Análise de Algoritmos para o Problema do Caixeiro Viajante

Philipe Dutra Cunha , Salvador Cândido

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

{philipedc, salvador.candido}@ufmg.br

1. Introdução

O problema do Caixeiro Viajante (TSP) consiste em encontrar o menor circuito em um grafo que percorra todos os vértices. Devido à sua importância, diversos algoritmos surgiram para resolvê-lo.

Esse problema é NP-Difícil e soluções exatas para várias instâncias dele não são possíveis de serem encontradas em tempo hábil. Nesse contexto, o objetivo do trabalho é investigar uma solução exata, baseada em Branch-And-Bound e duas soluções aproximativas, o Twice-Around-The-Three e o algoritmo de Christofides.

2. Implementações

Os algoritmos foram escritos em Python com a ajuda das bibliotecas NumPy e NetworkX para gerar os grafos e calcular o Minimum Spanning Tree (MST) e o Matching de Peso Mínimo.

2.1. Twice-Around-The-Three

O algoritmo Twice-Around-The-Three (TAT) é descrito pelos passos:

- Gere uma Minimum Spanning Tree (MST) a partir do grafo inicial
- Duplique cada aresta da MST para torná-la um grafo euleriano
- Adicione atalhos à MST, para cada aresta v-w-x remova v-w e w-x e adicione v-x, para gerar atalhos
- Adicione uma aresta do vértice inicial ao final para criar um circuito hamiltoniano

Sabe-se que é um algoritmo 2-aproximativo, ou seja, a solução é no máximo duas vezes pior que a solução ótima.

2.2. Christofides

O algoritmo de Christofides consiste nos passos:

- Gere uma MST X a partir do grafo inicial
- Construa o subgrafo completo induzido S no grafo original pelos os vértices de grau ímpar de X
- Encontre o Minimum Matching de S e adicione suas arestas à X
- Encontre o passeio euleriano
- Adicione atalhos, como no algoritmo TAT
- Adicione uma aresta entre o vértice inicial e o final para gerar um circuito hamiltoniano

O algoritmo de Cristofides é 1.5-aproximativo

2.3. Branch-And-Bound

O algoritmo de Branch-And-Bound divide o problema em subproblemas e utiliza limites para podar as soluções inviáveis

Por ser uma solução exata, demora um tempo consideravelmente maior que os anteriores para finalizar, logo algumas heurísticas foram implementadas para que terminasse em um tempo inferior à 30 minutos para as instâncias definidas nas especificações do trabalho.

A implementação criada utiliza o resultado do algoritmo de Christofides como solução viável inicial. Além disso, o bound utilizado é o seguinte:

- Some para cada vértice não-visitado os pesos das suas duas arestas incidentes com os menores pesos.
- Divida o resultado anterior por 2, para compensar pelo fato de que cada aresta pode ter sido somada duas vezes.
- Utilize esse bound para realizar podas na árvore de busca do algoritmo.

A estratégia de busca empregada foi a DFS, que busca encontrar uma folha rapidamente e possivelmente encontrar soluções viáveis com custos menores, o que permitiria realizar mais podas e mais cedo.

Entretanto, mesmo com essas estratégias não foi possível resolver nenhuma das instâncias usando o algoritmo de Branch-and-bound. A instância com o menor número de vértices (eil51.tsp) tem 51 vértices, o que leva a um espaço de busca da ordem de $51! \approx 10^{66}$ possibilidades.

Os resultados ótimos das instâncias analisadas foram retirados do seguinte link (fornecido na especificação deste trabalho):

http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html

3. Experimentos

Na seção de experimentos, iremos comparar o tempo e a qualidade da solução de algumas instâncias definidas na especificação do trabalho.

3.1. Valor

Descrição	**eil51**	**st70**	**rat99**
Ótimo	426	675	1211
TAT	594.0019	864.5307	1724.2229
TAT/Ótimo	1.3949	1.2833	1.4238
Cristofides	486.8249	759.1621	1367.4320
Cristofides/Ótimo	1.1428	1.1292	1.1292

Table 1. Valores e Razões combinados para eil51, st70 e rat99

Descrição	pr299	fnl4461
Ótimo	48191	182566
TAT	67061.8659	252012.60333045214
TAT/Ótimo	1.3913	1.38039176698
Cristofides	52968.2198	ND
Cristofides/Ótimo	1.0996	ND

Table 2. Valores e Razões combinados para pr299 e fnl4461

3.2. Tempo

Descrição	eil51	st70	rat99
TAT	0.00396	0.00598	0.01035
Cristofides	0.01144	0.02760	0.03997

Table 3. Tempo de execução para TAT e Cristofides para eil51, st70 e rat99

Descrição	pr299	fnl4461
TAT	0.08899	52.67640
Cristofides	0.67727	ND

Table 4. Tempo de execução para TAT e Cristofides para pr299 e fnl4461

4. Conclusões

Não existe uma forma correta e rápida de se resolver o problema por ele ser NP-Difícil. É preciso analisar cada cenário para decidir qual algoritmo será aplicado.

Caso o valor ótimo seja um requisito necessário, o Branch-And-Bound é a mellhor escolha dentre os algoritmos citados, desde que o tamanho da instância não seja muito grande. Além disso, são necessárias bounds e heurísticas muito boas para que essa solução funcione.

Como esse problema é NP-Difícil e tem muitas aplicações práticas, é preciso analisar outras variáveis, como o tamanho do problema. Nesse cenário, algoritmos aproximativos, como os dois exemplos analisados neste trabalho, são muito úteis por serem mais rápidos e fornecerem uma solução razoavelmente boa.

Entretanto, mesmo os algoritmos aproximativos tem suas limitações. A título de exemplo, a instância fnl4461 levou pouco mais de 52 segundos no TAT, mas ficou mais de duas horas executando o Christofides e não foi suficiente.