Relatório TP2-ALG2

Matheus Grandinetti

11 de dezembro de 2023

1 Introduction

Essa documentação lida com a resolução do problema do caixeiro viajante em três diferentes algoritmos, branch and bound, twice around the tree e cristofides. Vamos comparar os resultados obtidos com instâncias próprias e instâncias do site comopt em destaque na especificação do TP.

2 Implementação dos algoritmos

2.1 Informações gerais

Foi utilizada uma implementação própria de grafos, a classe 'Graph', utilizando o método da lista de adjacências. Essa classe é utilizada nos três algoritmos, fazendo as adaptações necessárias para o funcionamento específico deles cada um deles durante o desenvolvimento.

2.2 Branch and bound

Para a implementação desse algortimo foi criada a classe TspBB. Criamos uma árvore de resultados, testando os limiares correntes. Caso o limiar encontrado for pior que o limiar atual, paramos a exploração daquele ramo, partindo para o próximo desdobramento. Dessa forma, para expandir a árvore, utilizamos o conceito de 'best-first', para tentar eliminar o maior número possível de ramos explorados. Tanto o custo temporal quanto o custo espacial são exponenciais, o que pode se tornar inviável em grandes entradas de dados. O benefício desse método é que ele nos fornece uma solução exata para o problema.

2.3 Twice around the tree

Para a implementação desse algoritmo foi criada a classe TspTATT. O algoritmo possui duas principais fases: a construção de uma árvore geradora mínima e o percorrimento da mesma por meio da DFS. Tanto o código de árvore geradora quanto o de busca em profundidade foram implementados do zero, o primeiro na própria classe TspTATT e o segundo na classe Graph. O TATT possui aproximação de até duas vezes, custo assintótico $O(E \log V)$ e custo espacial $O(E+V)$, onde V é o número de vértices e E é o número de arestas do grafo.

2.4 Cristofides

Para a implementação desse algoritmo foi criada a classe TspC. O algoritmo possui quatro principais fases: a construção de uma árvore geradora mínima, a busca por nós de grau ímpar e o seu pareamento mínimo, a criação de um multigrafo com as arestas das fases anteriores e o encontro de um ciclo euleriano no grafo resultante. O código de árvore geradora foi implementado do zero, já o de circuito euleriano e o de nós ímpares foram implementados com a ajuda da biblioteca networksx. O Cristofides possui aproximação de até 1,5 vezes, custo assintótico $O(V^3)$ e custo espacial $O(V^2)$, onde V é o número de vértices e E é o número de arestas do grafo.

3 Análise dos algoritmos

3.1 Resultados encontrados

Na análise de soluções, confirmamos que o algoritmo de Cristofides é melhor em sua aproximação quando comparado com o Twice around the tree, possuindo soluções que são consideralvemente próximas dos resultados exatos. É importante ressaltar que os resultados com '*' na frente não foram obtidos pelo programa por exceder o tempo de 30 minutos especificado. Eles foram obtidos no site que disponibiliza as entradas, já citado anteriormente.

Arquivo de teste	BB	TATT	Cristofides
test4.tsp	205,1179	231,9578	205,1179
test5.tsp	221,7104	257,9184	240,3944
test6.tsp	271,4534	271,4534	295,9717
test7.tsp	264,4458	314,5210	289,1365
test8.tsp	263,7805	344,8444	269,4531
berlin52.tsp	7.542,0000*	10.403,8603	8.642,3312
st70.tsp	675,0000*	861,7650	756,2285
kroA100.tsp	21.282,0000*	30.516,9417	24.026,3606
rd400.tsp	15.281,0000*	20.978,2274	17.371,9015
pr1002.tsp	259.045,0000*	351.430,9838	285.071,9702

Tabela 1: Solução ótima encontrada.

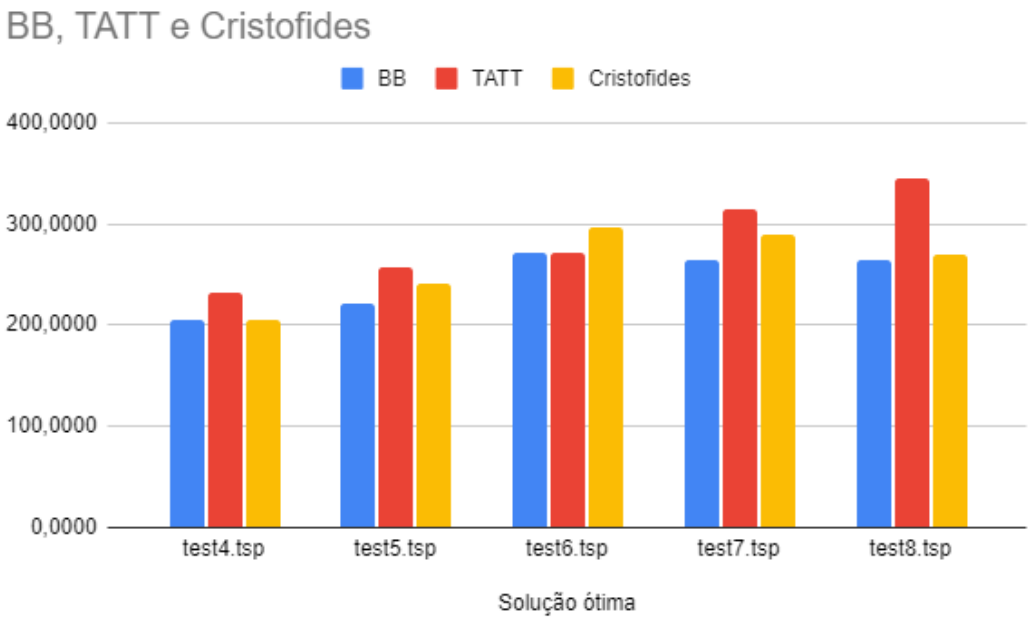

Figura 1: Gráfico das soluções encontradas (arquivos pequenos).

3.2 Tempo de execução

Na análise temporal, percebemos que o melhor algoritmo é o Twice around the tree. Para instâncias muito pequenas, o algoritmo de Cristofides perde até para o branch and bound em sí, porém em uma escala de tempo bem pequena. É importante reassaltar que para instâncias maiores o branch and bound não conseguiu executar as operações em menos de 30 minutos, estourando o tempo limite.

Arquivo de teste	BB	TATT	Cristofides
test4.tsp	0,00004435	0,00011826	0,00084829
test5.tsp	0,00009537	0,00010848	0,00109839
test6.tsp	0,00035286	0,00009441	0,00215411
test7.tsp	0,00056911	0,00009227	0,00194263
test8.tsp	0,00090098	0,00012255	0,00211477
berlin52.tsp	30+	0,00067043	0,01112628
st70.tsp	30+	0,00193191	0,02761388
kroA100.tsp	30+	0,00270677	0,05644965
rd400.tsp	30+	0,03563428	2,57207727
pr1002.tsp	30+	0,21239161	22,93631839

Tabela 2: Tempo de execução encontrado.

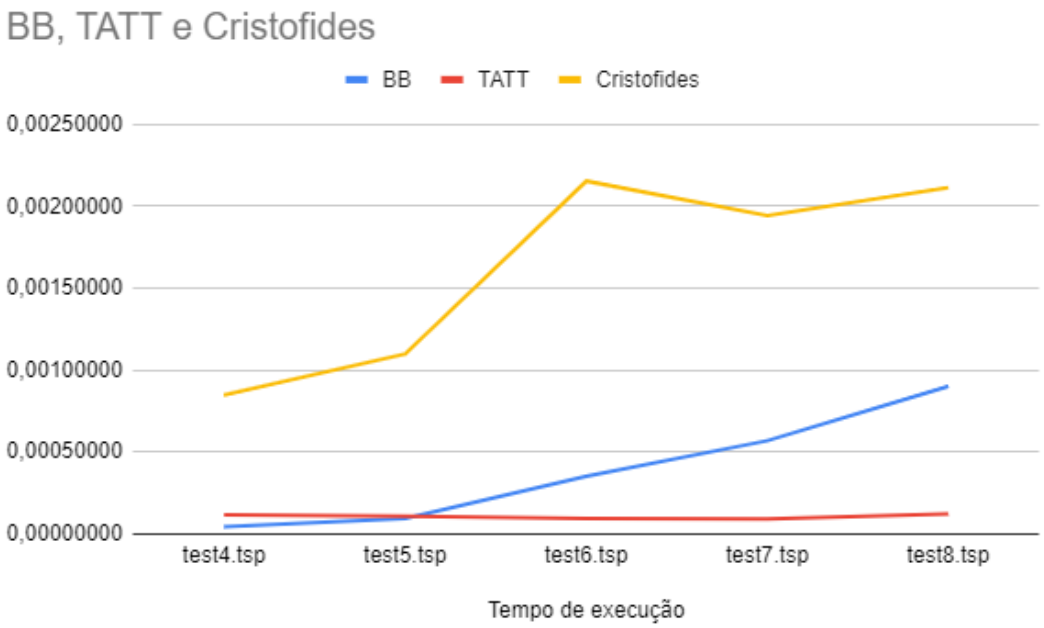

Figura 2: Gráfico dos tempos de execução encontradas (arquivos pequenos).

4 Conclusões

Se a prioridade do projeto é a obtenção de resultados mais próximos do ótimo quanto possível, o algoritmo de Cristofides é recomendado, pois, como verificado na análise de resultados, as soluções encontradas são mais próximas do real. Caso a prioridade do projeto seja a economia de tempo, o algoritmo Twice around the tree é recomendado, pois segundo a análise de resultados, o tempo de execução é em média menor do que os outros dois analisados.