Trabalho Prático 2 – Soluções para problemas difíceis

Victor Henrique Silva Ribeiro[1], **Lucas Sacramento**[2]

[1]Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – Minas Gerais – Brazil

{victor.henrique,lucas.sacramento}@dcc.ufmg.br
{victor.henrique5800,lucas.crsacramento}@gmail.com

Abstract. *In this work, we implemented and analyzed three algorithms to solve the Euclidean Traveling Salesman Problem (TSP): an exact approach based on branch-and-bound, and two approximate solutions, Twice-Around-The-Tree and Christofides. Each algorithm was evaluated considering three aspects: execution time, memory consumption, and solution quality. For this purpose, we utilized instances extracted from the TSPLIB library and compared the obtained results in terms of computational performance and accuracy. The experiments highlighted the trade-offs between accuracy and computational efficiency, emphasizing the conditions under which each approach is most suitable. We conclude that while branch-and-bound is better suited for certain instances due to its exactness, the approximate algorithms are more efficient for other instances, providing solutions of acceptable quality within reduced time.*

Resumo. *Neste trabalho, implementamos e analisamos três algoritmos para resolver o O Problema do Caixeiro Viajante (TSP) na sua forma euclidiana: uma abordagem exata baseada em branch-and-bound, e duas soluções aproximadas, Twice-Around-The-Tree e Christofides. Cada algoritmo foi avaliado considerando três aspectos: tempo de execução, consumo de memória e qualidade das soluções obtidas. Para isso, utilizamos instâncias extraídas da biblioteca TSPLIB e comparamos os resultados obtidos em termos de desempenho computacional e precisão. Os experimentos demonstraram os trade-offs entre exatidão e eficiência computacional, destacando as condições em que cada abordagem é mais adequada. Concluímos que, enquanto o branch-and-bound é mais indicado para certas instâncias devido à sua exatidão, os algoritmos aproximados são mais eficientes para outras, oferecendo soluções de qualidade aceitável em tempo reduzido.*

1. Introdução

O Problema do Caixeiro Viajante (TSP, do inglês *Traveling Salesman Problem*) é um dos desafios mais conhecidos na área de Pesquisa Operacional e Otimização Combinatória. Em sua formulação clássica, o objetivo é encontrar um circuito que visite cada cidade (vértice) exatamente uma vez, retornando à cidade de origem, com custo total mínimo[Lawler et al. 1985]. Apesar de simples em termos conceituais, o TSP é reconhecidamente *NP-Difícil*, o que implica grande complexidade para encontrar soluções exatas em instâncias de tamanho moderado ou grande.

A versão euclidiana do TSP, foco deste trabalho, assume que os custos entre as cidades respeitam a desigualdade triangular, sendo proporcionais às distâncias euclidianas entre pontos em um plano 2D. Esse cenário é bastante comum em aplicações

de roteamento e logística, onde a distância geográfica é o custo principal a ser minimizado[Applegate et al. 2006]. Em contrapartida, mesmo nessa variante mais estruturada, a dificuldade para encontrar soluções exatas permanece alta para grandes instâncias, demandando estratégias que equilibrem exatidão e eficiência computacional.

Neste trabalho, investigamos três métodos para abordar o TSP euclidiano. Dois deles são algoritmos aproximativos: *Twice-Around-The-Tree* e *Christofides*, que fornecem soluções com garantias de aproximação em relação ao custo ótimo. O terceiro é um algoritmo exato, baseado em *branch-and-bound*, que, embora seja capaz de encontrar a rota ótima, pode demandar tempo de execução proibitivo em instâncias maiores. Avaliamos e comparamos essas abordagens em termos de tempo de execução, uso de memória e qualidade das soluções. Para isso, utilizamos um conjunto de instâncias disponibilizadas na biblioteca TSPLIB, que oferece dados de coordenadas para problemas reais ou sintéticos de TSP.

Este artigo está estruturado da seguinte forma. Na Seção 2, revisamos alguns conceitos básicos necessários ao entendimento do TSP e das técnicas de resolução adotadas. Na Seção 3, descrevemos a arquitetura dos experimentos, incluindo as características do ambiente computacional e a forma de coleta de dados. As Seções 4 e 5 apresentam os resultados experimentais e discutem cada um dos algoritmos selecionados, evidenciando vantagens e limitações. Por fim, na Seção 6, concluímos com observações gerais e sugestões de trabalhos futuros.

2. Conceitos Básicos

O Problema do Caixeiro Viajante (TSP) pode ser formalizado como um grafo $G = (V, E)$, em que cada vértice $v \in V$ representa uma cidade, e cada aresta $(u, v) \in E$ possui um custo associado, que, no caso euclidiano, corresponde à distância entre os pontos que representam u e v. O objetivo é encontrar um circuito hamiltoniano (ou seja, um ciclo que visita cada vértice exatamente uma vez) de menor custo possível.

Desigualdade Triangular. Na forma euclidiana, as distâncias respeitam a desigualdade triangular, que estabelece que, para quaisquer três vértices u, v, w, a soma das distâncias $d(u, v) + d(v, w)$ é sempre maior ou igual à distância direta $d(u, w)$. Essa propriedade é fundamental para as garantias de aproximação dos algoritmos *Twice-Around-The-Tree* e Christofides, pois permite estimar limites superiores para o custo de rotas parciais.

Complexidade *NP-Difícil*. O TSP é classificado como um problema *NP-Difícil*, o que implica que não há, até o momento, algoritmos polinomiais conhecidos que o resolvam exatamente para todas as instâncias de forma eficiente. Essa característica justifica o uso de algoritmos aproximativos em diversos cenários práticos, especialmente quando há limitações de tempo ou recursos computacionais.

Branch-and-Bound. O método exato de *branch-and-bound* emprega uma estratégia de busca em árvore, onde cada nó da árvore de busca representa um subconjunto de rotas parciais. Esse algoritmo determina limites inferiores para cada ramo, podando (isto é, descartando) ramos que comprovadamente não conduzem a soluções ótimas[Levitin 2003a].

Embora *branch-and-bound* seja capaz de encontrar a solução exata, a explosão combinatória típica do TSP faz com que o algoritmo seja inviável para instâncias maiores, sobretudo quando há restrições de tempo.

Algoritmos Aproximativos. Como alternativa ao alto custo computacional de métodos exatos, algoritmos aproximativos buscam rotas de boa qualidade, ainda que não necessariamente ótimas, em tempo menor. Duas abordagens populares para o TSP euclidiano são:

- **Twice-Around-The-Tree (2-Approximation):** Constrói-se uma Árvore Geradora Mínima (MST) no grafo euclidiano e, em seguida, percorrem-se suas arestas duplicadas em uma ordem de pré-ordem, ajustando o percurso para obter um circuito hamiltoniano final. A garantia de aproximação é de que o custo da solução não excede o dobro do ótimo[Levitin 2003c].
- **Christofides (1.5-Approximation):** Também inicia com a construção de uma MST, mas adiciona a etapa de emparelhamento perfeito mínimo (*Matching*) dos vértices de grau ímpar. O custo adicional desse *Matching* mantém o fator de aproximação em 1.5[Levitin 2003b], resultando em soluções, em média, mais próximas do ótimo do que o algoritmo *Twice-Around-The-Tree*.

3. Arquitetura dos experimentos

A condução dos experimentos visa mensurar o comportamento de cada algoritmo — *branch-and-bound*, *Twice-Around-The-Tree* e Christofides — diante de diferentes instâncias do TSP euclidiano, ressaltando os aspectos de tempo de execução, uso de memória e qualidade das soluções.

Seleção das Instâncias. Foram escolhidas instâncias do conjunto TSPLIB[Reinelt 1991] cujo número de vértices não excedesse 2000, de modo a viabilizar o armazenamento da matriz de distâncias em disco e na memória principal. O critério de limite de tamanho foi adotado para evitar que a etapa de pré-processamento das distâncias (cálculo da matriz de adjacência) se tornasse inviável em termos de recursos computacionais.

Pré-processamento das Distâncias. Cada instância, representada por um arquivo com as coordenadas dos nós, foi submetida a um pré-processamento para geração da matriz de adjacência completa, contendo os custos entre todos os pares de vértices. Essa matriz foi então armazenada em disco para uso posterior pelos algoritmos, evitando cálculos repetitivos de distância durante a execução de cada método.

Ambiente Computacional. Os experimentos foram executados em uma máquina virtual do tipo `c2-standard-8` na Google Cloud Platform (GCP), equipada com 8 vCPUs e 32 GB de memória RAM. O sistema operacional empregado foi Linux. Esse ambiente foi escolhido por oferecer poder de processamento e memória suficientes para executar as rotinas de busca dentro de um limite de 30 minutos.

Coleta de Métricas. Cada algoritmo foi executado nas mesmas condições, com um limite máximo (*time-out*) de 30 minutos para gerar uma solução. Três métricas principais foram registradas:

1. **Uso de Memória**: Monitoramos o consumo de memória exclusivo das estruturas internas dos algoritmos, desconsiderando o espaço utilizado para carregar a matriz de adjacência.
2. **Tempo de Execução**: Medimos o tempo total empregado até o algoritmo devolver a melhor solução encontrada ou até o estouro do limite de 30 minutos.
3. **Acurácia da Solução**: Avaliamos a qualidade das soluções obtidas comparando seu custo total com o custo ótimo conhecido (*benchmark*) fornecido pela TSPLIB.

Procedimento de Execução. Inicialmente, cada instância era carregada, juntamente com a respectiva matriz de distâncias pré-computada. Em seguida, o algoritmo selecionado (1) iniciava a rotina de busca; (2) caso concluísse antes dos 30 minutos, o tempo de execução e uso de memória eram registrados; (3) caso contrário, forçava-se a interrupção do processo, registrando a melhor solução parcial encontrada até então e indicando *timeout* na métrica de tempo e memória.

Análise Comparativa. Ao final, os resultados foram consolidados e comparados entre si. Para o *branch-and-bound*, foram avaliados somente os desvios (*worse percentage*) em relação ao ótimo, uma vez que o método não finalizou dentro do limite de 30 minutos em nenhuma instância. Para os algoritmos aproximativos, foi possível comparar diretamente o fator de aproximação, o tempo de execução e o consumo de memória em todas as instâncias testadas.

4. Resultados experimentais

4.1. Branch-And-Bound

O algoritmo de *branch-and-bound* foi testado em todas as instâncias selecionadas, porém todas as execuções excederam o limite de 30 minutos (time-out) sem concluir a busca exata. Dessa forma, os resultados apresentados referem-se à melhor solução encontrada antes da interrupção do algoritmo, expressa em termos da métrica de *worse percentage*, isto é, o desvio percentual em relação ao custo ótimo conhecido para cada instância.

A figura 2 sumariza os valores de *worse percentage* para cada instância. Observa-se que não foi possível coletar dados de uso de memória ou de tempo total de execução (pois não houve término do algoritmo), restando como indicador principal de desempenho o quão próxima estava a melhor solução parcial do valor ótimo.

Desempenho Geral. De modo geral, o *branch-and-bound* obteve soluções parciais significativamente diferentes do ótimo em muitas instâncias, sobretudo naquelas em que não há estruturas homogêneas que facilitem a poda do espaço de busca. Por outro lado, algumas instâncias apresentaram resultados melhores quando o problema exibia subestruturas mais claras, permitindo ao algoritmo restringir mais rapidamente as rotas inviáveis. Entretanto, na maior parte dos casos, o *worse percentage* excedeu a faixa de 100%, indicando que a melhor solução encontrada ainda estava muito distante do valor ótimo.

Influência do Número de Nós. A Figura 1 exibe a variação do *worse percentage* em função do número de nós das instâncias. Embora fosse esperado que instâncias maiores levassem a soluções mais distantes do ótimo, não se identificou correlação direta entre o número de nós e o desvio percentual. Essa falta de padrão reforça que o tempo de execução limitado (30 minutos) e a natureza do problema (que pode ou não apresentar subestruturas homogêneas) afetam de forma não uniforme o resultado final.

Observações Específicas. Durante os testes, constatou-se:

- **Subestruturas homogêneas ajudam na poda**: instâncias com regiões bem definidas ou *clusters* de nós apresentaram melhor desempenho, pois o algoritmo conseguiu eliminar rapidamente ramos inviáveis e focar em subconjuntos promissores.
- **Falta de subestruturas homogêneas degrada o desempenho**: em instâncias como `vm1084_tsp`, observada na Figura 4, onde os nós parecem espalhados de forma mais díspar, o algoritmo sofreu para encontrar soluções competitivas antes do *time-out*, resultando em desvios acima de 2000%.
- **Ausência de correlação forte com o número de nós**: mesmo instâncias menores, mas com topologia complexa, podem apresentar *worse percentage* elevado. Já instâncias maiores, mas com agrupamentos de nós homogêneos, acabam tendo desvios relativamente menores.

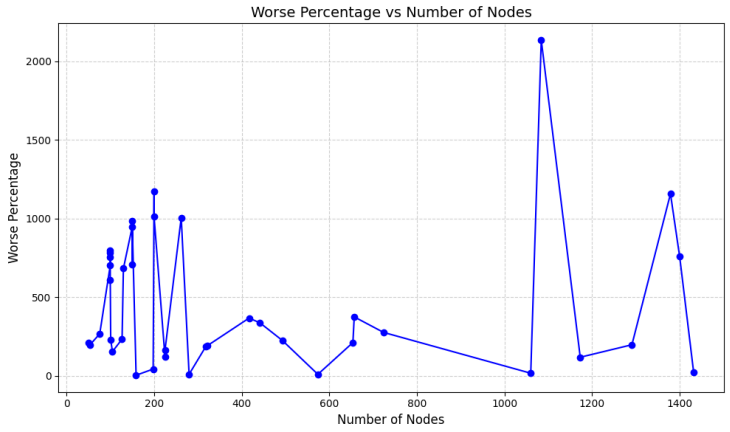

Figure 1. *Worse percentage* em função do número de nós para as instâncias avaliadas via *branch-and-bound*.

Exemplos de Instâncias com Melhor e Pior Desempenho. Nas Figuras 3 e 4 apresentamos dois exemplos extremos. Em `u1432_tsp` (Figura 3), o desvio encontrado foi relativamente menor, apontando para a existência de subestruturas mais favoráveis ao *branch-and-bound*. Já em `vm1084_tsp` (Figura 4), o resultado foi um dos piores, o que sugere pouca homogeneidade geográfica entre os nós e, consequentemente, menor eficiência da poda.

Conclui-se, portanto, que o uso de *branch-and-bound* para TSP, embora teoricamente capaz de prover soluções exatas, não foi viável na prática para instâncias de médio e grande porte sob a restrição de 30 minutos de tempo de execução. O aspecto estrutural

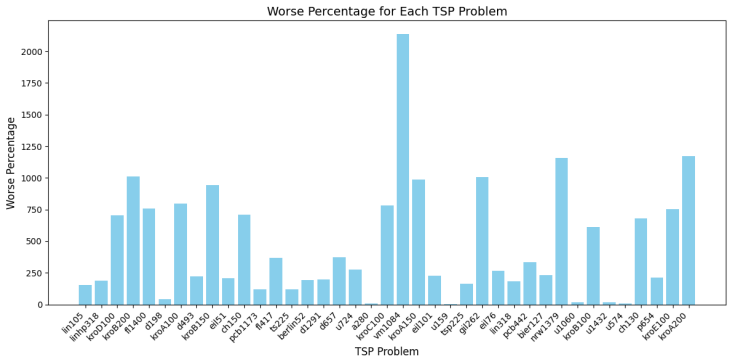

Figure 2. *Worse percentage* de cada instância TSP, considerando a melhor solução encontrada antes do *time-out*.

Figure 3. Instância `u1432_tsp`, uma das que apresentaram o menor *worse percentage*.

de cada instância (mais ou menos homogênea) exerce papel decisivo na capacidade de poda do algoritmo, havendo pouca ou nenhuma relação direta entre o número de nós e a qualidade das melhores soluções encontradas.

4.2. Algoritmos aproximativos

Introdução aos Algoritmos Avaliados. Dois algoritmos aproximativos foram avaliados: o *Twice-Around-The-Tree* e o algoritmo de Christofides. Ambos são heurísticas que exploram estruturas do problema de forma eficiente, mas apresentam diferenças significativas em termos de precisão, tempo de execução e consumo de memória. O algoritmo de Christofides, sendo 1.5-aproximativo, tem garantia teórica de que a solução não será mais que 50% superior ao ótimo. Já o *Twice-Around-The-Tree*, embora seja 2-aproximativo, é mais simples e geralmente mais rápido.

Acurácia da Solução. Como esperado, o algoritmo de Christofides apresentou melhor desempenho geral em termos de custo, conforme mostrado na Figura 5. Sua natureza 1.5-aproximativa garante soluções mais próximas do custo ótimo, enquanto o *Twice-Around-The-Tree* apresentou desvios maiores, especialmente em instâncias de maior porte. Ape-

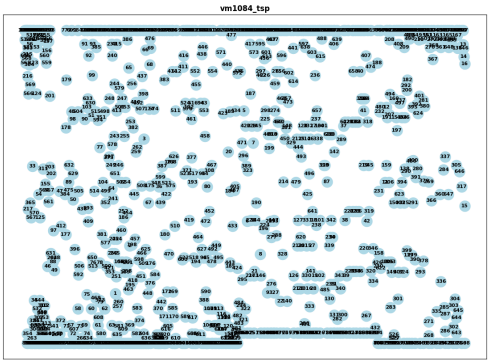

Figure 4. Instância `vm1084_tsp`, que obteve um dos maiores valores de *worse percentage*.

sar disso, para instâncias pequenas ou com topologias simples, a diferença entre os dois algoritmos foi marginal.

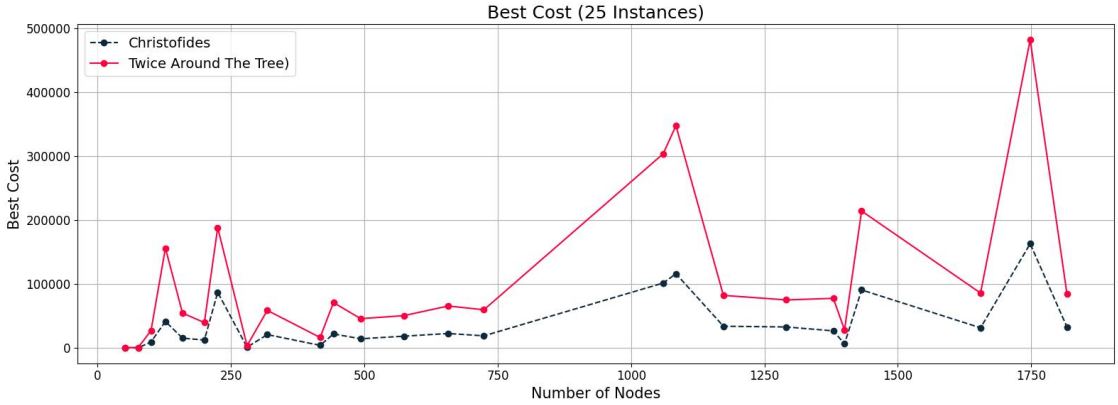

Figure 5. *Best Cost* para as instâncias avaliadas.

Uso de Memória. A Figura 6 demonstra que ambos os algoritmos têm consumo de memória semelhante. Apesar de alguns experimentos apontarem consumo ligeiramente maior para o *Twice-Around-The-Tree*, no caso médio, o consumo se manteve em um limiar próximo. Isso reforça que, embora mais elaborado, o algoritmo de Christofides não exige recursos computacionais significativamente superiores para instâncias de tamanhos moderados.

Tempo de Execução. O algoritmo de Christofides apresentou maior tempo de execução, como indicado na Figura 7. Isso se deve principalmente à etapa de cálculo do *Matching*, cujo impacto aumenta com o número de nós [Christofides 1976]. Por outro lado, o *Twice-Around-The-Tree*, sendo mais simples, demonstrou desempenho consistentemente mais rápido, tornando-se mais adequado para aplicações em tempo real ou em sistemas com restrições severas de processamento.

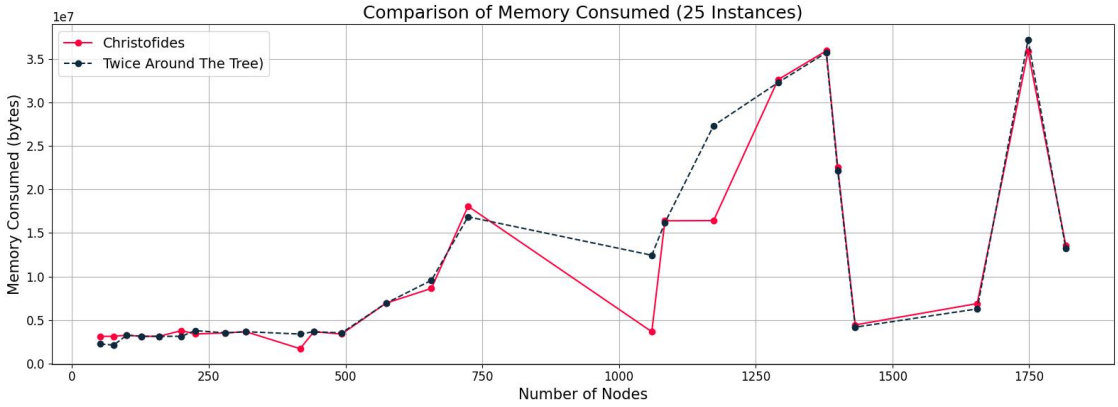

Figure 6. Memória em bytes consumida para cada instância.

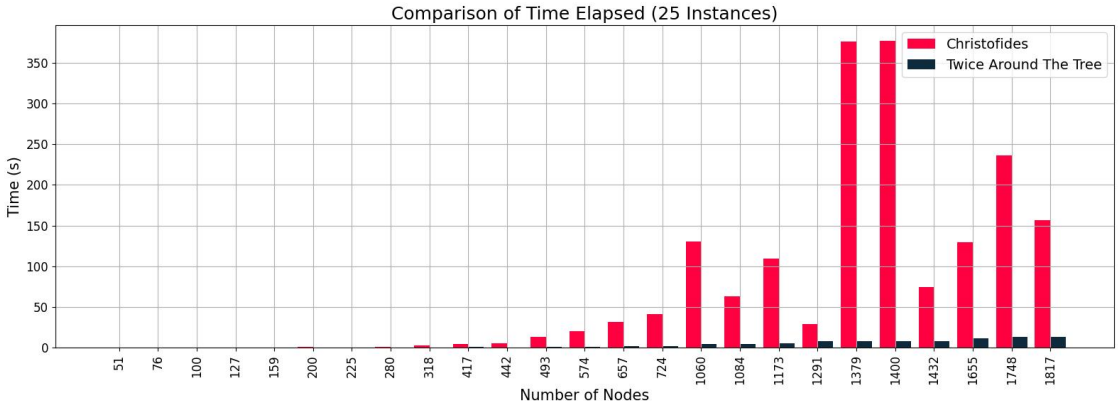

Figure 7. Tempo de execução por instância.

Fator de Aproximação. A Figura 8 destaca que o algoritmo de Christofides tem maior acurácia, com soluções consistentemente mais próximas do ótimo. No entanto, a simplicidade e eficiência do *Twice-Around-The-Tree* justificam seu uso em cenários onde precisão não é o único critério relevante.

Cenários de Aplicação. Em problemas que demandam maior precisão e permitem maior tempo de processamento, o algoritmo de Christofides é claramente superior. No entanto, para aplicações que requerem respostas rápidas e toleram soluções subótimas, o *Twice-Around-The-Tree* apresenta uma alternativa eficiente e prática. A escolha entre os dois métodos deve, portanto, considerar não apenas o tamanho e a complexidade do problema, mas também as restrições específicas de tempo e recursos do sistema onde serão implementados.

5. Conclusão

Os experimentos confirmam que o algoritmo Christofides se destaca em termos de precisão, oferecendo soluções mais próximas do ótimo, embora exija maior tempo de execução e consumo de memória, especialmente em grafos grandes. Por outro lado, o *Twice-Around-The-Tree* é mais eficiente computacionalmente, mas apresenta maior

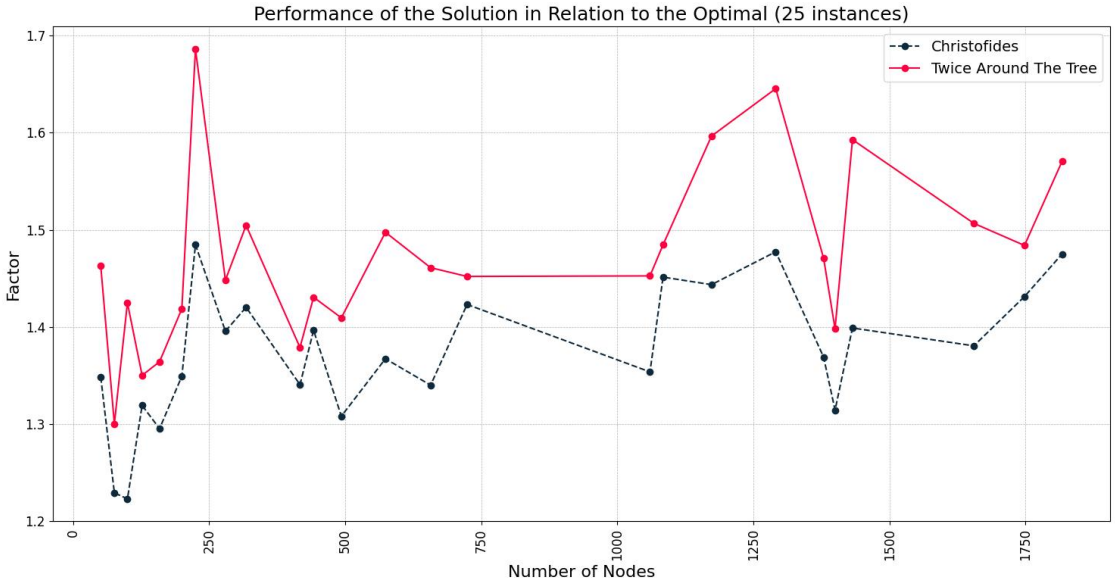

Figure 8. Fator de aproximação do ótimo por instância.

desvio em relação ao custo ótimo. Ambos os algoritmos possuem características que podem ser decisivas dependendo do contexto e das restrições do problema a ser resolvido.

References

Applegate, D. L., Bixby, R. E., Chvátal, V., and Cook, W. J. (2006). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B. (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons.

Levitin, A. (2003a). Branch-and-bound. In *Introduction to the design analysis of algorithms / Anany Levitin. — 3rd ed.*, pages 432–438. Pearson; 3rd edition (September 29, 2011).

Levitin, A. (2003b). Christofides. In *Introduction to the design analysis of algorithms / Anany Levitin. — 3rd ed.*, pages 448–450. Pearson; 3rd edition (September 29, 2011).

Levitin, A. (2003c). Twice-around-the-tree. In *Introduction to the design analysis of algorithms / Anany Levitin. — 3rd ed.*, pages 446–448. Pearson; 3rd edition (September 29, 2011).

Reinelt, G. (1991). TSPLIB95. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`. Accessed: January 16, 2025.