Trabalho Prático 2 - O Problema do Caixeiro Viajante Algoritmos 2

Gabriella de Lima Araujo[1], **Nicolas Von Dolinger M. Rocha**[2]

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

{gabriellalima, nicolasvondolinger}@dcc.ufmg.br

Abstract. *This article analyzes different approaches to solving the Traveling Salesman Problem (TSP), which belongs to the class of NP-hard problems. Specifically, three strategies are discussed: an exact implementation based on the Branch-and-Bound algorithm and two approximate heuristics, Twice Around the Tree and the Christofides algorithm.*

Resumo. *Este artigo analisa diferentes abordagens para resolver o Problema do Caixeiro Viajante (PCV), pertencente à classe de problemas NP-difíceis. Especificamente, são discutidas três estratégias: uma implementação exata baseada no algoritmo Branch-and-Bound e duas heurísticas aproximativas, o Twice Around the Tree e o algoritmo de Christofides.*

1. Introdução

O Problema do Caixeiro Viajante (PCV) é um dos problemas mais importantes na ciência da computação devido à sua relevância prática e à sua complexidade teórica. Pertencente à classe de problemas NP-difíceis, o PCV consiste em encontrar a rota mais curta que visita um conjunto de cidades exatamente uma vez e retorna à cidade de origem. Sua aplicabilidade vai além da teoria, abrangendo áreas como logística, planejamento e redes de comunicação.

Este trabalho aborda os aspectos práticos de algoritmos para a resolução do Problema do Caixeiro Viajante, com foco em instâncias em que a função de custo é determinada pela distância euclidiana no plano cartesiano bidimensional. Três abordagens são exploradas: uma implementação exata baseada no algoritmo Branch-and-Bound e duas heurísticas aproximativas, Twice Around the Tree e o algoritmo de Christofides, que apresentam fatores de aproximação de 2 e 1.5, respectivamente.

O objetivo é promover uma compreensão e reflexões sobre o equilíbrio entre eficiência e exatidão na resolução de problemas desafiadores, com foco especial no Problema do Caixeiro Viajante.

2. Implementação

2.1. Configurações da Máquina

A máquina em que todos os testes foram executados possui o sistema operacional Debian(versão mais recente disponível no momento dos testes), 12 GB de memória RAM e um processador AMD Ryzen 5.

2.2. Escolhas de projeto

O projeto foi desenvolvido em C++ por ser uma linguagem de programação de mais baixo nível em comparação ao Python, a outra opção disponível. Essa característica permite que o C++ ofereça uma execução mais rápida, o que foi particularmente importante devido à natureza NP-difícil do problema abordado.

Foi utilizado o compilador `g++` para a compilação do código, garantindo compatibilidade e desempenho. Além disso, um *Makefile* foi empregado para automatizar o processo de compilação, facilitando a organização do projeto e agilizando a execução de tarefas repetitivas, como a limpeza de arquivos gerados.

2.3. Abordagens implementadas

A implementação de todos os algoritmos assume que o grafo de entrada é completo, ou seja, existe uma aresta entre todos os pares de vértices, e que a função de custo entre os vértices é uma métrica, que satisfaz as seguintes propriedades:

- $c(u, v) \geq 0$
- $c(u, v) = 0$ se e somente se $u = v$
- $c(u, v) = c(v, u)$
- $c(u, v) \leq c(u, w) + c(w, v)$ para todos os vértices u, v, w

Nesta seção, apresentamos as abordagens implementadas para resolver o Problema do Caixeiro Viajante. Dividimos a análise em três subseções, cada uma detalhando uma das abordagens exploradas: a solução exata baseada no algoritmo Branch-and-Bound e as heurísticas aproximativas Twice Around the Tree e o algoritmo de Christofides.

Table 1. Comparação entre os Algoritmos

Algoritmo	Aproximação	Complexidade
Branch-and-Bound	Exato	Exponencial
Twice-Around-the-Tree	2-aproximado	O(n log n)
Christofides	1.5-aproximado	O(n^3)

2.3.1. Branch-and-Bound

O algoritmo Branch-and-Bound é uma abordagem exata utilizada para resolver o Problema do Caixeiro Viajante. Ele explora sistematicamente o espaço de soluções, dividindo-o em subproblemas menores (branch) e utilizando limites inferiores e superiores para eliminar subproblemas que não podem conter a solução ótima (bound), reduzindo assim o esforço computacional necessário.

A implementação deste trabalho considerou uma matriz de adjacência para representar o grafo de entrada, assumindo que o grafo é completo e que os custos das arestas satisfazem as propriedades de uma métrica (não negatividade, simetria e desigualdade triangular). A abordagem foi projetada para minimizar a função de custo total ao visitar todos os vértices exatamente uma vez e retornar ao vértice inicial, formando um circuito hamiltoniano de custo mínimo.

Para reduzir o número de subproblemas explorados, foram implementadas as seguintes estratégias de poda:

- Cálculo de um limite inferior: Um limite inferior inicial foi calculado usando o custo de uma Árvore Geradora Mínima (AGM), similar à utilizada no algoritmo Twice-Around-the-Tree.
- Poda por custo: Subproblemas cujo limite inferior excedesse o custo da melhor solução encontrada até o momento foram descartados.

A exploração das soluções foi realizada por meio de uma busca em profundidade (DFS), que permitiu um controle eficiente da pilha de subproblemas. Apesar de ser computacionalmente intensivo, o Branch-and-Bound garante a obtenção da solução exata para o problema.

2.3.2. Twice-Around-the-Tree

O algoritmo Twice-Around-the-Tree é um algoritmo 2-aproximado, ou seja, a solução obtida é, no máximo, duas vezes pior que a ótima. Ele constrói uma árvore geradora mínima (AGM) a partir da entrada e, neste trabalho, o algoritmo utilizado para gerar a árvore foi o algoritmo de Prim, devido à sua facilidade de implementação e ao fato de lidarmos com entradas de grande escala.

Após a construção da árvore, um algoritmo de busca em profundidade (DFS) é empregado para encontrar um caminho que se inicia em um determinado vértice, passa por todos os outros vértices e retorna ao inicial sem repetir vértices, ou seja, é gerado um circuito hamiltoniano.

2.3.3. Algoritmo de Christofides

O algoritmo de Christofides é um algoritmo 1.5-aproximado para o Problema do Caixeiro Viajante, ou seja, a solução obtida por esse algoritmo é, no máximo, 1.5 vezes pior que a solução ótima. Ele combina conceitos de árvores geradoras mínimas (AGM), emparelhamento mínimo e ciclo hamiltoniano para gerar uma solução de qualidade garantida, sem recorrer a uma busca exaustiva de todas as soluções possíveis.

A implementação deste algoritmo segue a seguinte sequência de etapas:

- Construção da Árvore Geradora Mínima (AGM): Assim como no algoritmo Twice-Around-the-Tree, é construída uma árvore geradora mínima a partir dos vértices do grafo de entrada. Neste trabalho, o algoritmo de Prim foi utilizado para gerar a árvore, devido à sua simplicidade e eficiência em grafos densos.
- Emparelhamento Mínimo dos Vértices de Grau Ímpar: Após a construção da AGM, identificam-se os vértices com grau ímpar na árvore. Esses vértices são então emparelhados de forma ótima, minimizando o custo das arestas que conectam os vértices de grau ímpar, utilizando o algoritmo de emparelhamento mínimo.
- Formação do Ciclo Euleriano: Com a AGM e o emparelhamento mínimo, cria-se um ciclo Euleriano no grafo, onde cada aresta é percorrida exatamente uma vez. Para transformar esse ciclo Euleriano em um circuito hamiltoniano, removem-se as arestas repetidas.
- Otimização da Solução: O ciclo resultante é otimizado por meio de uma busca para encontrar a melhor ordem de visitação dos vértices, garantindo que o circuito final seja o mais curto possível.

Essa abordagem, embora não gere a solução ótima, provê uma boa aproximação com uma garantia de qualidade, tornando-a útil para problemas de grande escala onde a solução exata é inviável.

3. Resultados

Nesta seção, apresentamos e analisamos os resultados obtidos a partir da execução das três abordagens implementadas para resolver o Problema do Caixeiro Viajante Euclidiano. As tabelas apresentam os resultados obtidos para cada algoritmo individualmente, apresentando a resposta obtida, o valor ótimo apresentado pela biblioteca, o tempo necessário para a computação e a memória máxima utilizada.

Foram utilizados diferentes conjuntos de dados, com variadas quantidades de vértices, para testar o desempenho de cada algoritmo. Além disso, foi feita uma comparação entre as abordagens exata e aproximadas, destacando as vantagens e limitações de cada uma em cenários de grande escala.

3.1. Instâncias pequenas

Nesta seção, analisamos o comportamento de cada algoritmo ao lidar com entradas pequenas e que podem ser resolvidas em um curto período de tempo. Nesse sentido, o objetivo é analisar a corretude dos algoritmos e também o comportamento de tempo e custo computacional em relação a entradas pequenas.

Table 2. Branch-and-Bound

Número de Vértices	Ótimo	Resultado	Tempo (s)	Memória Máxima (Mb)
4	80	80	0.000083	1.79688
5	16	16	0.000036	1.82422

Table 3. Twice-Around-the-Tree

Número de Vértices	Ótimo	Resultado	Tempo (s)	Memória Máxima (Mb)
4	80	80	0.00004	1.80469
5	16	19	0.00003	1.83594

Table 4. Algoritmo de Christofides

Número de Vértices	Ótimo	Resultado	Tempo (s)	Memória Máxima (Mb)
4	80	95	0.000016	1.74609
5	16	16	0.000017	1.83594

Analisando os resultados, é possível perceber que o algoritmo ”Branch and Bound” encontra a resposta ótima em tempo hábil em ambas as instâncias. Em relação aos dois algoritmos aproximativos, cada um deles, em uma das instâncias, consegue encontrar a resposta ótima, e em uma das outras instâncias, encontra uma resposta dentro do valor aproximado que cada algoritmo propõe alcançar.

3.2. Instâncias da TSPLIB

Nesta seção, analisamos o comportamento de cada algoritmo ao lidar com instâncias contendo uma quantidade considerável de pontos. As instâncias utilizadas neste trabalho foram obtidas na TSPLIB, uma biblioteca amplamente reconhecida que reúne conjuntos de dados de exemplo para o Problema do Caixeiro Viajante (TSP) e problemas relacionados. A TSPLIB foi desenvolvida e é mantida pela Universität Heidelberg, sendo composta por instâncias provenientes de diversas fontes e de diferentes tipos.

Devido à alta complexidade computacional do algoritmo Branch-and-Bound, não é possível finalizar a execução dentro do limite de tempo estipulado. Por esse motivo, os resultados mostrados para esse algoritmo correspondem aos valores obtidos em 30 minutos de execução, apenas para um pequeno grupo de testes consideravelmente pequenos comparados com outras instâncias. Observa-se que os algoritmos aproximativos abordados demonstraram desempenho significativamente superior em relação ao tempo de execução.

Table 5. Branch-and-Bound

Número de Vértices	Ótimo	Resultado	Tempo (s)	Memória Máxima (Mb)
51	426	1308	1800	1.86719
52	7542	22205	1800	1.84375
70	675	3410	1800	1.82422
99	1211	2124	1800	3.39844

Table 6. Twice-Around-the-Tree

Número de Vértices	Ótimo	Resultado	Tempo (s)	Memória Máxima (Mb)
51	426	587	0.000864	2.44141
52	7542	9873	0.000177	2.44141
70	675	858	0.001729	2.44141
99	1211	1677	0.002824	3.42578
127	118282	153525	0.001563	3.30078
130	6110	7858	0.002909	3.37109
150	6528	9001	0.001986	3.39062
198	15780	17599	0.002290	3.78125
493	35002	43661	0.011936	5.49219
657	48912	64643	0.019961	7.86328
1084	239297	313637	0.056561	17.21480
1748	336556	448834	0.142362	30.66410
1889	316536	455217	0.161728	33.76950
2103	80450	124923	0.240323	45.74610
3038	137694	198079	0.505968	86.14450
3795	28772	41222	0.781931	118.44500
4461	182566	251202	1.174240	173.54300
15112	1573084	2218900	13.427900	1807.89000
18512	645238	893164	19.756200	2688.45000

Table 7. Algoritmo de Christofides

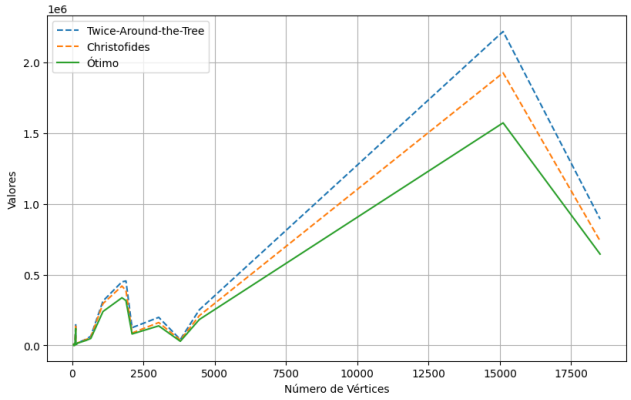
Número de Vértices	Ótimo	Resultado	Tempo (s)	Memória Máxima (Mb)
51	426	569	0.000216	2.44141
52	7542	9355	0.000229	2.44141
70	675	796	0.001519	2.44141
99	1211	1303	0.000810	2.44141
127	118282	144688	0.001141	3.36719
130	6110	7045	0.001161	3.34766
150	6528	7620	0.001538	3.35156
198	15780	17813	0.002457	3.70312
493	35002	40870	0.014760	5.19531
657	48912	58507	0.025569	7.51562
1084	239297	294936	0.070570	16.21090
1748	336556	419230	0.181372	28.89840
1889	316536	387451	0.208453	32.03910
2103	80450	87299	0.240893	45.75780
3038	137694	161295	0.515040	86.14840
3795	28772	36965	0.807484	118.44900
4461	182566	209717	1.189990	173.63300
15112	1573084	1926703	13.916100	1808.22000
18512	645238	738356	20.034700	2688.75000

3.3. Análise dos Resultados

Nessa seção, vamos analisar e discutir os dados coletados durante os testes utilizando as instâncias disponibilizadas pela TSPLIB.

O gráfico abaixo apresenta os dados coletados em relação ao número de vértices de cada instância e a resposta dada por cada algoritmo. Portanto, é possível observar que o algoritmo de Christofides possui uma aproximação mais precisa que o ”Twice-Around-the-Tree”.

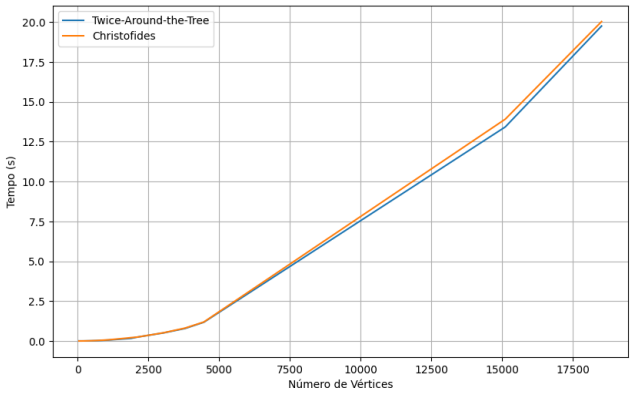
Figure 1. Comparação das soluções dos algoritmos aproximativos.



Nesse sentido, também podemos fazer a análise em relação ao tempo de execução de cada algoritmo para as mesmas instâncias. O algoritmo ”Twice-Around-the-Tree”

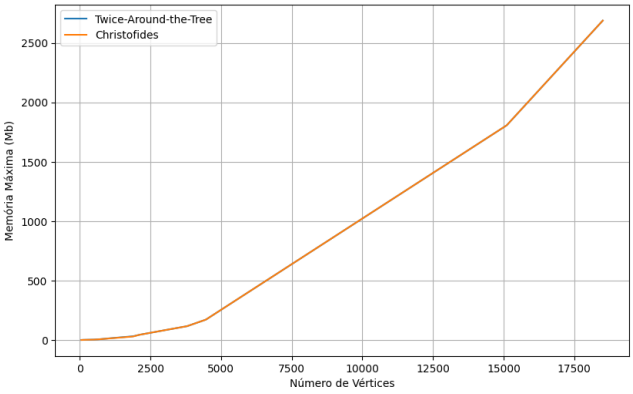
possui um tempo de execução consideravelmente menor que o algoritmo de Christofides, principalmente dado o crescimento do número de vértices de cada instância.

Figure 2. Comparação do tempo de execução dos algoritmos aproximativos.



É possível fazer a análise em relação ao pico de uso de memória por cada algoritmo. Cada algoritmo possui um passo a passo, no entanto, alguns desses passos se repetem, por isso o uso de memória foi muito similar para ambos os algoritmos. É importante notar que o uso de memória para computar cada instância cresce junto com o número de vértices.

Figure 3. Comparação do uso de memória dos algoritmos aproximativos.



Com base nos dados coletados, é possível observar que tanto o algoritmo Twice-Around-The-Tree quanto o algoritmo de Christofides encontram soluções que estão de acordo com suas respectivas propostas de aproximação. Além disso, é possível notar que o algoritmo de Christofides produziu resultados significativamente mais próximos do ótimo em comparação ao algoritmo Twice-Around-The-Tree. No entanto, essa maior precisão veio acompanhada de um custo, já que o tempo de execução e a memória utilizada pelo algoritmo de Christofides é um pouco maior.

Essa diferença pode ser explicada pelo trade-off entre eficiência e precisão ao lidar com heurísticas e algoritmos c-aproximados. Quanto menor o valor de 'c' - ou seja, quanto mais próximo o resultado do algoritmo estiver da solução ótima - maior será o esforço computacional necessário para alcançá-lo, já que estamos tratando de um problema NP-difícil.

4. Conclusão

Nesse sentido, após a realização dos testes, podemos perceber que os algoritmos possuem vantagens e desvantagens diferentes em relação ao PCV. Em instâncias muito pequenas, o algoritmo Branch and Bound consegue entregar a solução ótima em um tempo pequeno. No entanto, a sua aplicação em instâncias com o número de vértices consideravelmente grande, essa abordagem se torna inviável, devido ao tempo e poder computacional que são necessários para obter a resposta ótima, já que o espaço de busca para encontrá-la cresce exponencialmente. Nesse sentido, abordagens utilizando algoritmos aproximativos são necessárias para resolver o problema com uma resposta consideravelmente boa.

O algoritmo "Twice-Around-the-Tree" apresentou o menor tempo de execução em relação às outras abordagens e o mesmo uso de memória do algoritmo de Christofides. No entanto, mesmo com essa vantagem, é importante destacar que as suas respostas para as instâncias podem ser no máximo 2 vezes piores que a resposta ótima. Nesse sentido, o seu uso deve levar em consideração o tempo de execução em detrimento da qualidade da resposta, atentando-se também à sua aplicação.

O algoritmo de Christofides apresentou as melhores respostas em comparação com a resposta ótima, podendo ser no máximo 1.5 vezes pior que a solução ótima apresentada e com o tempo de execução consideravelmente pequeno, apesar de ser maior que o tempo necessário para o "Twice-Around-the-Tree". Nesse sentido, essa abordagem é extremamente interessante, tendo em vista a qualidade da resposta e o tempo de processamento.

Portanto, o trabalho realizado permitiu a implementação de diferentes algoritmos para a resolução de um problema NP-difícil, que apresenta dificuldades inerentes da classe, o que permitiu analisar de forma detalhada as diferentes abordagens e como soluções aproximativas podem ser úteis para solucionar esse tipo de problema.

5. Referências

Abdul Bari. '7.3 Traveling Salesman Problem - Branch and Bound'. Disponível em: https://www.youtube.com/watch?v=1FEP_sNb62k. Acesso em: 24 dez. 2024.

Alon Krymgand. 'The Christofides Algorithm'. Disponível em: https://alon.kr/posts/christofides. Acesso em: 27 dez. 2024.

Universität Heidelberg. TSPLIB. Disponível em: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. Acesso em: 23 dez. 2024.