Trabalho Prático 2 - Problema do Caixeiro Viajante
Algoritmos 2

Francisco Teixeira Rocha Aragão

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

Abstract. *This article seeks to explore the Traveling Salesman Problem (TSP), which is an NP class problem. Thus, an exact algorithm was implemented using Branch and Bound to accurately solve the problem, in addition to two approximate algorithms, Twice Around the Tree and Christofides. Therefore, the implementations, analyzes and uses of such algorithms to solve the problem will be discussed.*

Resumo. *Esse artigo busca explorar o problema do Caixeiro Viajante (PCV), que é um problema da classe NP. Assim, foi imeplementado um algoritmo exato utilizando Branch and Bound para resolução do problema, além de dois algoritmos aproximativos, Twice Around the Tree e Christofides. Desse modo, será discutido as implementações, anaĺises e usos de tais algoritmos.*

1. Introdução

Esse trabalho busca explorar o problema do caixeiro viajante, usando um algoritmo exato baseado em Branch and Bound além de dois algoritmos aproximativos, o Twice Around the Tree e o de Christofides. Desse modo, tais discussões abordam a dificuldade da resolução de problemas pertencentes a classe NP, além de maneiras de tratarmos essas situações através de algoritmos aproximativos.

2. Definição do problema

O problema do caixeiro viajante aborda a situação em que temos um conjunto de cidades e queremos então percorrer todas essas cidades apenas uma vez, voltando ao ponto inicial com o menor custo possível. Tal problema pode ser modelado com o uso de grafos, em que as cidades são representadas por vértices e os caminhos (que devem ser de menor custo possível) sendo as arestas.

3. Algoritmos utilizados

Para resolução e análise do problema, foram utilizados três algoritmos diferentes que serão discutidos a seguir:

3.1. Branch And Bound

A estratégia de Branch and Bound, consiste em explorar todo o espaço de busca de maneira inteligente, evitando certos ramos de computação de acordo com a possível solução a ser encontrada. Assim, o algoritmo escolhe aqueles nós da árvore de possibilidades que oferecem melhores estimativas de custos (ou seja, que apresentam soluções melhores do que as já encontradas), de modo a direcionar a computação para ramos mais

promissores. Dessa forma, mesmo que a complexidade ainda continue sendo exponencial, na prática conseguimos melhorar esse desempenho, mesmo que a complexidade não seja alterada.

Sua implementação foi feita utilizando uma estratégia A* com o uso de uma fila de prioridades. Essa estratégia é uma mistura de busca em largura com busca em profundidade (Best First Search), explorando os melhores ramos dentre os disponíveis no momento. Tal estratégia é feita escolhendo o ramo mais promissor no nível atual (busca em largura) e enão esse nó é explorado (busca em profundidade).

Como o algoritmo explora toda a árvore de busca (mesmo que ignore alguns ramos que não são produtivos), a solução encontrada é a ótima. No entanto, como será mostrado a seguir, nem sempre é possível encontrar o ótimo em tempo hábil pelo caráter exponencial do problema.

3.2. Twice Around the Tree

Esse algoritmo aproximativo consiste em utilizar uma árvore geradora mínima (AGM) entre todos os vértices. Desse modo, ao obter a AGM associada ao grafo, basta agora construirmos um ciclo hamiltoniano, ou seja, um caminho que inicia e termina no mesmo ponto e que passa por todos os vértices apenas uma vez. Dessa forma, pela definição de ciclo hamiltoniano, estamos encontrando a solução do PCV fazendo essas etapas. Para implementação do trabalho, foi utilizado a biblioteca Networkx que facilita a construção do grafo, a obtenção da árvore geradora mínima e do ciclo hamiltoniano, tornando o código bem simples e direto.

O algoritmo funciona partindo da hipótese de que a função de distância é uma métrica, ou seja, respeita as seguintes premissas (sendo 'u', 'v' e 'w' vértices):

1. $distancia(u, v) \geq 0$
2. $distancia(u, v) = distancia(v, u)$
3. $distancia(u, v) \iff u = v$
4. $distancia(u, v) \leq distancia(u, w) + distancia(w, v)$

Desse modo, como algoritmo busca então construir um ciclo hamiltoniano na árvore geradora mínima para resolução do problema, no processo alguns vértices podem acabar se repetindo. Como exemplo, se tenho algum vértice A ligado a somente algum vértice B, para chegar e sair de A, preciso passar por B duas vezes. Seguindo o exemplo, se vamos de B para A, de A para B e de B para C, podemos evitar a repetição indo direto de A para C. Isso é possível pelo item 4 da função métrica e também pelo fato de que no PCV, o grafo é completo, então existem arestas entre todos os vértices.

Desse modo, o algoritmo acaba considerando a duplicação de várias arestas que não são necessárias, já que em alguns vértices, podemos passar por eles sem repetições, o que afeta o fator de aproximação do algoritmo que é 2-aproximado. Tal fato pode ser otimizado, como será visto no algoritmo a seguir. Em todo caso, o algoritmo é polinomial, com a complexidade dominada pela construção da árvore geradora mínima, que é polinomial, possuindo uma performance muito melhor do que o de branch and bound, porém, retornando um resultado aproximado.

3.3. Christofides

O segundo algoritmo aproximativo utilizado foi o de Christofides. Esse algoritmo, de maneira semelhante ao Twice Around the Tree, também leva em consideração que a função de distância é uma métrica, porém atua de maneira um pouco mais inteligente.

Assim como dito anteriormente, o algoritmo de Christofides também atua utilizando a árvore geradora mínima que representa o grafo. Porém, é feito após isso um matching de peso mínimo nos vértices de grau ímpar, para então ser construido um novo grafo com as arestas do matching mínimo juntamente a árvore geradora mínima. Dessa forma, é possível fazer um circuito euleriano (visitar todas as arestas uma única vez) e então removover os vértices repetidos assim como no algoritmo anterior. Sua implementação no trabalho foi feita novamente utilizando a biblioteca Networkx que facilitou o processo para construção de subgrafos, matching de peso mínimo e para encontrar o circuito euleriano.

Desse modo, o ponto de melhoria desse algoritmo está no fato de que é construído um matching de peso mínimo em cima dos vértices de grau ímpar, ou seja, iremos 'duplicar' as arestas apenas daqueles vértices que precisam ser duplicadas, pois os vértices de grau par possuem arestas para entrar e sair, o que não acontece com os de grau ímpar. Com isso, a algoritmo de Christofides é 1.5 aproximado, dando respostas melhores no pior caso do que o algoritmo Twice Around the Tree. Porém, mesmo sendo um algoritmo polinomial, apresenta um custo assintótico maior pelas operações feitas sobre os vértices de grau ímpar, em comparação ao algoritmo anterior.

4. Desempenho

Os testes que serão analisados a seguir foram todos feitos em uma máquina com Manjaro XFCE com kernel 6.1, 8GB de Ram e processador i3. Todo o código foi desenvolvido em Pythom, sendo as informações sobre tempo obtidas com a biblioteca time e sobre memória com o auxílio da biblioteca memory-profiler.

4.1. Testes iniciais

Realizando alguns testes menores para checagem inicial dos algoritmos, foi possível perceber alguns resultados interessantes. Testando o algoritmo para instâncias de 4 até 10 vértices, foi perceptível a diferença de velocidade e de qualidade da solução entre os algoritmos.

Percebe-se que, como esperado, o algoritmo usando Branch and Bound consegue alcançar sempre a resposta ótima (os ótimos foram conferidos a mão pequeno tamanho das instâncias de teste). Além disso, é perceptível o uso máximo de memória praticamente igual entre todos os problemas como demonstrado nas tabelas 1, 2 e 3 que estão abaixo. No entanto, ao aumentar o número de vértices, ainda que sejam poucos, pelo comportamento ainda exponencial do algoritmo, o tempo sobe exponencialmente, aumentando consideravelmente até a última instância de 11 vértices, tornando o problema praticamente inviável de ser testado para instâncias maiores, como será exemplificado mais abaixo.

Já os algoritmos aproximativos apresentaram bons resultados, o Twice Around the Tree, como previsto, por possuir menor complexidade assintótica, conseguiu os menores

tempos para todas as instâncias. Entretanto, a solução foi a pior em comparação ao de Christofides, que em alguns casos chegou até mesmo a alcançar o valor ótimo como mostrado na tabela 3. Em todo caso, ambos os algoritmos respeitaram os limiares de aproximação, retornando valores no máximo, 2 e 1.5 aproximado, respectivamente.

Número de Vértices	Tempo (s)	Memória máxima	Solução	Ótimo
4	0.00016	51.22	14	14
5	0.0005	51.16	16.06	16.06
10	13.456	65.09	24.880	24.880
11	37.376	77.93	25.76	25.76

Table 1. Resultados Branch and Bound

Resultados do Twice Around the Tree para as mesmas instâncias:

Número de Vértices	Tempo (s)	Memória máxima	Solução	Ótimo
4	0.0013	51.23	16	14
5	0.0014	51.28	18.06	16.06
10	0.0015	63.46	32.605	24.880
11	0.0015	75.32	35.230	25.76

Table 2. Resultados Twice Around the Tree

Resultados de Christofides para as mesmas instâncias:

Número de Vértices	Tempo (s)	Memória máxima	Solução	Ótimo
4	0.0022	51.28	14	14
5	0.0030	51.33	16.06	16.06
10	0.0037	63.47	27.132	24.880
11	0.0082	75.32	25.76	25.76

Table 3. Resultados Christofides

4.2. Testes definitivos

Agora foram utilizadas instâncias maiores fornecidas na descrição do trabalho.

- Resultados de Branch and Bound:

Com dito acima, o algoritmo utilizando Branch and Bound mostrou-se inviável na prática. Foi testado o desempenho do algoritmo para a menor instância de teste informada, contendo 51 vértices. Após 30 minutos, a melhor solução encontrada até o momento era 896, sendo que o ótimo é 426. Assim, as demais instâncias maiores nem foram testadas visto que também não seriam resolvidas. Uma nova estratégia foi utilizada, já iniciando o algorirtmo com a solução encontrada pelo algoritmo de Christofides, juntamente ao peso mínimo associado a esse caminho, sendo possível, teoricamente, podar mais ramos logo de início. Porém, ainda não foi possível finalizar o algoritmo em tempo hábil, mostrando a dificuldade em lidar com soluções exponenciais.

Desse modo, os testes para o algoritmo de Branch and Bound não foram feitos para instâncias maiores, sendo coletados apenas os resultados de Christofides e Twice Around the Tree que serão discutidos abaixo.

- Resultados para os algoritmos aproximativos:

Como possível de ser observado com as tabelas abaixo, os algoritmos aproximativos cumpriram seu papel e conseguiram resolver as maiores instâncias de testes fornecidas no trabalho, ao contrário do Branch and Bound em que a execução tornou-se inviável. Desse modo, é perceptível como o Twice Around the Tree mostrou-se ser o algoritmo mais rápido, demorando quase 30 segundos e menos de 1.5 GB de memória para 2392 vértices, ao contrário de Christofides que demorou mais de 400 segundos e mais de 2GB de memória ram como exemplificados nas tabelas 4 e 5. No entanto, a solução encontrada nesse segundo caso foi bem melhor, o que é generalizado para as demais situações em que a aproximação feita por Christofides foi bem mais próxima ao ótimo do que o algoritmo Twice Around the Tree.

Ainda sobre a qualidade da solução, vale mencionar que mesmo que Christofides tenha uma aproximação de 1.5, a pior solução apresentada foi para 70 vértices, obtendo um valor 1.14 vezes aproximada (valor obtido de 775). Em todo caso, tal resultado ainda está bem abaixo do limiar máximo do algoritmo, que seria no pior caso 675 (valor ótimo) * 1.5 = 1012.5. O mesmo é válido para Twice Around the Tree, em que no pior caso, retornou uma solução 1.47 vezes pior do que o ótimo (valor obtido de 95076), que está abaixo do limite de 64253 (ótimo) * 2 = 128506. Desse modo, de maneira geral o algoritmo de Christofides é uma boa escolha em questão da qualidade da solução, porém, existe o trade-off relacionado a tempo e memória, consumindo mais memória e uma quantidade bem maior de tempo para instâncias maiores, em comparação ao Twice Around the Tree.

Vale destacar que para instâncias maiores que 2392 vértices, não foi possível executar os testes pois não havia memória suficiente disponível para a execução dos algoritmos.

- Resultados de Twice Around the Tree para as instâncias de testes:

Número de Vértices	Tempo (s)	Memória máxima	Solução	Ótimo
52	0.187	476.234	10116.014	7542
70	0.221	449.140	873.351	675
107	0.198	314.718	54238.036	44303
124	0.300	454.648	74140.959	59030
200	0.508	776.968	40030.866	29437
318	0.802	458.0	58145.441	42029
575	3.194	698.078	9438.933	6773
724	5.056	953.101	57779.667	41910
1084	5.316	663.480	315268.348	239297
1577	10.874	722.273	31223.096	22249
1889	16.811	1224.746	449811.488	316536
2152	21.650	1171.320	95076.215	64253
2392	27.809	1389.375	523132.906	378032

Table 4. Resultados Twice Around the Tree

- Resultados de Christofides para as mesmas instâncias de testes:

Número de Vértices	Tempo (s)	Memória máxima	Solução	Ótimo
52	0.309	436.859	8235.198	7542
70	0.274	449.140	775.039	675
107	0.317	314.718	48379.607	44303
124	0.331	454.648	63071.622	59030
200	1.187	430.957	32045.584	29437
318	3.347	452.093	47643.238	42029
575	11.035	425.92	7594.845	6773
724	18.714	530.464	46770.281	41910
1084	30.156	648.375	258620.572	239297
1577	41.940	1000.875	24054.873	22249
1889	53.440	1366.0	343073.261	316536
2152	131.348	1709.890	72000.007	64253
2392	405.337	2067.632	416299.652	378032

Table 5. Resultados Christofides

Observação: Nem todos os arquivos do conjunto de testes foram colocados na tabela, sendo informado um conjunto menor para exemplificar o comportamento geral dos algoritmos utilizados.

5. Conclusões

Após os testes feitos que foram discutidos anteriormente, percebe-se os pontos positivos e negativos de cada um dos algoritmos implementados. Caso a solução deva consumir o mínimo de recursos da máquina, deve ser usado o algoritmo Twice Around the Tree, que mostrou-se ser o mais rápido e com o menor consumo de memória de todos. No entanto, deve ser levado em consideração a qualidade da resposta que pode não ser uma aproximação muito boa dependendo da aplicação.

Já o algoritmo de Christofides demonstrou sua capacidade de aproximação, produzindo resultados muito próximos do ótimo nas instâncias de testes utilizadas. Assim, tal algoritmo apresenta-se como uma escolha mais sólida para resolução desse problema, em contraste com o maior tempo gasto devido a sua complexidade e também ao maior uso de memória. Porém, dependendo da aplicação, tais condições são compensadas pela melhor qualidade da solução.

Pensando agora na implementação para solução exata, o algoritmo utilizando Branch and Bound mostrou-se inviável para resolução do problema. Assim, para valores na faixa de 2^4 ou maiores, o algoritmo torna-se extremamente lento, tendo em vista o espaço de busca enorme que deve ser visitado para a solução ser encontrada. Cabe então o uso de heurísticas para tentar podar mais ramos e evitar computações desnecessárias, para assim tentar diminuir o tempo necessário para conclusão do algoritmo. Assim, mesmo que o algoritmo seja exato, sua aplicação deve ser muito bem pensada e dependendo da situação até deixada de lado, dando espaço para algum algoritmo aproximativo que pode ser mais proveitoso.

Assim, de maneira geral, o presente trabalho conseguiu explorar bem a dificuldade de resolução de problemas da classe NP e como soluções aproximativas e heurísticas mostra-se úteis para tratamento desses casos.

6. Referências

NETWORKX.ORG. Software for Complex Networks. 2023. Disponível em: https://networkx.org/documentation/stable/index.html. Acesso em: 04 dez. 2023.

ZHAO, Roy Mathew, Divya Cherukupalli Kevin Pusich, Kevin. Software for Complex Networks. Disponível em: https://cse442-17f.github.io/Traveling-Salesman-Algorithms/. Acesso em: 05 dez. 2023.

EDUCATION, Washington. Lecture notes on bipartite matching. Disponível em: https://sites.math.washington.edu/ raymonda/assignment.pdf. Acesso em: 04 dez. 2023.