# Trabalho Prático de Algorítimos II
# Traveling Salesman Problem

**Henrique R. S. Ferreira**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
31.270-901 – Belo Horizonte – MG – Brasil

***Resumo.*** *Neste trabalho vamos fazer uma abordagem prática do Problema do Caixeiro Viajante (TSP), que é bastante conhecido por sua dificuldade e, também, pela sua ampla aplicação no mundo atual. Este problema consiste em gerar um caminho de tamanho mínimo que passa em "n" cidades e depois retorna para o ponto inicial. No decorrer do trabalho, serão discutidas três implementações distintas que resolvem o problema com custos computacionais bem distintos, os algoritmos analisados são: Twice-Around-the-Tree, Christofídes e Branch and Bound.*

## 1. Introdução

Como dito acima, o TSP é um problema mundialmente famoso e de complexidade fatorial se considerarmos uma abordagem de força bruta, porém existem diversas maneiras de superar esses custos e resolver instâncias que jamais seriam resolvidas se mal implementadas.
Os algoritmos que vamos analisar são bem diferentes, tanto o Twice-Around-the-Tree quanto o Christofídes são aproximativos, ou seja, não necessariamente chegam no ótimo, porém eles têm um fator constante de proximidade desse ótimo. Já o Branch and Bound (BnB) é um algoritmo exato, portanto já podemos imaginar que ele será o mais difícil entre os três.
O problema do TSP também é trabalhado na ótica de Pesquisa Operacional (*Otimização*), e é utilizado uma variação do BnB, chamada de Branch and Cut, que mistura o BnB com o algoritmo dos Planos de Cortes, desenvolvido na década de 50, por Ralph Gomory.
O TSP é muito importante não só pela importância científica de se resolver um algoritmo NP-Difícil, mas pelo seu uso extenso em problemas do dia a dia. Grandes empresas, como a Amazon enfrentam esse empecilho diariamente, pois eles possuem centros de distribuição e caminhões de entrega que precisam ser o mais eficientes possível para realizarem as entregas e retornarem para os centros.
Neste trabalho são geradas instâncias de pontos aleatórios entre 16 e 1024, a partir desses pontos é construída uma matriz de adjacência que guarda a distância Euclidiana ou de Manhattan entre ambos os pontos. Com isso, usamos nossos algoritmos para resolverem o TSP para essas entradas.

## 2. Twice-Around-the-Tree

Este primeiro algoritmo é o mais rápido entre os outros estudados, ele consiste em, a partir de um grafo, gerar uma Árvore Geradora Mínima (MST) e rodar uma busca em profundidade, seguindo uma pré-ordem nos nós. Porém este é um algoritmo **2-Aproximado**, ou seja, no pior dos casos ele retorna uma solução com custo 2 vezes pior que a ótima.

APPROX-TSP-TOUR$(G, c)$

1 select a vertex $r \in G.V$ to be a "root" vertex
2 compute a minimum spanning tree $T$ for $G$ from root $r$
using MST-PRIM$(G, c, r)$
3 let $H$ be a list of vertices, ordered according to when they are first visited
in a preorder tree walk of $T$
4 **return** the hamiltonian cycle $H$

**Figure 1. Twice-Around-the-Tree**

Durante os testes, foi perceptível que essa era a implementação mais eficiente no que tange ao tempo de processamento. Para a entrada de 1024 cidades foram gastos 16.6 segundos para achar o valor ótimo, isso vale para uma implementação em python3 que não é a mais eficiente. Já para 16 cidades, o tempo é da casa de milésimos de segundos.
Porém como conseguimos falar o quão bom um algoritmo é? Para isso usamos a programação linear sem a restrição de integralidade para conseguirmos um limite inferior, também chamado de Held-Karp bound, com isso basta comparar a solução obtida com a esperada.
O custo desse algoritmo é dominado pela MST, que por sua vez é $O(n^2)$.

## 3. Christofídes

Este algoritmo é bem semelhante ao anterior, porém com fator de aproximação de 1.5, o que gera um custo maior. Ele consiste em computar a MST do grafo e um matching perfeito de peso mínimo a partir dos vertices de grau ímpar do grafo. Depois disso basta acrescentar as arestas desse matching na MST e rodar a DFS em pré-ordem.
Tanto esse, como o Twice-Around-the-Tree, foram implementados usando a biblioteca NetworkX de python3, a escolha foi devido à melhor documentação da mesma em relação ao Igraph, logo a implementação dos algoritimos para calcular a MST e o Matching, foram feitas a partir de bibliotecas terceiras.
Durantes os testes ele gastou 7 minutos para realizar a tarefa mais cara, a de 1024 cidades, entretanto como dito antes, a solução gerada por ele é bem mais próxima do ótimo, trata-se de um algoritmo com custo $O(n^3)$.
Esse é considerado o algoritmo aproximativo com melhor fator de aproximação, porém existem estudos que mostram que para instâncias simétricas é possível obter um algoritmo 1.25 aproximado.

## 4. Branch and Bound

O BnB é o único destes que é exato, e por isso ele é o mais caro de todos, para a instância de 16 cidades eu fui capaz de rodar na minha maquina em 3 segundos, porém percebi que isso variava entre as entradas, ou seja, enquanto os outros algoritmos rodavam sempre em tempso similares para a mesma instância, esse ja poderia demorar mais ou menos dependendo do tanto que ele percorre na árvore. Já para a instância de 32 cidades, não fui capaz de rodar dentro dos 30 minutos. Isso acontece pelo fato de ele ser um algoritimo exponencial, já que ele gera uma árvore binaria de subproblemas.

Na visão de Pesquisa Operacional, ele tende a ser muito mais rapido devido ao Simplex que otimiza as operações, porem ele não deixa de ser exponencial. O algoritimo implementado seguiu a segunte implementação: Figura[2]
Contudo a implementação da função de *Bound* foi inteiramente baseada na ideia geral do algoritmo, ou seja, usamos um limite inferior que leva como calculo a soma das 2 arestas de menor peso por vértice, dividido por 2. Caso uma aresta seja forçada, por exemplo, a aresta "a-d", devemos inseri-la tanto no vértice "a", quanto no "d" e para a outras permanece o mesmo valor.

**Procedure** bnb-tsp(A,n)

```
root = (bound([0]), 0, 0, [0]);
queue = heap([root]);
best = ∞;
sol = ∅;
while queue ≠ ∅ do
    node = queue.pop();
    if node.level > n then
        if best > node.cost then best = node.cost; sol = node.s;
    end
    else if node.bound < best then
        if node.level < n then
            for k = 1 to n − 1 do
                if k ∉ node.s and A[node.s[−1]][k] ≠ ∞ and
                 bound(node.s ∪ {k}) < best then
                    queue.push((bound(node.s ∪ {k}), node.level +
                     1, node.cost + A[node.s[−1]][k], node.s ∪ {k}))
                end
            end
        end
        else if A[node.s[−1]][0] ≠ ∞ and bound(node.s ∪ {0}) < best
         and ∀i ∈ node.s then
            queue.push((bound(node.s ∪ {0}), node.level + 1, node.cost +
             A[node.s[−1]][0], node.s ∪ {0}))
        end
    end
end
```

**Figure 2. Branch and Bound**

## 5. Conclusões

Este trabalho foi bastante interessante do ponto de vista computacional, pois fui capaz de ver a batalha entre ter o ótimo ou ter uma solução boa o suficiente gastando pouco. No mais a análise de algoritmos foi bastante interessante para aperfeiçoar o conhecimento neles, e também de ver como outros algoritmos podem ser tão úteis para ajudar a solucionar problemas muito difíceis na teoria.

Isso mostra que mesmo que sabemos que um problema é NP-Hard, ou qualquer que seja a complexidade dele, pensar em maneiras de simplificar o problema e em condições que reduzem a complexidade, são a chave que temos nos dias atuais de resolvermos esses obstáculos.

## 6. Referências Bibliográficas

[Reinelt 2003], [Volgenant and Jonker 1982], [Christofides 1976], and [Deng et al. 2015].

## References


Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group.

Deng, Y., Liu, Y., and Zhou, D. (2015). An improved genetic algorithm with initial population strategy for symmetric tsp. *Mathematical Problems in Engineering*, 2015.

Reinelt, G. (2003). *The traveling salesman: computational solutions for TSP applications*, volume 840. Springer.

Volgenant, T. and Jonker, R. (1982). A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. *European Journal of Operational Research*, 9(1):83–89.