Trabalho Prático 2
Problema do Caixeiro Viajante

Gabriel Teixeira Carvalho[1]

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

Abstract. *This article addresses the euclidean Traveling Salesman Problem (TSP). The main objective is to analyze different implementations, exact or approximate, in terms of complexity of time, space and quality of the solution, in addition to other technical aspects of the algorithms.*

Resumo. *Esse artigo lida com o Problema do Caixeiro Viajante (PCV) euclidiano. O objetivo principal é analisar diferentes implementações, exatas ou aproximadas, em termos de complexidade de tempo, espaço e qualidade da solução, além de outros aspectos técnicos dos algoritmos.*

1. Introdução

Nesse artigo, será abordado o Problema do Caixeiro Viajante, que consiste em achar um circuito Hamiltoniano de custo mínimo em um grafo. O PCV possui importantes aplicações práticas para as áreas de logística, roteamento e otimização de custos. Serão apresentadas três implementações para o problema, uma solução exata baseada em *Branch and Bound* e duas aproximadas, a versão *Twice Around The Tree* e a *Christofides*.

A seção 2 desse artigo trata sobre a definição formal do problema, enquanto na seção 3 são apresentadas as diferentes soluções para o problema. Na seção 4 são apresentados experimentos realizados com as três implementações. Por fim, na seção 5 estão presentes as análises dos experimentos e a conclusão.

2. Definição Formal

O Problema do Caixeiro Viajante, em sua versão de otimização, consiste em encontrar um circuito Hamiltoniano de custo mínimo em um grafo ponderado. Ou seja, deve-se achar o menor caminho que parte de um vértice inicial, passa por todos os outros exatamente uma vez e retorna à origem. Será abordada a versão Euclidiana do problema, na qual as seguintes condições são satisfeitas:

Seja $G(V, E)$ um grafo com o conjunto V de vértices e o conjunto E de arestas, então:

1. $\forall i, j \in V : distancia(i, j) > 0$
2. $\forall i, j \in V : distancia(i, j) = distancia(j, i)$
3. $\forall i, j, k \in V : distancia(i, j) \leq distancia(i, k) + distancia(k, j)$

Além disso, para calcular as distâncias serão utilizadas duas funções de custo:

Sejam P, Q pontos no plano com coordenadas (x_P, y_P) e (x_Q, y_Q), respectivamente, então:

- Distância Euclidiana: $distancia(P, Q) = \sqrt{(x_P - x_Q)^2 + (y_P - y_Q)^2}$
- Distância Manhattan: $distancia(P, Q) = |(x_P - x_Q) + (y_P - y_Q)|$

Naturalmente, para pontos no plano, as três condições do PCV Euclidiano são satisfeitas. Vale ressaltar que as distâncias calculadas para um conjunto de pontos utilizando a distância Manhattan serão, naturalmente, maiores que aquelas calculadas com a distância Euclidiana. Isso acontece devido à forma como o cálculo é feito, mas esse fato não impacta no desempenho dos algoritmos.

3. Algoritmos

O Problema do Caixeiro Viajante é um problema NP-difícil, para o qual ainda não foi descoberta nenhuma solução exata que o resolva em tempo polinomial. Além disso, mesmo para instâncias pequenas, resolvê-lo de forma exata torna-se impraticável. Por conta disso, além de uma solução exata, serão abordados dois algoritmos aproximativos, que garantem soluções que são, no máximo, duas vezes pior que a solução ótima.

3.1. *Branch and Bound*

Esse algoritmo consiste em uma abordagem de Branch and Bound, caracterizada pela poda de ramos não promissores da árvore de possibilidades. Isso permite resolver o problema de forma mais eficiente do que por força bruta e, ainda assim, garante a exatidão da resposta.

Para escolher quais nós serão considerados primeiro e quais devem ser podados, é definido um limite inferior (bound) para cada nó, baseado na solução parcial. Para calcular o bound de um nó, deve-se somar as duas menores arestas ou as arestas da solução incidentes a cada vértice, dividindo o resultado por 2. Visando manter o tempo de processamento de cada nó o menor possível, a função responsável por calcular o limite inferior opera em tempo constante. Isso foi possível através do armazenamento do conjunto de arestas do bound em cada nó, bastando, então, atualizar apenas as arestas correspondentes aos dois últimos vértices da solução parcial do nó. Assim, a complexidade temporal da função se torna constante, ao custo de uma complexidade de espaço $O(n)$ para cada nó, onde $n = |V|$.

A partir dessa estratégia, utiliza-se um Min-Heap para manter a ordem dos nós de forma eficiente, permitindo inserir um novo elemento ou remover o valor mínimo do Heap com complexidade $O(\log n)$. Nesse contexto, a prioridade dos nós na estrutura de dados é baseada, primeiramente, no tamanho da solução do nó e usa, como critério de desempate, o limite inferior calculado. Dessa forma, alcança-se valores ótimos parciais de maneira mais rápida, o que impede que muitos nós sejam considerados desnecessariamente.

Além disso, sem perda de generalidade, pode-se ignorar todos os nós cuja solução é um caminho repetido mas em sentido contrário. Outra otimização utilizada é a previsão do resto do caminho ao alcançar uma solução com tamanho $n - 2$, o que evita processamentos desnecessários. Por fim, o algoritmo ainda continua com a mesma complexidade da solução por força bruta, ou seja, $O(n!)$, porém, é possível resolver instâncias muito maiores dessa forma otimizada.

3.2. *Twice Around the Tree*

O algoritmo *Twice Around the Tree* é uma solução aproximada para o TSP. A ideia por trás de sua implementação consiste em achar uma Árvore Geradora Mínima (AGM) para

o grafo do problema e, então, percorrer essa árvore seguindo uma pré-ordem.

A árvore pode ser encontrada através do algoritmo de Kruskal, que retorna uma AGM com complexidade de tempo $O(|E| \log |V|)$. Para encontrar o circuito Hamiltoniano a partir da árvore basta percorrê-la com uma Busca em Profundidade, que tem complexidade $O(|V_{AGM}| + |E_{AGM}|)$. Logo, o algoritmo possui complexidade assintótica temporal $O(|E| \log |V|)$. Além disso, conta com complexidade espacial linear no tamanho da entrada.

Essa abordagem não retornará, necessariamente, a resposta exata para toda instância do PCV. Porém, é possível demonstrar que qualquer solução S dada pelo algoritmo satisfaz:

$S \leq 2S^*$, onde S^* é a solução ótima para a instância. [Levitin 2011]

Portanto, apesar de o algoritmo não retornar sempre a melhor resposta possível, ele sempre executa uma quantidade polinomial de passos. Para muitas situações, é preferível encontrar uma resposta em tempo eficiente mesmo que não seja a resposta exata.

3.3. *Christofides*

Esse algoritmo é um aprimoramento do *Twice Around the Tree*. Ele consiste em:

1. encontrar uma AGM,
2. achar um emparelhamento mínimo M no subgrafo de G que é formado pelos nós com grau ímpar nessa árvore,
3. gerar um multigrafo G' a partir das arestas da AGM juntamente com as de M e,
4. achar um circuito Hamiltoniano a partir de um circuito Euleriano em G'.

A complexidade temporal do algoritmo é dominada pelo passo 2, pois é utilizado o algoritmo de Edmonds para achar um emparelhamento perfeito mínimo, o qual possui complexidade de tempo $O(|V|^2|E|)$[Chekuri 2010] e complexidade de espaço linear no tamanho da entrada. Entretanto, é demonstrado que esse algoritmo é 1.5-aproximado, ou seja, satisfaz a relação:

$S \leq 1.5S^*$, onde S^* é a solução ótima para a instância. [Levitin 2011]

Portanto, apesar de esse ser um algoritmo mais lento do que o anterior, ele permite encontrar resultados mais próximos do ótimo. Nesse sentido, o algoritmo Christofides pode ser preferível, dependendo do tamanho da instância ou da qualidade de resposta esperada para o problema.

4. Experimentos

Para mostrar na prática os aspectos comentados a respeito dos algoritmos, foram realizados experimentos de tempo e de qualidade da solução. Para o algoritmo de Branch and Bound, foi calculada a média do tempo gasto em 10 execuções de instâncias de tamanho 2^4, para ambas as funções de custo apresentadas. Todas as instâncias de tamanho 2^i, com $i > 4$, demoraram mais de 30 minutos, ou 1800 segundos, para serem executadas, sendo abortadas após essa quantidade de tempo.

Para ambos os algoritmos aproximativos, foi encontrada a média do tempo de execução de 10 iterações com instâncias de tamanho 2^i, com $4 \leq i \leq 10$.

Tabela 1. Desempenho do algoritmo Branch and Bound (em segundos)

Tamanho	Distância Euclidiana	Distância Manhattan
2^4	111.865396	101.834885
2^5	NA	NA
2^6	NA	NA
2^7	NA	NA
2^8	NA	NA
2^9	NA	NA
2^{10}	NA	NA

Tabela 2. Desempenho do algoritmo Twice Around the Tree (em segundos)

Tamanho	Distância Euclidiana	Distância Manhattan
2^4	0.001248	0.000910
2^5	0.002766	0.002982
2^6	0.011515	0.011416
2^7	0.025025	0.022551
2^8	0.135089	0.120374
2^9	0.664651	0.564767
2^{10}	3.247859	2.895863

Além disso, também foi analisada a qualidade das soluções para os algoritmos aproximativos. Para isso, foi calculada a média entre 1000 iterações com instâncias de tamanho 10 da métrica $\frac{S}{S^*}$, onde S é a solução encontrada pelo algoritmo e S^* é o valor da solução ótima.

5. Conclusão

Através da realização dos experimentos, é perceptível as vantagens de cada algoritmo. Apesar de o Branch and Bound permitir encontrar respostas exatas para o PCV, ele demanda grandes quantidades de tempo e espaço, mesmo para instâncias pequenas. Por isso, muitas vezes é necessário recorrer a algoritmos aproximativos. Entre eles, o Twice Around the Tree possui uma complexidade consideravelmente menor, porém, retorna soluções quase 10% piores do que o Christofides.

Portanto, conclui-se que a decisão acerca do algoritmo a ser utilizado deve ser feita baseado nas características da instância em questão e das prioridades do usuário. Caso a instância seja pequena, o Branch and Bound pode ser uma boa opção, por retornar uma resposta exata. Porém, com instâncias maiores deve-se optar pelo Christofides, que se aproxima do resultado ótimo, apesar de ser mais lento que o Twice Around the Tree, o qual deve ser escolhido para instâncias ainda maiores.

Além disso, quando utiliza-se a distância Manhattan como função de custo, ao invés da distância Euclidiana, foi possível perceber uma tendência a respostas consideravelmente mais rápidas porém ligeiramente piores em qualidade.

Referências

Chekuri, C. (2010). Min cost perfect matching. https://courses.engr.illinois.edu/cs598csc/sp2010/Lectures/Lecture11.pdf. [Online;

Tabela 3. Desempenho do algoritmo Christofides (em segundos))

Tamanho	Distância Euclidiana	Distância Manhattan
2^4	0.003010	0.003417
2^5	0.010737	0.009063
2^6	0.041406	0.032911
2^7	0.308233	0.257352
2^8	1.309372	1.086389
2^9	9.596196	6.892133
2^{10}	56.269430	44.271717

Tabela 4. Qualidade das soluções aproximadas

Algoritmo	Distância Euclidiana	Distância Manhattan
TATT	1.1477	1.1715
Christofides	1.0573	1.0597

acessado em 09/12/2022].

Levitin, A. (2011). *Introduction to the design & analysis of algorithms.* Addison-Wesley, 3rd edition.