Trabalho Prático de Algoritmos 2

Lorenzo Carneiro Magalhães[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas gerais (UFMG)
Belo Horizonte – MG – Brazil

Abstract. *The research focused on exact and approximate solutions for the Traveling Salesman Problem using key algorithms. Due to the problem's complexity and scaling issues, it emphasized strategies for large instances where exact solutions were impractical.*

Resumo. *O trabalho buscou explorar soluções exatas e aproximadas para o problema do Caixeiro Viajante, mais especificamente, foram implementado os algoritmos Twice-around-the-tree, Christofides, e branch-and-bound. Em conformidade com a dificuldade do problema, em muitos casos fornecer uma solução exata de uma instância grande foi impraticável, já que o escalonamento do problema ocorre rapidamente. Assim, devido às restrições de tempo e de recurso computacional, algumas estratégias foram adotadas para mitigar o problema, além da conclusão da impossibilidade do fornecimento de respostas exatas para instâncias grandes.*

1. Introduction

O Problema do Caixeiro Viajante ou Traveling Salesman Problem (TSP) é um desafio clássico na computação, destacando-se por sua aplicabilidade em diversas áreas, como logística, planejamento de rotas e otimização de redes. Este trabalho prático tem como objetivo explorar as estratégias de solução do PCV, reconhecendo sua importância não apenas como um problema matemático, mas também como uma ferramenta para a eficiência e economia em sistemas de transporte e distribuição. Ao abordar tanto aspectos teóricos quanto práticos, visamos proporcionar uma compreensão aprofundada do problema, destacando a relevância de suas soluções no contexto real e empresarial.

A ideia inicial do trabalho foi desenvolver os algoritmos assim como apresentados em classe. Essa abordagem não foi ruim e garantiu resultados satisfatórios, mas alguns algoritmos foram aprimorados a fim de obter melhor resultado e eficiência. Os algoritmos serão descritos com mais detalhamento nas seções específicas.

Para o teste dos algoritmos, foram usados alguns dos arquivos fornecidos, isso porque muitos deles contavam com uma quantidade massiva de vértices, o que na prática leva a um tempo longo de execução. Ademais, alguns arquivos tinham formatações diferentes, de maneira que fosse mais conveniente selecionar as que tinham a mesma formatação.

2. Implementação

O foco das implementações foi a precisão da respostas. Assim, o custo computacional foi maior em algumas funções. Essa estratégia foi adotada pois ao longo do trabalho foi perceptível a incapacidade de execução do algoritmo para muitos nós devido à restrição de poder computacional e tempo. Assim, para viabilizar uma boa resposta para os problemas, foi dado um refinamento nos algoritmos baseados em heurísticas

2.1. Branch and Bound

O algoritmo de Branch and Bound foi implementado para resolver o Problema do Caixeiro Viajante (TSP) de maneira ótima. Este algoritmo explora o espaço de soluções de forma estratégica, eliminando subconjuntos que não podem conter uma solução ótima. A implementação se baseia em uma estrutura de heap de mínimo para manter um controle eficiente das possíveis melhores soluções a serem exploradas.

A função branch_and_bound(G) recebe um grafo G como entrada, onde cada aresta tem um peso associado. A estratégia central é construir caminhos progressivamente, avaliando-os com base no custo total do caminho e em uma estimativa de custo inferior para a conclusão do caminho. A estimativa do limite inferior é calculada pela função lower_bound(G, path), que soma o custo do caminho atual com a estimativa do custo mínimo para visitar os nós restantes. O algoritmo expande o caminho com o menor limite inferior em cada etapa, garantindo assim que a solução ótima seja encontrada.

Embora o algoritmo em questão apresente resultados eficazes, seu custo computacional é uma preocupação significativa, marcado por uma complexidade de $O(n!)$. Essa característica o torna ineficiente para aplicações rotineiras, exigindo uma capacidade computacional considerável. Além disso, a eficiência das soluções aproximadas fornecidas por outros algoritmos é geralmente suficiente para a maioria das aplicações práticas, diminuindo a necessidade de recorrer a este algoritmo, exceto em casos onde uma solução ótima é absolutamente indispensável.

2.2. Twice Around the Tree

O algoritmo "Twice Around the Tree" é uma heurística para o TSP que começa construindo uma Árvore Geradora Mínima (MST) do grafo de entrada usando a biblioteca NetworkX. O ciclo Hamiltoniano é aproximado realizando um percurso de profundidade na MST, começando de um nó arbitrário e repetindo o percurso, resultando em uma visita dupla a cada aresta da MST.

O ciclo resultante é então simplificado removendo visitas repetidas aos nós, mantendo a primeira ocorrência de cada nó e adicionando o nó inicial ao final para formar um ciclo. O custo total do ciclo é calculado somando os pesos das arestas no ciclo Hamiltoniano.

Assim, como resultado tem-se o custo computacional é de $O(m \cdot logn)$, de maneira que o algoritmo execute muito rapidamente, conforme será mostrado na seção resultados.

2.3. Algoritmo de Christofides

O algoritmo de Christofides oferece uma solução aproximada para o TSP com uma garantia de que o custo não é mais do que 1,5 vezes o ótimo. A implementação começa construindo uma Árvore Geradora Mínima (MST) do grafo de entrada. Em seguida, identifica-se os nós de grau ímpar na MST e realiza-se um emparelhamento mínimo desses nós.

A união do emparelhamento mínimo com a MST resulta em um multigrafo, no qual um Circuito Euleriano é encontrado. Este circuito é transformado em um Circuito Hamiltoniano, removendo visitas repetidas aos nós. Além disso, o algoritmo implementado aplica então a heurística 2-opt para otimizar o circuito, trocando segmentos do circuito para reduzir o custo total em detrimento de mais custo computacional. Por fim, o algoritmo calcula o custo total do circuito, somando os pesos das arestas percorridas.

Esse algoritmo foi implementado com foco em eficiência e precisão, adaptando-se às características específicas do TSP e aproveitando as funções otimizadas oferecidas pelo NetworkX. No entando o custo do algoritmo é $O(n^3)$, mas executa rapidamente. O grande porém é a execução com busca local, que gera um grande custo adicional.

3. Método

Ao longo do trabalho foram feitos diversos experimentos para o teste desses algoritmos. A primeira abordagem foi o teste manual para uma instância de 52 vértices. Essa primeira etapa foi importante pois clareou a inviabilidade de execução do branch and bound para instâncias com muitos vértices a fim de se obter uma solução ótima. Nesse experimento, a memória do computador acabava em pouco tempo, além de demorar muito para explorar todos os caminhos gerados.

Tendo isso em mente, duas ideias surgiram: mudar o branch and bound para adotar uma estratégia de exploração em DFS e manter o código executando por muito tempo ou manter a abordagem atual de busca mais rápida porém consumindo mais memória e desistir da execução de instâncias com muitos vértices. Como havia a restrição de tempo, optei pela segunda opção.

Após essas experiências, percebi o valor dos algoritmos baseados em heurísticas. A partir disso, realizei testes com esses algoritmos da seguinte maneira: foram separadas 10 instâncias arbitrárias de tamanho menor do que 600 vértices e usando-as, foram executados os algoritmos baseados em heurísticas (twice around the treem, algoritmo de christofides e algortitmo de christofides com busca local). Com os resultados prontos, foram comparados as soluções ótimas com as obtidas.

4. Resultados

Os resultados obtidos demonstram uma melhoria significativa na eficácia do algoritmo de Christofides após a implementação de uma estratégia de busca local. Essa otimização resultou em um desempenho notavelmente aprimorado, embora tenha ocasionado um aumento no tempo de execução. Apesar disso, o algoritmo manteve-se viável, diferenciando-se positivamente do algoritmo de branch and bound em termos de tempo de processamento.

É crucial mencionar que o algoritmo de branch and bound não foi integrado aos algoritmos aproximativos devido à sua extensa duração de execução. Esta característica restringe sua aplicabilidade a problemas de menor escala, ao contrário dos algoritmos heurísticos, que exibem uma execução eficiente mesmo em situações com um grande número de vértices.

A tabela a seguir apresenta os resultados experimentais. Os dados são expressos em termos de erro relativo em comparação ao custo ótimo, conforme indicado na fonte dos dados. As colunas são definidas da seguinte forma: 'error_tw' representa o erro relativo do algoritmo "twice around the tree", 'error_chr_w' indica o erro relativo do algoritmo de Christofides sem busca local, 'error_chr' refere-se ao erro relativo do algoritmo de Christofides com busca local, 'time_first_part' mostra o tempo de execução para os algoritmos "twice around the tree" e Christofides sem busca local, e 'time_ls' denota o tempo de execução do algoritmo de Christofides com busca local.

error_tw	error_chr_w	error_chr	time_first_part	time_ls
34.29%	30.76%	5.93%	0.310	9357.918
34.13%	20.73%	4.79%	0.004	0.883
34.12%	19.32%	6.42%	0.016	27.126
18.42%	9.88%	0.95%	0.001	0.007
16.89%	20.85%	5.47%	0.010	12.355
38.36%	28.14%	7.49%	0.022	61.785
34.79%	19.42%	3.91%	0.010	10.239
38.24%	21.28%	5.23%	0.010	9.816
35.60%	21.68%	8.90%	0.010	10.977
38.35%	17.07%	6.94%	0.104	784.964
13.54%	6.41%	0.10%	0.001	0.009
14.06%	4.78%	0.56%	0.001	0.036

Para uma visão mais clara dos erros, o gráfico de box plot apresentado na **figura 1** compara o erro relativo dos três algoritmos, oferecendo uma análise visual clara da variabilidade e precisão de cada um. A estratégia "Twice around the tree" mostra uma variação de erro considerável com uma mediana alta, sugerindo uma inconsistência na precisão. Por outro lado, "Christofides w/o Is" (sem busca local) revela uma precisão melhorada com uma mediana mais baixa, mas ainda assim apresenta uma gama de erros e outliers que indicam variações pontuais significativas.

Em contraste, o método "Christofides" (com busca local) demonstra uma mediana mais baixa e a menor dispersão de erros entre as três abordagens, sem outliers visíveis, indicando uma consistência e precisão superiores. Este método, portanto, se destaca como o mais estável e confiável para minimizar o erro relativo, conforme demonstrado pela concentração de resultados em torno de uma linha de erro mais baixa, sugerindo sua eficácia em uma variedade de condições.

Por fim, para uma análise geral, também tem-se os resultados gerais na **figura 2**, contendo o custo de cada algoritmo em cada instância. Este gráfico de barras ilustra uma comparação dos custos associados a diferentes algoritmos ao resolver instâncias do problema do caixeiro viajante (TSP), com os custos representados em uma escala logarítmica. As barras representam os algoritmos e o custo ótimo, distribuídos por uma série de experimentos realizados de 0 a 11 no eixo X.

Observa-se que, em cada experimento, os custos variam entre os algoritmos, mas mantêm uma proximidade relativa, refletindo um desempenho semelhante em termos de custo logarítmico. O algoritmo ótimo serve como um benchmark de referência, permitindo avaliar o quão próximo os outros algoritmos chegam ao custo ideal. Em uma inspeção visual, percebe-se uma tendência de os algoritmos Christofides com e sem busca local apresentarem custos consistentemente próximos ao ótimo, enquanto "Twice around the tree" mostra um desvio ligeiramente maior em certos experimentos, assim como é mostrado no gráfico anterior.

5. Conclusão

Em conclusão, a análise dos algoritmos aplicados ao problema do caixeiro viajante, conforme representado nos gráficos discutidos, sugere que a seleção de uma estratégia de

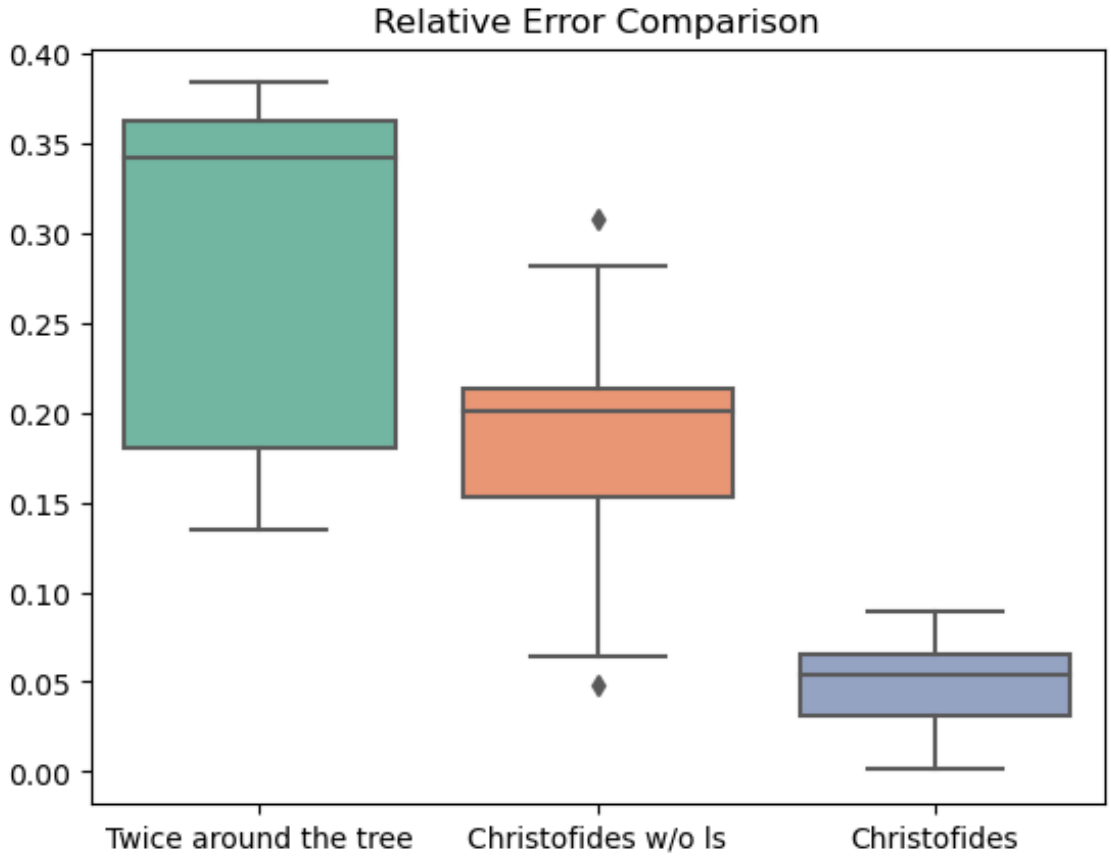

Figure 1. Boxplot do erro relativo de cada algoritmo

otimização ideal deve levar em conta um equilíbrio entre precisão e eficiência computacional. O algoritmo de Christofides, tanto na sua forma pura quanto modificada sem busca local, demonstrou uma capacidade consistente de produzir soluções com baixo erro relativo e custos próximos ao ótimo, conforme ilustrado nos box plots e gráficos de barras. Em contraste, o método Twice around the tree, apesar de sua utilidade em certos contextos, apresentou maior variabilidade nos resultados, sugerindo uma aplicabilidade mais limitada quando a precisão é de suma importância.

Importante notar, o algoritmo ”Branch and Bound”, que não foi representado visualmente nos gráficos mas foi discutido no trabalho, embora forneça soluções de qualidade ótima, enfrenta desafios significativos em termos de tempo de execução, especialmente à medida que a dimensão do problema aumenta. Essa característica o torna menos viável para aplicação em problemas de grande escala ou em situações que exigem soluções em tempo real.

Portanto, conclui-se que, para a resolução prática do problema do caixeiro viajante, os algoritmos baseados no algoritmo de Christofides são recomendados devido ao seu desempenho robusto e eficiente. Eles oferecem um compromisso favorável entre tempo de execução e proximidade com a solução ótima, alinhando-se assim às necessidades de aplicações operacionais onde o tempo é um recurso valioso. Em contrapartida, o Branch and Bound, apesar de sua precisão teórica, pode não ser a melhor escolha para a implementação prática devido à sua demanda computacional elevada. Este estudo reforça

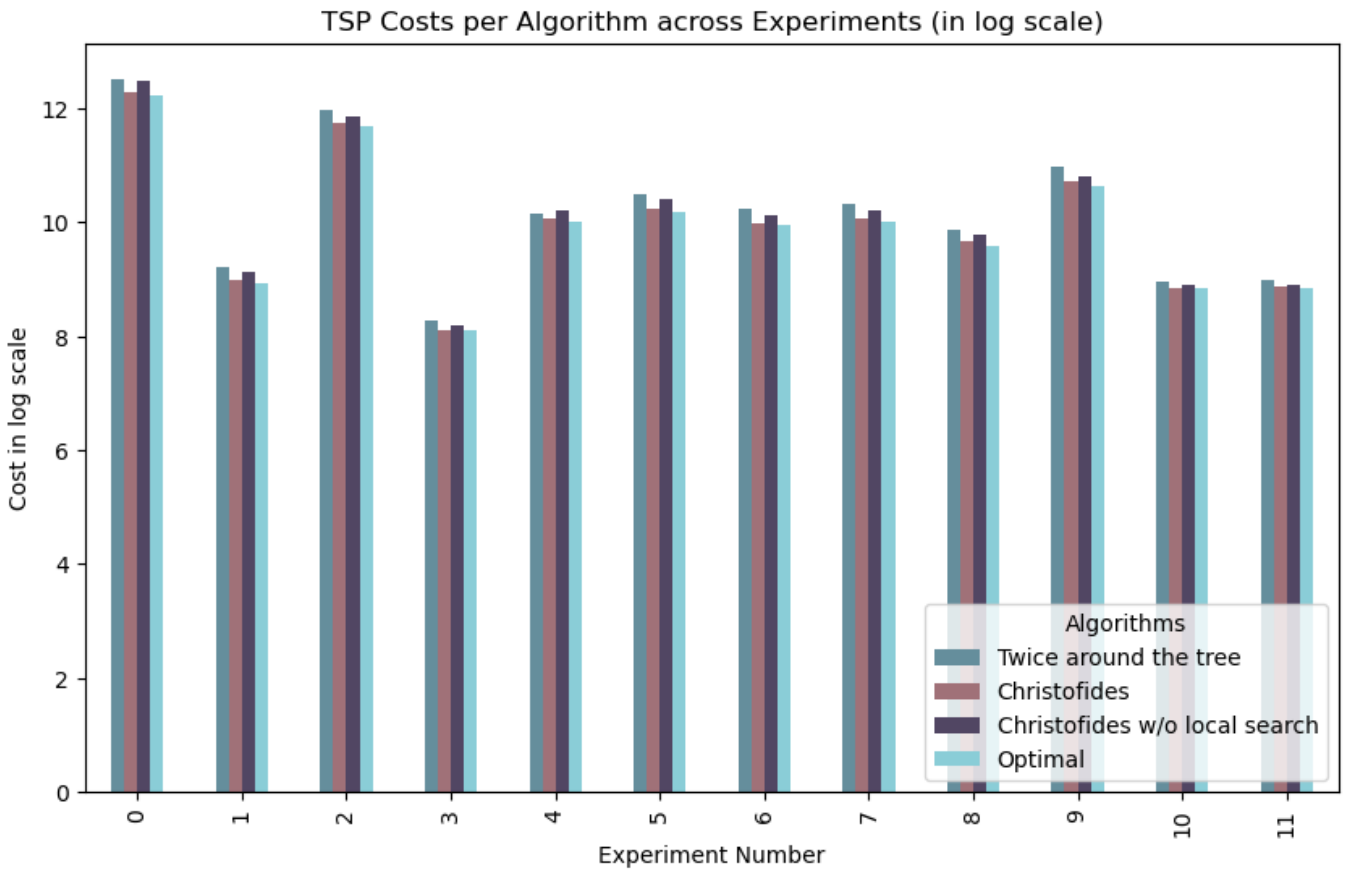

Figure 2. Custos absolutos na escala log dos algoritmos nas instâncias de teste.

a importância de considerar múltiplas métricas de desempenho ao avaliar algoritmos de otimização, e destaca a necessidade de um equilíbrio cuidadoso entre a qualidade da solução e a viabilidade computacional.

References