Análise de desempenho de algoritmos aproximativos no problema do caixeiro viajante
TP2 - Algoritmos 2

Gabriel A. C. Fadoul[1]

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

gabriel.fadoul@dcc.ufmg.br

Abstract. *This article describes the performance of approximate algorithms in solving the traveling salesman problem. Two algorithms were tested: Twice-Around-The-Tree and Christofides. Both performed well up to a certain number of nodes in the test case where excessive memory usage occurs and does not terminate execution. As a proposal for future work, it would be the use of alternative data structures that optimize the total memory used by the program and also the retrieval and storage in these same structures.*

Resumo. *Este artigo descreve a performance de algoritmos aproximativos na resolução do problema do caixeiro viajante. Dois algoritmos foram testados: o Twice-Around-The-Tree e o Christofides. Ambos tiveram um boa performance até um certo número de nós no caso de teste, onde ocorre um uso excessivo de memória e não termina a execução. Como proposta para trabalhos futuros, seria a utilização de estruturas de dados alternativas que otimizem a memória total usada pelo programa e também a recuperação e armazenamento nessas mesmas estruturas.*

1. Introdução

O problema do caixeiro viajante um clássico na área de Ciência da Computação, onde nos é dado um grafo e é preciso percorrer todos os vértices, sem repeti-los, minimizando a distância percorrida. Ele é um problema sabidamente NP-Difícil, e ao longo dos vários anos alguns algoritmos aproximativos foram desenvolvidos a fim de encontrar uma solução dentro de um tempo e discrepância com o valor ótimo razoável.

Esse trabalho se propõem a fazer uma análise quantitativa da performance de três desses algoritmos: Christofides, Twice-Around-The-Tree e Breach and Bound.

2. Implementação

Foram escolhidas algumas ferramentas para implementar o código dos algoritmos que queremos testar:

- **Python 3.10.4** Foi utilizado para implementar os algoritmos, juntamente com algumas outras bibliotecas próprias dele.;
- *NetworkX* Biblioteca utilizada para criação e manipulação de estruturas de grafos;
- **Numpy** Biblioteca que implementa estruturas de dados e funções performáticas;

- **Pandas** Biblioteca que implementa estruturas de dados, como o Dataframe, que me permitiu manipular os resultados para extraí-los para o formato CSV;
- **Math, Functools, Datetime e Memory Profile** Bibliotecas secundárias, algumas utilizadas para realizar cálculos de forma mais performática e outras para monitorar a performance dos algoritmos.
  Toda essa implementação foi testada num ambiente em nuvem do serviço Google Colab, que possui

3. Resultados

Para uma visualização completa dos dados, pode-se verificar o CSV de cada um dos algoritmos, dentro do repositório do trabalho. Analisando a performance de cada um dos algoritmos, foi possível notar que o Branch-And-Bound, independente das suas variações, não consegue encontrar o valor para nenhum caso de teste passado. Já os resultados do Christofides e do TATT, na maior parte forma satisfatórios, exceto quando o número de nós do grafo passava de aproximadamente 5K