Trabalho Prático 2
Algoritmos 2

Camila Silva Alves[1]

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

camila.alves@dcc.ufmg.br

1. Introdução

O problema do caixeiro viajante é um problema np-difícil que consiste em: dado um conjunto de cidades, determinar o caminho com a menor distância, começando numa cidade qualquer, entre várias, visitando cada cidade precisamente uma vez e regressando à cidade inicial.

2. Objetivo

Este trabalho tem como objetivo implementar algoritmos vistos em sala de aula na disciplina de Algoritmos 2, para solucionar problemas difíceis. Assim, foram implementados os algoritmos *branch-and-bound*, *twice-around-the-tree* e algoritmo de *Christofides* para computação de rotas no problema do caixeiro viajante. Dessa forma, espera-se entender as dificuldades inerentes da implementação dos algoritmos citados.

3. Metodologia e métodos

O trabalho consistiu em implementar os três métodos citados na linguagem de programação python e analisar os resultados, comparando o desempenho computacional e o resultado dos três algoritmos. Para isso, foram utilizadas as bibliotecas *csv, timetsplib95, mathnetworkx,* e *heapq* para a implementação do código dos algoritmos. A biblioteca timetsplib95 foi utlizada para acesso aos dataset para o problema do caxeiro viajante.

Ambos os algoritmos de Christofides e Twice-around-the-tree utilizam arvore geradora mínima para conseguir aproximar da solução real do problema.

4. Experimentos e Resultados

Os resultados dos experimentos estão apresentados na Tabela 1, onde é destacada a relação entre o limiar e o custo de cada algoritmo. O Algoritmo Twice-around-the-tree é uma aproximação de 2, enquanto o algoritmo de Christofides é aproximadamente $\frac{3}{2}$. Nota-se que o algoritmo de Christofides se aproxima mais do limiar. No entanto, é importante observar que o Twice-around-the-tree demonstra maior eficiência em termos de velocidade, pois evita a execução de operações adicionais, como a busca de um conjunto de vértices de nível ímpar na árvore geradora mínima e a identificação do casamento perfeito de custo mínimo nesse novo conjunto, características presentes no algoritmo de Christofides.

5. Conclusão

Embora o algoritmo *branch-and-bound* forneça resultados exatos, sua eficiência é comprometida em instâncias extensas do problema do caixeiro viajante. Em contrapartida, os algoritmos *twice-around-the-tree* e de Christofides demonstraram uma execução rápida para tais cenários, sendo o último especialmente eficaz. Esses resultados destacam a importância de algoritmos aproximativos, que, embora não garantam soluções ótimas, oferecem respostas satisfatórias em um tempo computacional razoável, tornando-se escolhas vantajosas para problemas de grande escala.

References

[1] NetworkX Documentation. Christofides Algorithm. Disponível em: https://networkx.org/documentation/stable/_modules/networkx/algorithms/approximation/traveling_salesman.html#christofides. Acesso em 09/12/2023.

[2] GitHub - Christofides Algorithm Implementation. Disponível em: https://github.com/dsrahul30/Christofides/blob/master/Christofides/christofides.py. Acesso em 09/12/2023.

[3] NetworkX Documentation. Disponível em: https://networkx.org/documentation/stable/index.html. Acesso em 09/12/2023.

[4] JavaTpoint. Traveling Salesperson Problem using Branch and Bound. Disponível em: https://www.javatpoint.com/traveling-salesperson-problem-using-branch-and-bound. Acesso em 09/12/2023.

[5] CodeCrucks. Travelling Salesman Problem Solved using Branch and Bound. Disponível em: https://codecrucks.com/travelling-salesman-problem-solved-using-branch-and-bound/. Acesso em 09/12/2023.

[6] Notas de aula de Algoritmos 2 - Professor Renato Vimieiro.

Table 1. Relação entre custo e tempo entre cada algoritmo

Instance	Algorithm	Nodes	Limiar	Cost	Time
eil51	TAT	51	426	584	0.003
eil51	CHR	51	426	462	0.014
eil51	BAB	51	426	-	NA
berlin52	TAT	52	7542	10114	0.003
berlin52	CHR	52	7542	8560	0.014
berlin52	BAB	52	7542	-	NA
st70	TAT	70	675	888	0.005
st70	CHR	70	675	771	0.027
st70	BAB	70	675	-	NA
eil76	TAT	76	538	696	0.006
eil76	CHR	76	538	608	0.038
eil76	BAB	76	538	-	NA
pr76	TAT	76	108159	145336	0.006
pr76	CHR	76	108159	116684	0.024
pr76	BAB	76	108159	-	NA
rat99	TAT	99	1211	1693	0.009
rat99	CHR	99	1211	1393	0.057
rat99	BAB	99	1211	-	NA
kroA100	TAT	100	21282	27210	0.01
kroA100	CHR	100	21282	23293	0.069
kroA100	BAB	100	21282	-	NA
kroB100	TAT	100	22141	25885	0.011
kroB100	CHR	100	22141	24012	0.05
kroB100	BAB	100	22141	-	NA
kroC100	TAT	100	20749	27968	0.01
kroC100	CHR	100	20749	22752	0.061
kroC100	BAB	100	20749	-	NA
kroD100	TAT	100	21294	27112	0.011
kroD100	CHR	100	21294	23781	0.066
kroD100	BAB	100	21294	-	NA

kroE100	TAT	100	22068	29965	0.011
kroE100	CHR	100	22068	23819	0.077
kroE100	BAB	100	22068	-	NA
rd100	TAT	100	7910	10790	0.009
rd100	CHR	100	7910	8906	0.1
rd100	BAB	100	7910	-	NA
eil101	TAT	101	629	830	0.01
eil101	CHR	101	629	707	0.075
eil101	BAB	101	629	-	NA
lin105	TAT	105	14379	19495	0.011
lin105	CHR	105	14379	16487	0.066
lin105	BAB	105	14379	-	NA
pr107	TAT	107	44303	54237	0.011
pr107	CHR	107	44303	47906	0.051
pr107	BAB	107	44303	-	NA
pr124	TAT	124	59030	74139	0.014
pr124	CHR	124	59030	64668	0.067
pr124	BAB	124	59030	-	NA
bier127	TAT	127	118282	158626	0.015
bier127	CHR	127	118282	133311	0.171
bier127	BAB	127	118282	-	NA
ch130	TAT	130	6110	8129	0.016
ch130	CHR	130	6110	6841	0.0961
ch130	BAB	130	6110	-	NA
pr136	TAT	136	96772	151904	0.037
pr136	CHR	136	96772	103772	0.046
pr136	BAB	136	96772	-	NA
pr144	TAT	144	58537	80599	0.02
pr144	CHR	144	58537	70610	0.0497
pr144	BAB	144	58537	-	NA