Trabalho Prático 2 - Algoritmos II

Marcelo Augusto Mrad Marteleto

December 11, 2023

1 Introdução

O problema do Caixeiro Viajante (PCV) é um desafio clássico na área de otimização combinatória, com amplas aplicações em logística, planejamento de rotas e redes de comunicação. Este trabalho apresenta a implementação e análise de desempenho de três algoritmos distintos para abordar o PCV: *Twice Around the Tree*, *Christofides* e *Branch and Bound*.

O algoritmo *Twice Around the Tree* utiliza uma abordagem baseada em Árvore Geradora Mínima (AGM) para construir um ciclo Hamiltoniano. Por outro lado, o *Christofides* oferece uma solução aproximada eficiente, combinando AGM, emparelhamento perfeito mínimo e ajustes cuidadosos para obter uma solução com garantias de desempenho.

Além disso, exploramos o *Branch and Bound*, uma técnica de otimização que emprega estratégias inteligentes de busca para resolver problemas de busca exaustiva de maneira eficiente.

Este documento detalha a implementação de cada algoritmo, realiza uma análise de complexidade temporal e espacial, apresenta resultados obtidos experimentalmente e conclui com reflexões sobre o desempenho e possíveis melhorias.

2 Implementações

Nesta seção, descreveremos detalhadamente as implementações dos algoritmos utilizados para resolver o Problema do Caixeiro Viajante (PCV) e para fazer a análise dos resultados.

2.1 Twice Around the Tree

O algoritmo *Twice Around the Tree* emprega uma abordagem que inicia convertendo a matriz de adjacência do grafo em um grafo do NetworkX. Em seguida, calcula a Árvore Geradora Mínima (AGM) usando o algoritmo de Kruskal, garantindo conectividade mínima. A partir da AGM, realiza uma busca em profundidade (DFS) para obter uma travessia pré-ordem e, finalmente, forma o ciclo Hamiltoniano.

2.2 Christofides Algorithm

O algoritmo de Christofides adota uma abordagem mais complexa. Utiliza a AGM como base, identifica nós com grau ímpar na AGM e forma um subgrafo induzido apenas por esses nós. Em seguida, encontra um emparelhamento perfeito mínimo nesse subgrafo. Combinando a AGM, o emparelhamento e ajustes cuidadosos, o algoritmo gera um circuito Euleriano, do qual um ciclo Hamiltoniano é construído.

2.3 Branch and Bound

O algoritmo *Branch and Bound* utiliza uma abordagem de busca exaustiva otimizada. Implementado em Python, o código usa uma estrutura de dados especializada (`NoDeBusca`) para representar os estados na árvore de busca. O critério de prioridade é determinado pelo limite atual e custo acumulado. O algoritmo continua a explorar ramos promissores até encontrar a solução ótima.

2.4 Detalhes Compartilhados

Todos os algoritmos são implementados em Python, fazendo uso da biblioteca *NetworkX* para operações relacionadas a grafos e *NumPy* para manipulação eficiente de matrizes. Além disso, foi criado um script para rodar todas as instâncias fornecidas e testar rigorosamente cada algoritmo, para isso foi feito o uso da biblioteca *suprocess* para monitorar cada execução. Para a análise dos resultados foi utilizado a biblioteca *pandas* que gerou os gráficos presentes no relatório.

3 Resultados

3.1 Algoritmo *Twice Around the Tree*

Na **figura** 1, podemos ver a tabela de instâncias executadas pelo algoritmo em menos de 30 minutos (limite máximo imposto), nota-se que foi a grande maioria.

3.2 Algoritmo *Christofides*

Na **figura 2**, encontra-se a execução das instâncias abaixo do tempo limite, foi a grande maioria também, porém ficou abaixo do algoritmo twice and around

3.3 Algoritmo *Branch and Bound*

Nesse algoritmo, tive que criar instâncias pequenas, abaixo de 2^4, pois acima disso, que eram todas as que tinham sido disponibilizadas, o algoritmo não rodava abaixo do tempo limite. Na **figura 3**, encontra-se as instâncias que o algoritmo rodou.

3.4 Comparação Geral

Neste subtópico iremos abordar a comparação entre o dois algoritmos que tiveram êxito na maioria das instâncias e no tempo limite.

3.4.1 Uso de Memória

Ambos os algoritmos tiveram um uso de memória parecido podemos notar isso na **figura 4**, somente para instâncias um pouco maiores, o *twice and around* gastou mais memória do que o *christofides*

3.4.2 Qualidade da Solução

A qualidade da solução foi medido de acordo com o resultado obtido por cada algoritmo e comparado com a solução ótima do problema em questão. Infelizmente, o algoritmo branch and bound não rodou nas instâncias fornecidas, como citado acima, e não será considerado nessa análise.

Como esperado, o *christofides* obteve melhores resultados em comparação ao *twice and around* e ficou mais perto das soluções ótimas, de instâncias pequenas à grandes.

4 Conclusão

O presente trabalho abordou a implementação e análise de desempenho de três algoritmos clássicos para resolver o Problema do Caixeiro Viajante (PCV): *Branch and Bound*, *Twice Around the Tree* e *Christofides*. O escopo do estudo incluiu a execução desses algoritmos em instâncias de tamanho considerável, com o intuito de comparar e avaliar seu desempenho em cenários práticos.

Durante a implementação, observou-se que cada algoritmo possui características distintas, refletindo abordagens diversas na busca pela solução ótima ou aproximada. O *Branch and Bound*, por sua natureza exaustiva, mostrou-se eficiente para instâncias de pequeno a médio porte, proporcionando soluções precisas. Entretanto, seu desempenho decai consideravelmente ao lidar com instâncias maiores, demonstrando limitações em termos de escalabilidade.

A comparação entre esses algoritmos em instâncias de diferentes tamanhos ressalta a importância de considerar a natureza do problema e as características específicas das instâncias ao escolher a abordagem mais adequada. Cada algoritmo apresentou seu desempenho ótimo em determinadas circunstâncias, destacando a relevância de escolher a estratégia que melhor se alinha com as características da aplicação em questão.

5 Figuras

Nesse tópico encontram-se todas as figuras citadas na documentação

Instancia	Tempo de execucao (s)	Uso de memoria (MB)	Qualidade da solucao (distancia total)
a10.tsp	0.0010008811950683594	0.0703125 MB	3439.0
b15.tsp	0.0020022392272944922	0.09375 MB	3345.0
a20.tsp	0.0	0.125 MB	3488.0
a30.tsp	0.0	0.171875 MB	4364.0
eil51.tsp	0.01564621925354004	0.64453125 MB	584.0
berlin52.tsp	0.0	0.62109375 MB	10114.0
st70.tsp	0.015634536743164062	1.02734375 MB	888.0
eil76.tsp	0.01563286781311035	1.1875 MB	696.0
pr76.tsp	0.023123979568481445	1.16015625 MB	145336.0
rat99.tsp	0.031224966049194336	1.97265625 MB	1693.0
kroA100.tsp	0.046885251998901 37	1.9765625 MB	27210.0
kroB100.tsp	0.021064996719360 35	2.02734375 MB	25885.0
kroC100.tsp	0.047227859497070 31	1.99609375 MB	27968.0
kroD100.tsp	0.0502467155456543	1.9453125 MB	27112.0
kroE100.tsp	0.031253337860107 42	2.08984375 MB	29965.0
rd100.tsp	0.031245231628417 97	2.109375 MB	10790.0
eil101.tsp	0.031260251998901 37	1.953125 MB	830.0
lin105.tsp	0.046889543533325195	2.26953125 MB	19495.0
pr107.tsp	0.037313699722290 04	2.2421875 MB	54237.0
pr124.tsp	0.060418844223022 46	2.99609375 MB	74139.0
bier127.tsp	0.080537557601928 71	3.0859375 MB	158626.0
ch130.tsp	0.064353942871093 75	3.1796875 MB	8129.0
pr136.tsp	0.093725681304931 64	3.640625 MB	151904.0
pr144.tsp	0.090531110763549 8	4.05859375 MB	80599.0
ch150.tsp	0.117151498794555 66	4.1875 MB	8347.0
kroA150.tsp	0.093747377395629 88	4.21875 MB	35119.0
kroB150.tsp	0.109377384185791 02	4.28515625 MB	36150.0
pr152.tsp	0.120305061340332 03	4.25390625 MB	87995.0
u159.tsp	0.127057313919067 38	4.74609375 MB	57791.0
rat195.tsp	0.270994424819946 3	7.45703125 MB	3234.0
d198.tsp	0.156252861022949 22	7.80859375 MB	18936.0
kroA200.tsp	0.156216621398925 78	9.05859375 MB	40028.0
kroB200.tsp	0.162212848663330 08	7.97265625 MB	40703.0
ts225.tsp	0.177327394485473 63	10.8828125 MB	188008.0
tsp225.tsp	0.189859390258789 06	10.52734375 MB	5161.0
pr226.tsp	0.194333791732788 09	11.23046875 MB	116694.0
gil262.tsp	0.258059024810791	13.29296875 MB	3308.0
pr264.tsp	0.232227563858032 23	13.484375 MB	66470.0
a280.tsp	0.328507900238037 1	15.0 MB	3592.0
pr299.tsp	0.335658073425292 97	16.50390625 MB	64645.0
lin318.tsp	0.482088327407836 9	18.421875 MB	58138.0
linhp318.tsp	0.389789342880249	18.8359375 MB	58138.0
rd400.tsp	0.788331747055053 7	32.296875 MB	20331.0
fl417.tsp	0.941182851791381 8	34.625 MB	16200.0
pr439.tsp	0.772995710372924 8	39.13671875 MB	144623.0
pcb442.tsp	1.321531295776367 2	40.234375 MB	69223.0
d493.tsp	1.278400182723999	44.2109375 MB	44884.0
u574.tsp	1.328374147415161 1	60.875 MB	48902.0
rat575.tsp	1.304615974426269 5	59.12890625 MB	9393.0
p654.tsp	1.745330810546875	76.0273437 5 MB	49699.0
d657.tsp	1.849424362182617 2	83.9375 MB	66342.0
u724.tsp	3.097815990447998	100.98828125 MB	57721.0
rat783.tsp	2.62933611869812	114.82421875 MB	11921.0
pr1002.tsp	4.761837005615234	175.8359375 MB	342216.0
u1060.tsp	7.073614597320557	205.6015625 MB	305849.0
vm1084.tsp	5.614326953887939 5	221.11328125 MB	315277.0
pcb1173.tsp	6.650229930877685 5	240.703125 MB	80761.0
d1291.tsp	10.804388046264648	286.22265625 MB	75282.0
rl1304.tsp	8.902010202407837	308.93359375 MB	348309.0
rl1323.tsp	9.0702805519104	321.328125 MB	380427.0
nrw1379.tsp	12.179904937744 14	387.91015625 MB	79188.0
fl1400.tsp	9.554164171218872	406.8671875 MB	28022.0
u1432.tsp	10.136729478836 06	389.0703125 MB	214417.0
fl1577.tsp	13.593971729278564	411.22265625 MB	30257.0
d1655.tsp	17.302374839782715	500.265625 MB	85285.0
vm1748.tsp	16.593135595321655	579.80859375 MB	442756.0
u1817.tsp	18.071835994720 46	593.484375 MB	82013.0
rl1889.tsp	26.652684926986694	668.30859375 MB	448611.0
d2103.tsp	27.273663997650146	775.50390625 MB	123086.0
u2152.tsp	26.226724147796 63	808.703125 MB	93801.0
u2319.tsp	34.157940626144 41	911.046875 MB	320506.0
pr2392.tsp	33.713236093521 12	970.19921875 MB	523107.0
pcb3038.tsp	57.614311695098 88	1716.32421875 MB	197498.0
fl3795.tsp	104.851320505142 21	2025.52734375 MB	36256.0
fnl4461.tsp	153.943231344223 02	2503.9140625 MB	255829.0
rl5915.tsp	613.141539573669 4	3022.94140625 MB	842903.0
rl5934.tsp	644.771690607070 9	3668.9570312 MB	817449.0
rl11849.tsp	Timeout	1	0.0

Figure 1: Tabela de Resultados para o algoritmo *Twice Around the Tree*.

Instancia	Tempo de execucao (s)	Uso de memoria (MB)	Qualidade da solucao (distancia total)
a10.tsp	0.00200247764587402 34	0.1015625 MB	2900.0
b15.tsp	0.00835204124450683 6	0.125 MB	2782.0
a20.tsp	0.0	0.17578125 MB	3236.0
a30.tsp	0.0	0.2421875 MB	3844.0
eil51.tsp	0.025173187255859375	0.69921875 MB	462.0
berlin52.tsp	0.015623331069946289	0.69921875 MB	8591.0
st70.tsp	0.07812905311584473	1.20703125 MB	770.0
eil76.tsp	0.09377479553222656	1.39453125 MB	608.0
pr76.tsp	0.053248405456542 97	1.18359375 MB	116684.0
rat99.tsp	0.10937356948852539	2.2109375 MB	1393.0
kroA100.tsp	0.11221051216125488	2.52734375 MB	23293.0
kroB100.tsp	0.10147333145141602	2.3125 MB	24012.0
kroC100.tsp	0.11702179908752441	2.3203125 MB	23470.0
kroD100.tsp	0.125	2.37890625 MB	23583.0
kroE100.tsp	0.125	2.4921875 MB	23783.0
rd100.tsp	0.21506190299987793	2.52734375 MB	8906.0
eil101.tsp	0.1562497615814209	2.45703125 MB	707.0
lin105.tsp	0.1305861473083496	2.48828125 MB	16320.0
pr107.tsp	0.14888906478881836	2.546875 MB	47891.0
pr124.tsp	0.09374403953552246	3.2734375 MB	64438.0
bier127.tsp	0.41927337646484375	3.65234375 MB	132448.0
ch130.tsp	0.25789308547973633	3.3203125 MB	6773.0
pr136.tsp	0.11495089530944824	3.7734375 MB	103771.0
pr144.tsp	0.1400909423828125	3.859375 MB	70404.0
ch150.tsp	0.31867408752441406	4.62890625 MB	7135.0
kroA150.tsp	0.7458550930023193	4.921875 MB	29688.0
kroB150.tsp	0.6001486778259277	4.9140625 MB	30053.0
pr152.tsp	0.142904520034790 04	4.25 MB	79311.0
u159.tsp	0.3661172389984131	4.92578125 MB	47586.0
rat195.tsp	0.5780565738677979	8.015625 MB	2630.0
d198.tsp	0.8694841861724854	8.68359375 MB	17347.0
kroA200.tsp	1.0426633358001 71	8.98046875 MB	33232.0
kroB200.tsp	0.7540261745452881	8.94921875 MB	33119.0
ts225.tsp	0.2247281074523925 8	11.05078125 MB	134282.0
tsp225.tsp	0.5849764347076416	11.625 MB	4397.0
pr226.tsp	0.5272202491760254	11.17578125 MB	92415.0
gil262.tsp	1.220771074295044	15.52734375 MB	2724.0
pr264.tsp	0.6719141006469727	14.62109375 MB	54662.0
a280.tsp	1.1448252201080322	16.05078125 MB	2928.0
pr299.tsp	1.7612199783325195	17.25390625 MB	52969.0
lin318.tsp	2.1274921894073486	20.33984375 MB	47246.0
linhp318.tsp	2.5458669662475586	20.4296875 MB	47246.0
rd400.tsp	5.232697010040283	35.828125 MB	17501.0
fl417.tsp	2.4249329566955566	34.73046875 MB	13175.0
pr439.tsp	4.0165932178497314	34.8203125 MB	119525.0
pcb442.tsp	6.013031005859375	39.85546875 MB	54868.0
d493.tsp	10.276641368865967	49.13671875 MB	38697.0
u574.tsp	9.509456634521484	63.58203125 MB	41424.0
rat575.tsp	9.3036208152771	61.5703125 MB	7811.0
p654.tsp	2.882143020629883	76.94140625 MB	39292.0
d657.tsp	16.49935531616211	86.74609375 MB	55007.0
u724.tsp	20.640677452087402	109.5234375 MB	47431.0
rat783.tsp	22.411659717559814	130.0703125 MB	10036.0
pr1002.tsp	49.89134740829468	199.859375 MB	286203.0
u1060.tsp	53.71830439567566	224.2890625 MB	249289.0
vm1084.tsp	43.37251353263855	228.90625 MB	260770.0
pcb1173.tsp	48.12723779678345	254.109375 MB	63767.0
d1291.tsp	20.933788299560547	288.6953125 MB	57655.0
fl1304.tsp	26.703613281 25	310.359375 MB	277037.0
fl1323.tsp	21.320290088653564	323.7265625 MB	298272.0
nrw1379.tsp	147.084681987762 45	408.94140625 MB	63703.0
fl1400.tsp	159.67664456367493	455.27734375 MB	23005.0
u1432.tsp	79.755531072616 58	416.734375 MB	171469.0
fl1577.tsp	43.961637020111084	470.3203125 MB	24606.0
d1655.tsp	100.83222818374634	526.33984375 MB	69721.0
vm1748.tsp	147.69261121749878	586.78125 MB	376275.0
u1817.tsp	97.65887260437012	612.5078125 MB	66125.0
rl1889.tsp	74.67782020568848	613.1875 MB	344580.0
d2103.tsp	28.196170330047607	776.41796875 MB	84209.0
u2152.tsp	161.7704141139984	834.78125 MB	72407.0
u2319.tsp	622.4770214557648	1110.125 MB	271669.0
pr2392.tsp	546.4919443130493	1068.984375 MB	425940.0
pcb3038.tsp	900.7297370433807	1819.69140625 MB	154670.0
fl3795.tsp	1525.8957216739655	1906.484375 MB	31935.0
fnl4461.tsp	Timeout	1	0.0

Figure 2: Tabela de Resultados para o algoritmo *Christofides*.

Instancia	Tempo de execucao (s)	Uso de memoria (MB)	Qualidade da solucao (distancia total)
a10.tsp	7.4109766483306885	2.4140625 MB	2160.0
b15.tsp	Timeout	1	0.0

Figure 3: Tabela de Resultados para o algoritmo *Branch and Bound*.

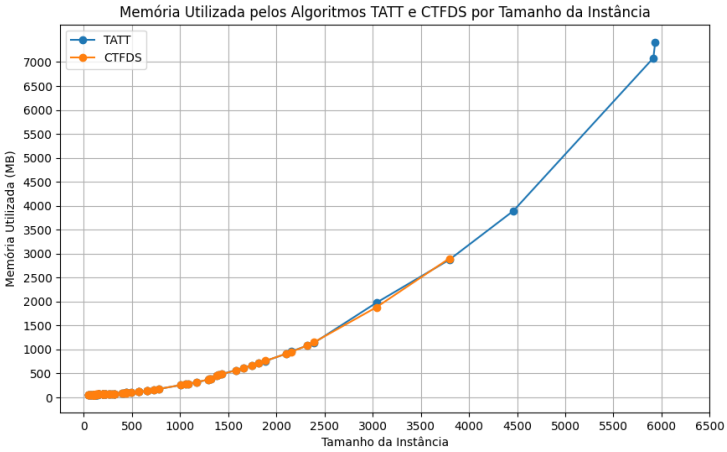

Figure 4: Comparação Geral - Uso de Memória.

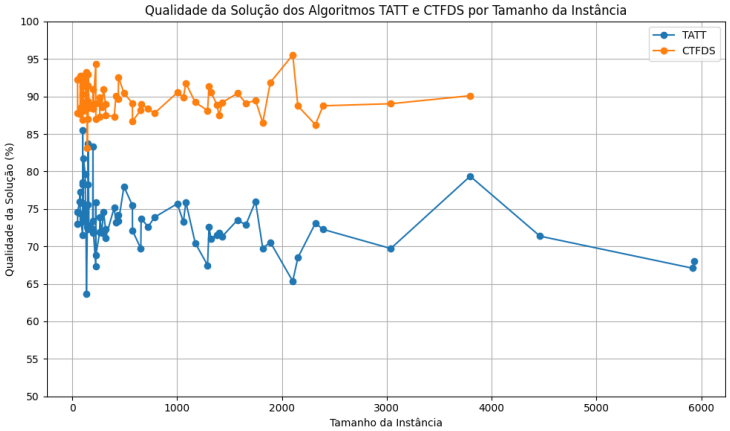

Figure 5: Comparação Geral - Qualidade da Solução.