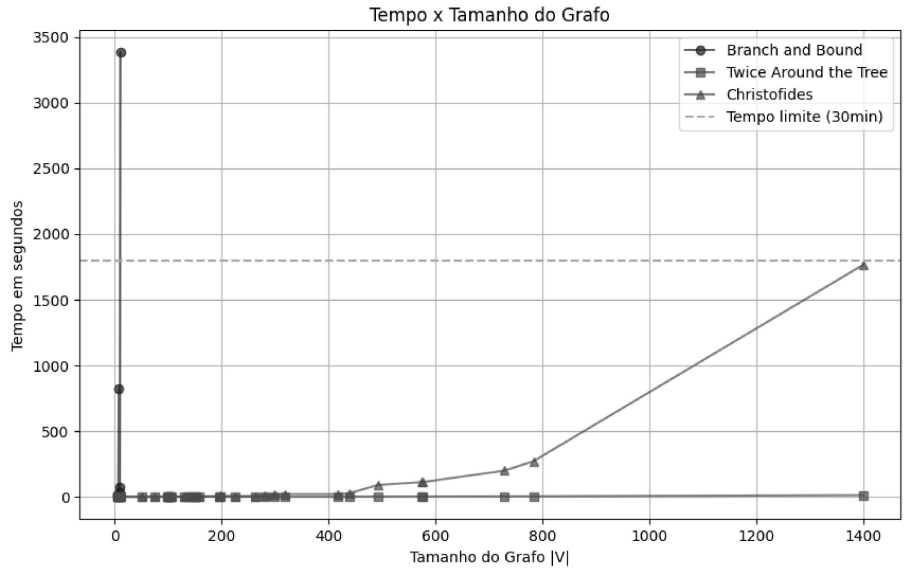

Figure 1. Comparação de tempo de processamento.

Por isso é mais útil mostrar cada algoritmo em separado:

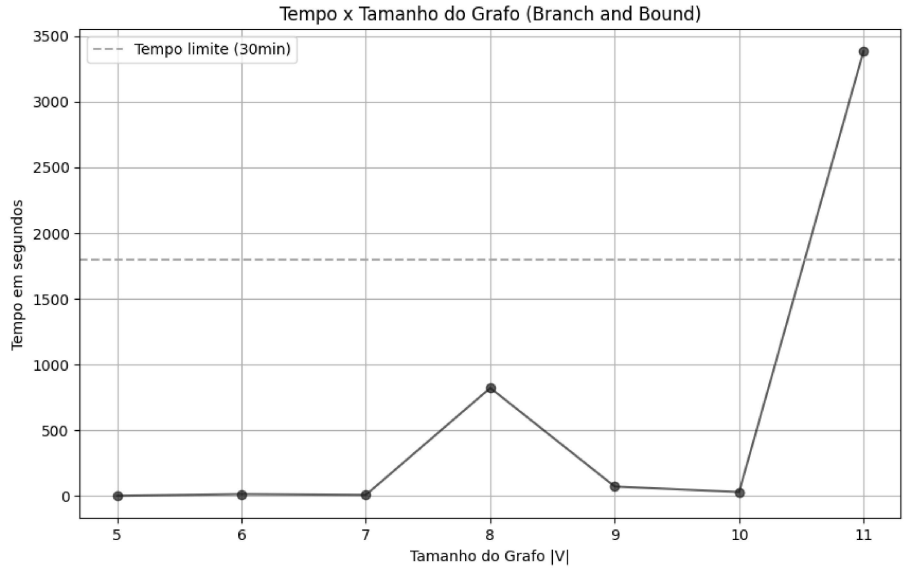

Figure 2. Tempo de processamento para Branch and Bound.

Podemos ver na (Figure 2) que não vemos um constante crescimento seguindo o aumento do tamanho do grafo, mesmo nessa pequena amostra, a variação do tempo é também dependente de como os cortes na árvore de busca são efetivos.

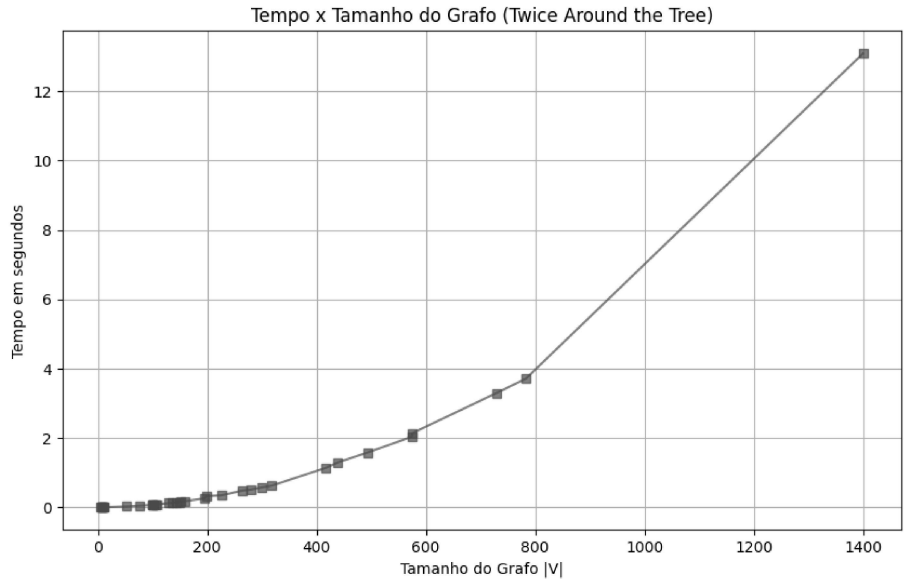

Figure 3. Tempo de processamento para Twice Around the Tree.

Na (Figure 3) o *Twice Around the Tree* foi muito mais rápido que os outros, com tempo máximo de 13 segundos.

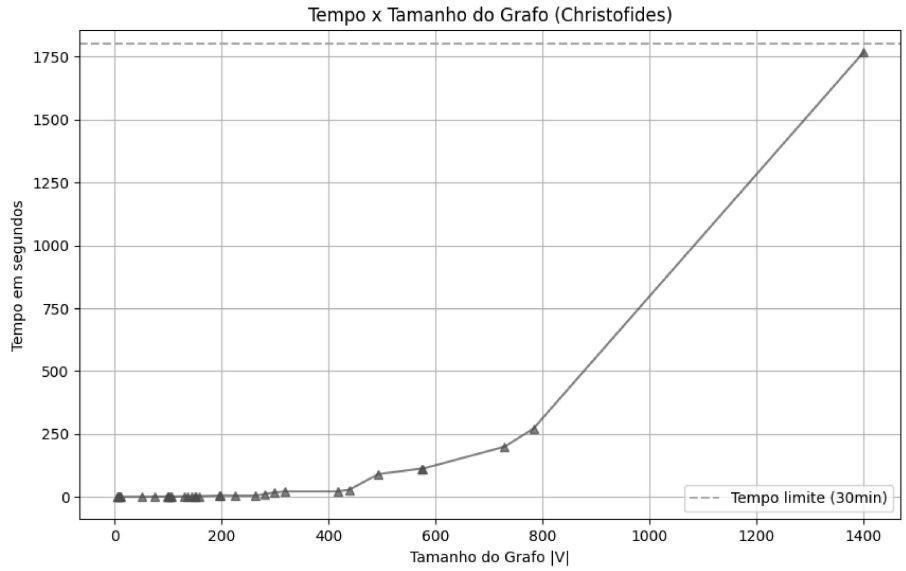

Figure 4. Tempo de processamento para Christofides.

Em fim, temos em (Figure 4) o caso do algoritmo de *Christofides*, todas as instâncias terminaram dentro do limite de 30 minutos, sendo o grafo *fl1400* demorando 1767 segundos. Ambos seguem suas taxas de crescimento conforme suas complexidades, $O(n^2)$ e $O(n^3)$.

5.2. Comparação de Proximidade

Como o *Branch and Bound* lhe dá a resposta ótima, como mostrado na (Table 1) temos a comparação entre os dois algoritmos aproximativos com o resultado exato.

Instância	Ótimo	TAT	Ch
a280	2579	3800	2924
berlin11	4038	4747	4131
berlin52	7542	10114	8560
ch5	1704	1851	1704
ch7	1884	1912	1884
ch9	2265	2501	2568
ch130	6110	8129	6841
ch150	6528	8413	7182
d6	4711	4711	4849
d8	3320	3443	3380
d198	15780	19129	17306
d493	35002	45085	38574
eil10	160	173	172
eil101	629	864	707
fl417	11861	16273	13073
fl1400	20127	28632	22872
kroA100	21282	27210	23293
kroB100	22141	25885	24012
lin105	14379	20176	16487
lin318	42029	59158	47451
pr76	108159	145336	116684
pr107	44303	54237	47906
pr136	96772	151904	103772
pr144	58537	80599	70610
pr152	73682	92018	79224
pr226	80369	115268	92415
pr264	49135	70619	54525
pr299	48191	64572	52998
pr439	107217	146846	120358
rat99	1211	1679	1393
rat195	2323	3395	2657
rat575	6773	9497	7778
rat783	8806	12065	10064
u159	42080	58296	46896
u574	36905	49655	41329
u729	41910	57709	47879

Table 1. Comparação entre o resultado ótimo e os resultados aproximados.

O algoritmo de *Christofides* apresentou melhores resultados ao algortimo *Twice Around the Tree* em praticamente todas as instâncias testadas, para confirmar também mostramos os valores relativos.

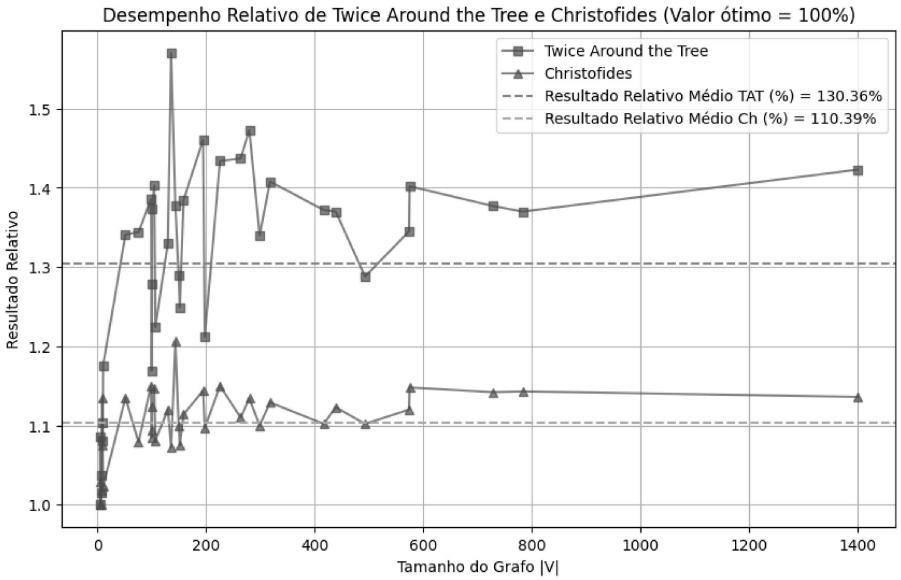

Figure 5. Desempenho Relativo.

Mostrando na (Figure 5) o *Twice Around the Tree* com cerca de 30% acima do resultado ótimo (variação entre 1 e 1.57) e o algoritmo de *Christofides* com 10% acima (variação entre 1 e 1.2) em média.

6. Conclusão

O Problema do Caixeiro Viajante é um problema que embora tenha uma formulação simples, é na realidade bastante complexo. Muitos estudos foram feitos na tentativa de se encontrar resultados que se aproximam o máximo possível de seu melhor resultado sem que haja um grande uso de tempo de computação para que instâncias maiores possam serem resolvidas.

Vimos três formas de se resolver esse problema, com maiores complexidades resultando em melhores resultados ao custo de tempo. Com os algoritmos de *Christofides* e *Twice Around the Tree*, podemos encontrar boas aproximações do resultado ideal usando algoritmos com tempo polinomiais.

Mesmo sendo relativamente simples e usarem conceitos conhecidos como a árvore geradora mínima, pode-se encontrar resultados satisfatórios com uma relativa aproximação (10% a 30% de perda em média).

Embora o *Branch and Bound* ser uma forma de reduzir o custo da árvore de busca em relação à força bruta, ele se mostrou insatisfatório nas instâncias testadas, mostrando um alto custo temporal mesmo com tamanhos de entrada pequenos ($n = 11$ demorando 56 minutos). Outras formas de cálculo do limite para os cortes poderiam melhorar seu desempenho.

References

Heidelberg, R.-K.-U. (2018). Tsp data. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/. Accessed: (10/01/2025).

kartik (2022). Approximate solution for travelling salesman problem using mst. https://www.geeksforgeeks.org/approximate-solution-for-travelling-salesman-problem-using-mst/. Accessed: (06/01/2025).

kartik (2024). Prim's algorithm for minimum spanning tree (mst). https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/. Accessed: (06/01/2025).

NetworkX.org (2024). christofides — networkx 3.4.2 documentation. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.approximation.traveling_salesman.christofides. Accessed: (06/01/2025).

TSP (2024). Travelling salesman problem. https://en.wikipedia.org/wiki/Travelling_salesman_problem. Accessed: (07/01/2025).

Wiener, R. (2003). Branch and bound implementations for the traveling salesperson problem. *JOURNAL OF OBJECT TECHNOLOGY*.