Trabalho Prático 2

Algoritmos 2

João Antonio Oliveira Pedrosa
Matrícula: 2019006752

[1] Universidade Federal de Minas Gerais
Belo Horizonte - MG - Brasil

joao.pedrosa@dcc.ufmg.br

1. Introdução

O problema do Caixeiro Viajante é um dos problemas NP-Completo mais conhecidos. Apesar de ser computacionalmente difícil, ele possui aplicações em diversas áreas e, portanto, é importante desenvolver técnicas para resolvê-lo. Nesse trabalho prático, foi proposto o desenvolvimento de 3 algoritmos para a resolução do Problema do Caixeiro Viajante Métrico: Twice Around the Tree, Christofides e Branch and Bound. Dois deles possuem tempo polinomial e solução aproximada, enquanto o outro possui tempo exponencial e solução exata.

2. Implementação

Um Jupyter Notebook contendo a implementação utilizada, bem como testes pode ser encontrado no link:

`https://github.com/mrcolorblind/tps_alg2`

Para a resolução do problema, uma classe para cada algoritmo foi criada:

2.1. Twice Around The Tree

O algoritmo segue os seguintes passos para encontrar uma solução aproximada para o Problema do Caixeiro Viajante Métrico:

- Gera uma lista de adjacência contendo o grafo completo de todos os pontos no plano, ou seja, a partir de cada ponto, geramos uma aresta para todos os outros pontos, com distância igual à distância entre os dois pontos no plano.
- Gera uma Árvore Geradora Mínima desse grafo.
- Registra a ordem de visita dos vértices em uma travessia da árvore. A distância percorrida durante essa travessia é no máximo duas vezes maior do que a distância percorrida na solução ótima do problema.

Para essa classe, eu implementei manualmente o algoritmo de Kruskal e utilizei uma estrutura de *Disjoint Set Union* para manter quais os componentes conexos de minha floresta enquanto gerava a árvore. O *Disjoint Set Union*, da maneira como foi implementado, possui complexidade igual à $\mathcal{O}(logn)$ para cada chamada. É possível fazer essa mesma estrutura com complexidade igual à Inversa de Ackermann, fazendo uso da técnica *Small to Large* juntamente com *Path Compression*, mas, como no caso do Kruskal já existe uma complexidade logarítmica na hora de ordenar as arestas, não julguei

necessário implementar ambas as técnicas e utilizei apenas *Path Compression*. Para a travessia da árvore, fiz uma simples busca em profundidade. Como a DFS possui tempo linear e o DSU possui tempo logarítmico, a complexidade do algoritmo inteiro fica, no fim das contas, limitada à uma simples ordenação das arestas. Já que geramos um grafo completo, a complexidade fica sendo igual a $\mathcal{O}(V^2 \log(V^2)$.

2.2. Algoritmo de Christofides

O algoritmo de Christofides é capaz de encontrar uma solução no máximo 1.5 vezes pior do que a solução ótima do Problema do Caixeiro Viajante Métrico. Para tal, ele realiza os seguintes passos:

- Encontra uma árvore geradora mínima do grafo completo de todos os pontos do plano.
- Encontra as arestas pertencentes ao Matching Máximo de Custo Mínimo do subgrafo completo induzido pelos vértices que possuem grau ímpar na Árvore Geradora Mínima.
- Une essas arestas encontradas com as arestas da Árvore Geradora Mínima em um grafo G.
- Encontra um Circuito Euleriano em G.
- A solução encontrada para o TSP são os vértices ordenados na ordem em que aparecem nas arestas do Circuito Euleriano. Vértices repetidos são ignorados. Cada vértice é introduzido na solução apenas na primeira vez em que aparece.

Para essa classe, eu utilizei os métodos da biblioteca **Networkx**, como orientado pelo professor. Com a implementação da biblioteca, encontrar a Árvore Geradora Mínima possui custo $O(E \log E)$ e encontrar o Matching Máximo de Custo Mínimo possui custo $\mathcal{O}(V^3)$. Como o número de arestas do grafo é no máximo V^2, o algoritmo, no fim das contas, possui complexidade igual à $\mathcal{O}(V^3)$.

2.3. Branch-and-Bound

O algoritmo segue o procedimento padrão de todos os algoritmos que utilizam essa técnica: uma solução recursiva brute-force que mantém, durante sua execução, a melhor estimativa que temos de solução possível de ser encontrada e, de acordo com essa estimativa, interrompe as recursões infrutíferas, que não tem chance de chegar em um bom resultado. A estimativa de custo para a recursão atual é feita levando em conta as duas arestas mais próximas de cada vértice que ainda não foi adicionado à solução. Como no pior caso podemos visitar todos os possíveis $V!$ circuitos, a complexidade desse algoritmo fica sendo $\mathcal{O}(V!)$. É válido ressaltar, entretanto, que no caso médio, são feitas muito menos operações do que isso.

Algoritmo	**Número de Pontos**							
	8	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
Twice Around	0ms	1ms	4ms	12ms	41ms	188ms	890ms	5s
Christofides	0ms	6ms	9ms	38ms	211ms	2.25s	11s	1m53s
Branch and Bound	4ms	17s	N/A	N/A	N/A	N/A	N/A	N/A

3. Testes

3.1. Tempo

O algoritmo Branch-and-Bound, mesmo com as otimizações, se mostrou lento demais para ser utilizado na maioria dos casos e só pode ser executado em casos com um número de pontos ≤ 16. Segue uma tabela com o tempo médio de execução de cada algoritmo de acordo com o número de pontos no plano:

3.2. Otimalidade

Em relação à otimalidade, o Twice Around the Tree tem uma solução, no pior caso, 2 vezes pior que a solução ótima enquanto o algoritmo de Christofides possui solução no máximo 1.5 pior do que a solução ótima. Um teste, utilizando 100000 amostras aleatórios foi conduzido, para analisar como essas soluções se comportam na prática. Nenhum caso foi encontrado aleatoriamente em que o Twice Around the Tree fosse mais do que 1.5 vezes pior que a solução ótima. Da mesma forma, o Algoritmo de Christofides se mostrou, no pior caso, apenas 1.25 vezes pior do que a solução ótima. Segue um histograma mostrando a distribuição das otimalidades das soluções apresentadas por ambos os algoritmos:

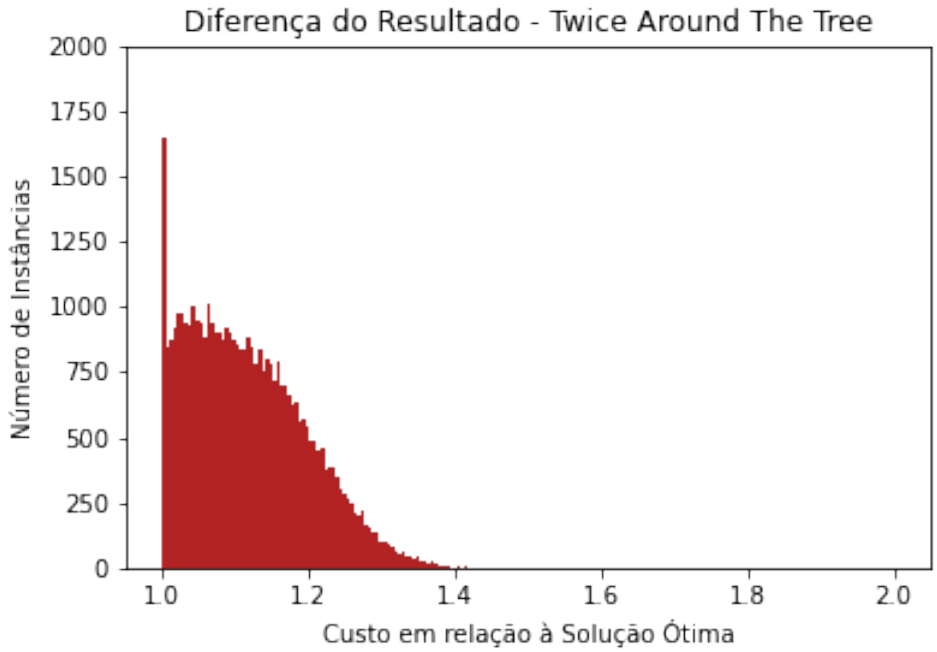

Figura 1. Distribuição dos resultados do algoritmo Twice Around the Tree.

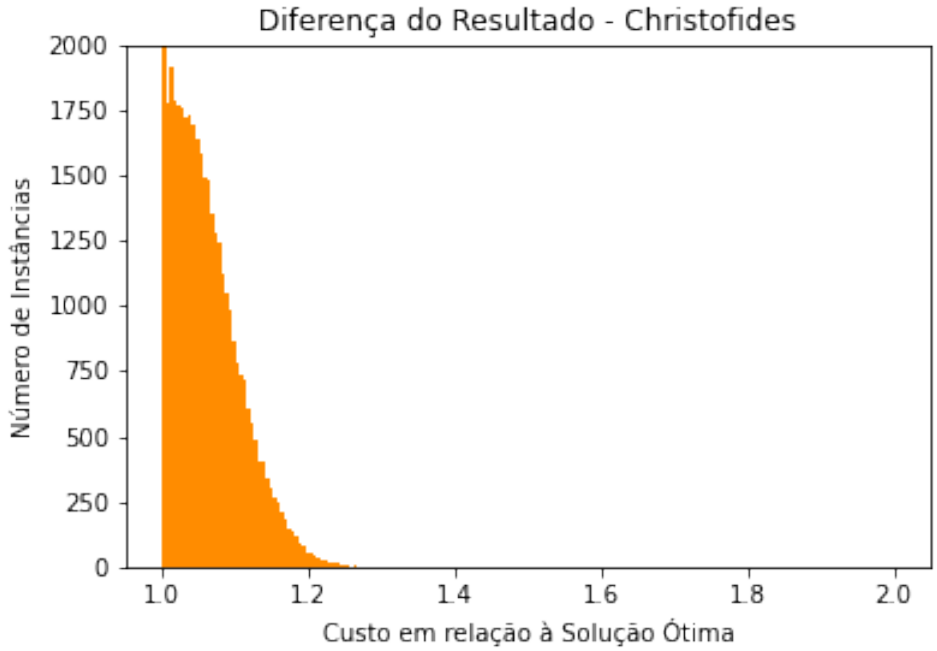

Figura 2. Distribuição dos resultados do Algoritmo de Christofides.

4. Conclusão

Os experimentos mostram que a dificuldade na resolução do Problema do Caixeiro Viajante reside na otimalidade. Encontrar uma solução ótima pode ser um problema extremamente difícil, entretanto, para soluções aproximadas, existem algoritmos muito eficientes. É possível perceber também que a Teoria da Complexidade pode ser um pouco enganosa. Mesmo com o pior caso sendo extremamente lento, na maioria das instâncias, a solução roda com um número absurdamente menor de operações do que o pior caso. É válido ressaltar, entretanto, que o fato do problema ser o caixeiro métrico, abre espaço para diversas otimizações e propriedades que não podem ser explorados no problema clássico.