Desbravando o Caixeiro Viajante: Uma Comparação Entre Branch and Bound, Twice Around e Christofides

Jalmir de Jesus Ferreira da Silva Junior [1]

[1]Departamento de Ciência da Computação, UFMG
Belo Horizonte, MG, Brazil - CEP 31270-901

Resumo. *O Problema do Caixeiro Viajante é um dos problemas clássicos e amplamente estudados em otimização combinatória. Este trabalho tem como objetivo comparar três algoritmos populares para a resolução do problema: Branch and Bound, Twice Around e Christofides. Através de uma análise empírica, com a aplicação desses algoritmos a instâncias clássicas, comparamos os desempenhos em termos de tempo, espaço e qualidade da solução.*

1. Introdução

A busca por soluções ótimas em problemas de roteirização desempenha um papel fundamental em diversas áreas de pesquisa e aplicação prática, como a otimização de redes de transporte, distribuição de recursos e planejamento logístico. Exemplos típicos incluem o planejamento de rotas para veículos de serviços, como ônibus urbanos, ambulâncias e carros de polícia, onde a eficiência no uso de recursos e no tempo de deslocamento é crucial. Um dos desafios mais emblemáticos nesse campo é o **Problema do Caixeiro Viajante (TSP)**, que, apesar de sua formulação simples, apresenta uma complexidade computacional que cresce de maneira exponencial à medida que o número de cidades aumenta. Considerado um problema NP-difícil, o TSP exige o desenvolvimento de abordagens inovadoras e técnicas avançadas de otimização para encontrar soluções eficientes, especialmente quando lidamos com instâncias de grande escala.

Este estudo se dedica à análise comparativa de três algoritmos amplamente utilizados para resolver o TSP: Branch and Bound, Twice Around e Christofides. Cada um desses métodos oferece diferentes *trade-offs* entre precisão e desempenho, e são amplamente empregados em cenários práticos onde a solução exata pode não ser viável devido ao tempo de computação necessário. Ao investigar suas implementações e comparar o desempenho de cada um em diversas instâncias do problema, buscamos avaliar as condições ideais para a aplicação de cada abordagem, além de explorar as limitações e vantagens de cada algoritmo. Este trabalho visa, portanto, não apenas aprofundar o entendimento sobre esses métodos, mas também fornecer uma análise crítica de sua aplicabilidade em contextos reais de otimização.

2. Metodologia

A abordagem adotada para resolver o Problema do Caixeiro Viajante (TSP) envolve a implementação e comparação de três algoritmos: Branch and Bound, Twice Around e Christofides. A seguir, detalhamos as escolhas e decisões técnicas que orientaram a implementação de cada um desses algoritmos, incluindo a escolha das estruturas de dados, as estimativas de custo utilizadas, o tipo de busca aplicada no Branch and Bound e outras considerações relevantes.

2.1. Branch and Bound

O algoritmo **Branch and Bound** explora o espaço de soluções e poda subespaços inviáveis para reduzir a complexidade. Embora garanta soluções ótimas, sua complexidade de tempo cresce exponencialmente com o número de cidades, e o uso de memória também aumenta significativamente, tornando-o viável apenas para instâncias pequenas do problema [Cormen et al. 2009].

2.1.1. Estimativa de custo

O limite inferior inicial é calculado somando as menores arestas associadas a cada vértice. Durante a execução, a função `limite_inferior` refina o custo considerando a matriz de distâncias reduzida. Ela consiste em reduzir os valores da matriz de maneira que os custos de viagens mínimas entre os nós possam ser representados de forma simplificada, sem alterar a essência do problema. A matriz é ajustada para que a menor distância de cada linha e coluna seja subtraída, contribuindo para um valor de redução acumulado.

2.1.2. Estruturas de dados

- **Pilha:** Utilizada para armazenar os nós do espaço de busca. Cada nó contém o custo acumulado, o limite inferior estimado, o caminho atual e o conjunto de vértices não visitados.
- **Conjunto:** Representa os vértices não visitados, otimizando operações de busca e remoção.

2.1.3. Estratégia de busca

A estratégia implementada segue o depth-first search (DFS), onde os nós mais recentemente inseridos na pilha são explorados primeiro. Essa abordagem foi escolhida devido à sua simplicidade de implementação e boa compatibilidade com a poda de soluções inviáveis.

2.1.4. Detalhes da implementação

1. **Poda:** Subespaços são descartados se o limite inferior do nó for maior ou igual ao custo do melhor caminho encontrado.
2. **Ordenação:** Os próximos vértices a serem visitados são ordenados pelo custo de inserção, priorizando os caminhos mais promissores.
3. **Atualizações periódicas:** Estatísticas como número de podas, melhor custo encontrado e tempo decorrido são exibidas a cada 60 segundos para monitorar o progresso.

2.2. Twice Around

O algoritmo Twice Around é uma heurística que gera uma solução com garantia de aproximação de, no máximo, duas vezes o custo ótimo [Christofides 2022]. Ele se baseia na construção de uma árvore geradora mínima (MST), com custo de tempo $O(V^2)$

para matrizes de adjacência. A memória necessária é $O(V^2)$ para matrizes. Após construir a MST, as arestas são duplicadas, e um passeio euleriano é transformado em um ciclo Hamiltoniano usando busca em profundidade em $O(V)$.

2.2.1. Passos da Implementação

1. **Construção da MST:** Utiliza-se o algoritmo de Kruskal ou Prim para encontrar a MST que conecta todas as cidades com o menor custo total, sem formar ciclos.
2. **DFS na MST:** Realiza-se uma busca em profundidade (DFS) na MST para determinar a ordem de visita dos vértices, formando um passeio aproximado.
3. **Fechamento do ciclo:** O passeio obtido é fechado ao retornar ao vértice inicial, criando um ciclo hamiltoniano.
4. **Cálculo do custo:** O custo total do ciclo é calculado somando os pesos das arestas no percurso.

2.2.2. Detalhes da Implementação

- A escolha pela DFS permite que o passeio inclua cada vértice exatamente uma vez, garantindo simplicidade e eficiência.
- A estrutura de dados grafo é representada usando a biblioteca `networkx`, que fornece ferramentas eficientes para manipular grafos e calcular MSTs.

Essa abordagem é simples e rápida, mas, como a garantia é de apenas duas vezes o custo ótimo, essa abordagem pode gerar soluções distantes do ótimo em algumas instâncias.

2.3. Christofides

O algoritmo de Christofides é uma heurística aproximada com uma garantia de aproximação de, no máximo, 1,5 vezes o custo ótimo [Christofides 2022]. Ele combina as ideias de MST e emparelhamento perfeito: a construção da MST tem custo de tempo $O(V^2)$ para matrizes de adjacência. O emparelhamento perfeito em um subconjunto de vértices ímpares é resolvido em $O(V^3)$. A memória necessária é $O(V^2)$. Após combinar as arestas, um passeio euleriano é ajustado para formar um ciclo Hamiltoniano em $O(V)$.

2.3.1. Passos da Implementação:

1. **Construção da MST:** Assim como no Twice Around, utiliza-se uma MST para conectar todas as cidades com o menor custo total.
2. **Identificação de vértices ímpares:** Na MST, os vértices com grau ímpar são identificados, pois esses contribuem para formar ciclos eulerianos.
3. **Emparelhamento perfeito mínimo:** Um subgrafo é construído com os vértices ímpares, e as arestas de custo mínimo são adicionadas para formar um emparelhamento perfeito. Esse passo reduz o grau ímpar dos vértices.
4. **Construção do multigrafo:** Combina-se a MST com o emparelhamento perfeito, resultando em um multigrafo que é euleriano (todos os vértices têm grau par).
5. **Circuito Euleriano:** Um circuito euleriano é encontrado no multigrafo.

6. **Conversão para ciclo hamiltoniano:** O circuito euleriano é transformado em um ciclo hamiltoniano ao remover vértices repetidos no percurso.
7. **Cálculo do custo:** O custo total é calculado somando os pesos das arestas no ciclo hamiltoniano final.

2.3.2. Detalhes da Implementação:

- O uso de emparelhamento perfeito mínimo garante uma aproximação eficiente e está implementado usando a funcionalidade `max_weight_matching` de `networkx`, que considera pesos negativos para encontrar o custo mínimo.
- A transformação de circuito euleriano para hamiltoniano é feita de forma direta ao evitar a visita repetida de vértices.

3. Experimentação

A seção de experimentação visa fornecer uma avaliação empírica dos três algoritmos para resolver o Problema do Caixeiro Viajante (TSP): Branch and Bound, Twice Around e Christofides. Os experimentos foram realizados para comparar o desempenho de cada algoritmo em termos de tempo de execução, qualidade da solução e memória consumida. A seguir, descrevemos as instâncias de teste, os parâmetros utilizados, a avaliação de desempenho e uma análise detalhada dos resultados.

Os experimentos foram feitos na seguinte máquina:

- **Notebook**: Lenovo IdeaPad i3
- **Processador**: AMD Ryzen 5 5500U
- **Memória RAM**: 8 GB
- **Clock**: Base de 2.1 GHz, boost máximo de 4.0 GHz
- **Núcleos/Threads**: 6 núcleos e 12 threads

3.1. Descrição dos Experimentos

3.1.1. Instâncias de Teste

Os testes foram realizados com instâncias do Problema do Caixeiro Viajante de diferentes tamanhos, variando entre 52 e 2319 cidades. As instâncias foram selecionadas a partir de conjuntos de dados de instâncias compiladas da biblioteca TSPLIB [Reinelt 1991], em particular as instâncias cuja função de custo seja a distância euclidiana em 2D.

A lista compreende as seguintes instâncias: *berlin52, ch130, a280, d493, d657, rat783, u1060, d1291, u1817, u2319*

Adicionalmente, para obter um conjunto de dados viável para a realização dos experimentos utilizando o método Branch and Bound, foi empregado um conjunto de instâncias de 6 a 14 cidades. Cada elemento corresponde a uma cidade na Alemanha, com suas respectivas latitudes e longitudes.

3.1.2. Parâmetros dos Experimentos

Os parâmetros utilizados para a execução dos experimentos incluem:

- **Tempo de Execução**: O tempo total gasto para a execução de cada algoritmo, medido em segundos.
- **Memória Consumida**: O consumo de memória durante a execução, medido em megabytes (MB), utilizando ferramentas como tracemalloc para monitoramento de uso de recursos.
- **Custo Final**: O custo da solução final gerada pelo algoritmo, comparado ao valor ótimo quando disponível.
- **Erro Relativo**: A razão entre o custo da solução final gerada pelo algoritmo e o custo ótimo conhecido.

3.2. Avaliacão de Desempenho Branch and Bound

Devido à natureza exponencial do algoritmo Branch and Bound, decidimos não comparar diretamente o seu desempenho com o dos algoritmos heurísticos. O conjunto de testes utilizado é composto por instâncias com um número de cidades variando entre 6 e 14, o que limita a observação de alguns aspectos, como o custo de memória. Nesse contexto, optamos por focar nossa análise no tempo de execução, já que o Branch and Bound garante sempre uma solução ótima. Embora a memória seja um fator importante, ela não será facilmente observada em razão do tamanho relativamente pequeno das entradas utilizadas. Essa abordagem permitirá uma análise mais detalhada do comportamento do algoritmo em termos de eficiência temporal, sem a influência de limitações de hardware ou tamanho de entrada.

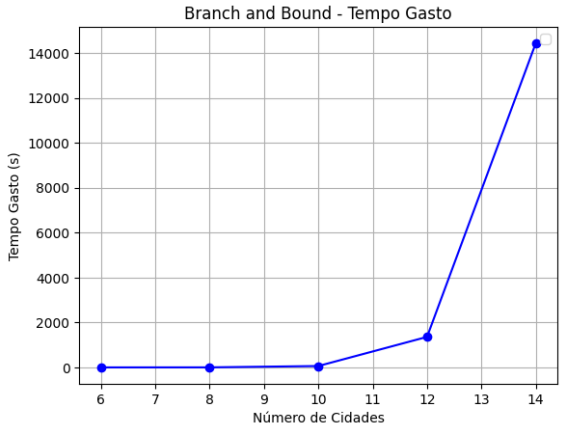

Figure 1. Comparação entre o tempo gasto em relação ao número de cidades

Os resultados apresentados no gráfico reforçam a natureza exponencial do algoritmo Branch and Bound, com o tempo de execução aumentando significativamente à medida que o número de cidades cresce. Embora o algoritmo garanta a solução ótima, sua escalabilidade é limitada para instâncias maiores, conforme esperado. Esses resultados destacam a importância de sua aplicação em problemas de tamanho reduzido ou como referência para avaliação de soluções heurísticas.

3.3. Avaliação de Desempenho entre Twice Around e Christofides

3.3.1. Qualidade da Solução

Para avaliar a qualidade das soluções obtidas pelos algoritmos **Twice Around** e **Christofides**, utilizamos dois gráficos: um de **custo total** (figura 2), que fornece uma

visão geral dos valores absolutos das soluções, e outro de **erro relativo** (figura 3), que destaca a aproximação em relação ao custo ótimo conhecido.

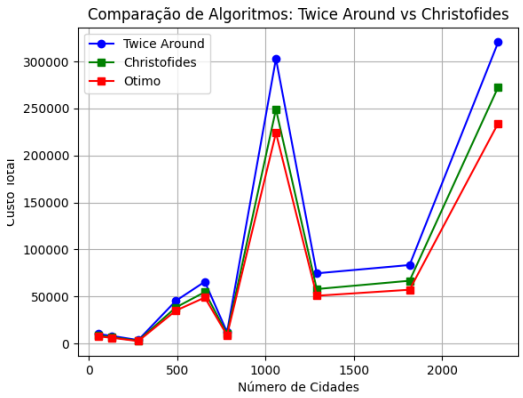

Figure 2. Comparação entre o custo total em relação ao número de cidades

O erro relativo é definido como a razão entre o custo da solução encontrada pelo algoritmo e o custo ótimo da instância. Essa métrica é especialmente útil para comparar a eficiência das heurísticas em gerar soluções de qualidade.

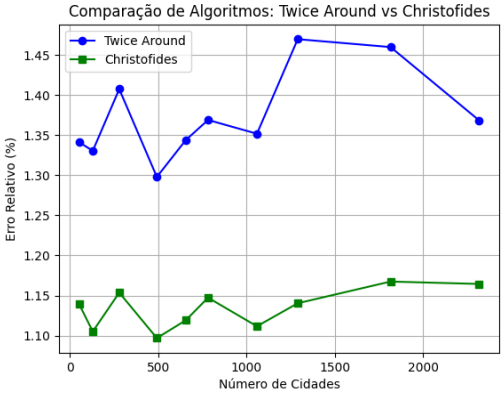

Figure 3. Comparação entre o erro relativo em relação ao número de cidades

O erro relativo do **Twice Around** apresentou uma variação leve, mantendo-se geralmente acima de 1.3 e alcançando um pico em instâncias maiores (1.48 para 1291 cidades). Isso reflete a natureza mais simples do algoritmo. A variação do erro está mais relacionada à estrutura do grafo, já que grafos com arestas de pesos muito variados tendem a penalizar soluções que dependem exclusivamente da MST.

O **Christofides** apresentou erros relativos consistentemente mais baixos, variando entre 1.10 e 1.16. Isso reflete a vantagem da abordagem que combina a MST com o emparelhamento perfeito mínimo, corrigindo as limitações do Twice Around. Mesmo em instâncias maiores, o erro manteve-se dentro da margem de 1.16, comprovando a eficácia do algoritmo para instâncias complexas.

Dessa forma, o Twice Around é mais suscetível a erros maiores em instâncias grandes, especialmente quando a MST é significativamente diferente do grafo com-

pleto. Quanto ao Christofides, embora mais custoso em termos computacionais, entrega soluções mais próximas do ótimo, independentemente do tamanho do problema.

3.4. Tempo Gasto

Os experimentos realizados para avaliar o tempo de execução dos algoritmos Twice Around e Christofides mostram comportamentos distintos conforme o número de cidades aumenta.

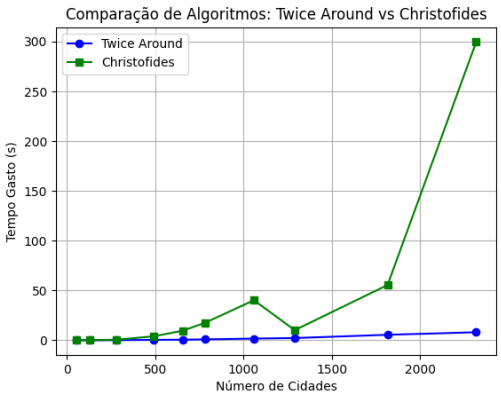

Figure 4. Comparação entre o tempo gasto em relação ao número de cidades

No caso do **Twice Around**, para instâncias pequenas (até 500 cidades), o algoritmo apresentou tempos de execução praticamente instantâneos, demonstrando alta eficiência. Para instâncias maiores, houve um leve aumento no tempo, mas, ainda assim, o algoritmo se manteve altamente eficiente, com tempos de execução não superiores a 10 segundos, mesmo para instâncias com mais de 2.000 cidades.

Já para o **Christofides**, o desempenho inicial do algoritmo foi semelhante ao do Twice Around, com tempos quase instantâneos para instâncias de até 500 cidades. Entretanto, após esse ponto, o Christofides apresentou um aumento mais significativo no tempo de execução, principalmente devido à necessidade de resolver o emparelhamento perfeito mínimo, que é computacionalmente mais custoso. Para 1.000 cidades, o tempo de execução chegou a 50 segundos, e para 2.000 cidades atingiu 300 segundos.

Dessa forma, temos que o Twice Around é claramente mais rápido e escalável para instâncias maiores, tornando-o uma escolha mais adequada quando o tempo de execução é uma prioridade. O Christofides, embora mais lento, oferece melhores garantias de qualidade da solução, justificando o tempo extra para instâncias onde a aproximação mais próxima do ótimo é importante. O aumento acentuado do tempo no Christofides é principalmente devido ao cálculo do emparelhamento perfeito mínimo, uma etapa que não está presente no Twice Around.

3.4.1. Custo de memória

O Twice Around apresentou um crescimento lento e controlado no consumo de memória, mesmo para instâncias maiores. Isso reflete a simplicidade do algoritmo, que utiliza

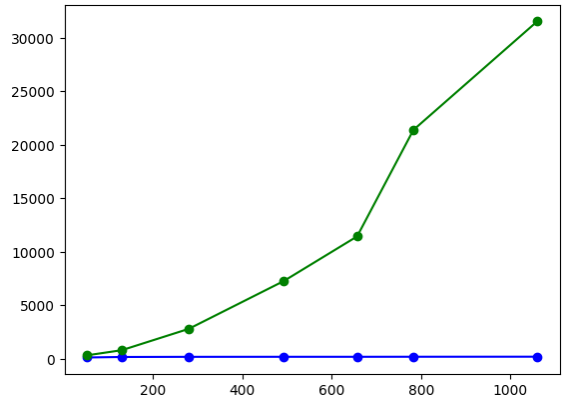

Figure 5. Comparação entre o uso de memória em relação ao número de cidades

operações diretas na estrutura da árvore geradora mínima (MST) e na busca em profundidade (DFS). Essa característica o torna adequado para problemas de grande escala onde o consumo de recursos precisa ser minimizado.

O Christofides, por outro lado, demonstrou um consumo significativamente maior de memória, com uma taxa de crescimento exponencial em relação ao número de cidades. O aumento acentuado deve-se à necessidade de calcular emparelhamentos perfeitos, que envolvem subgrafos completos dos vértices ímpares e operações computacionalmente intensivas para resolver o problema de fluxo.

4. Conclusão

Este trabalho analisou e comparou os algoritmos Branch-and-Bound, Twice Around, e Christofides para o problema do caixeiro viajante (TSP), considerando aspectos como qualidade da solução, consumo de memória e tempo de execução. Os experimentos permitiram identificar os limites e as vantagens de cada abordagem, fornecendo *insights* valiosos sobre a escolha do algoritmo em diferentes contextos.

Algoritmo	**Branch Bound**	**Twice Around**	**Christofides**
Tipo de Solução	Exata	Aproximada	Aproximada
Complexidade	Exponencial	Polinomial	Polinomial
Erro Relativo	N/A	2x	1.5x
Conjunto de Dados	Pequeno	Grande	Médio/Grande

Table 1. Comparação de tipos de solução

Os principais pontos observados foram:

1. **Branch-and-Bound**
   - É uma abordagem exata e, portanto, garante encontrar a solução ótima.
   - No entanto, seu custo computacional exponencial torna-o inviável para instâncias grandes, limitando seu uso a problemas de pequeno porte.
2. **Twice Around**
   - Foi o algoritmo mais rápido entre as heurísticas analisadas, com tempos de execução praticamente instantâneos para instâncias de até 500 cidades e apenas um leve aumento para instâncias maiores.

- Embora eficiente, apresentou erros relativos de cerca de 35% acima do custo ótimo, o que pode ser inadequado para problemas em que a qualidade da solução é crítica.

3. **Christofides**
    - Demonstrou um excelente equilíbrio entre qualidade e desempenho, com erro relativo em torno de 15% acima do custo ótimo, mantendo-se consistente mesmo em instâncias maiores.
    - Apesar disso, teve um crescimento significativo no tempo de execução para instâncias grandes devido ao cálculo do emparelhamento perfeito mínimo.

Com base nos resultados, é possível perceber que a escolha do algoritmo depende diretamente do objetivo e do tamanho do problema:

- Para instâncias pequenas, o Branch-and-Bound é ideal pela garantia de optimalidade.
- Para instâncias maiores onde o tempo de execução é crucial, o Twice Around é uma escolha eficiente, mas pode não ser indicado se a qualidade da solução for uma prioridade.
- O Christofides, embora mais lento, é preferível em cenários que exigem soluções de alta qualidade e podem tolerar um tempo de execução maior.

Os experimentos também destacaram a relação direta entre o aumento do número de cidades e o crescimento do erro relativo e do tempo de execução, especialmente para heurísticas como o Christofides. Essa relação reforça a necessidade de um balanceamento cuidadoso entre precisão e desempenho ao escolher um algoritmo para resolver o TSP.

Essas conclusões oferecem diretrizes práticas para a aplicação dos algoritmos em diferentes cenários, contribuindo para uma melhor compreensão de suas características e limitações.

References

Christofides, N. (2022). Worst-case analysis of a new heuristic for the travelling salesman problem. *SN Operations Research Forum*, 3(1):10.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. MIT Press, Cambridge, MA, 3rd edition.

Reinelt, G. (1991). TSPLIB—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384.