



# RELATÓRIO DO TRABALHO FINAL DE ALGORITMOS EM GRAFOS

Componentes: Gustavo Celestino (201810249), Igor Mendes (201810939), Mateus Carvalho (201810245) e Pedro Antônio (201810557)

Professor Mayron Moreira

# 1. INTRODUÇÃO

Motivação:

"Considere um conjunto de regiões em uma cidade, compostas por vários pontos de interesse. A demanda de cada região deve ser atendida por um único ponto que faça parte dela, e cada ponto pertence a apenas uma região.

O nosso estudo de caso consiste em uma empresa de entregas cujo objetivo é atender a demanda de cada região pertencente a um conjunto de regiões, cada uma composta por consumidores, através de exatamente um consumidor da respectiva região. Para tanto, considerando um número limitado de V veículos com capacidade C (cada um), encontre uma roteirização de cada veículo que saia e retorne do único depósito, e minimize o custo total de deslocamento entre os consumidores, considerando todos os veículos."

Após estudos e discussões, o grupo rotulou o problema dado como um Problema do Caixeiro-Viajante Generalizado (ou Generalized TSP) com alterações, como por exemplo, a adição do vértice adicional "depósito" que não pertence a nenhuma região e o fato de haver uma demanda (fluxo) a ser atendida.

Para a resolução do problema, foi decidido utilizar conceitos de algoritmos para variados tipos de problema: TSP (grafos hamiltonianos), por ter que visitar todas as regiões e não poder visitar uma região mais de uma vez; caminho mínimo, por necessitar otimizar a distância percorrida para executar todas as entregas; e fluxo máximo, para executar todas as entregas.

A ideia geral do algoritmo será apresentada na próxima seção do presente documento.

# 2. ALGORITMO

A ideia geral do algoritmo é calcular a carga ideal (demanda) que cada veículo deve atender e buscar "vizinhos mais próximos" com base não somente na distância entre esses vértices mas também na demanda a ser atendida. Para essa busca são usados conceitos de busca em profundidade. Quando um veículo atinge a carga ideal retorna imediatamente para a garagem e outro veículo sai. Se ao fim de todas as rotas ainda houverem seções não atendidas, o algoritmo procura a seção mais próxima para cada faltante e depois qual vértice é melhor, inserindo-o no meio da rota.

O pseudocódigo das funções principais é apresentado a seguir:

```
obtemRotaParcial (veic)
        cargaIdeal <- demandaTotal/quantidadeDeVeiculos
        vertice <- último vertice visitado pelo veículo veic
        melhorVizinho <- melhor vizinho obtido pela heurística

        SE melhorVizinho EXISTE FAÇA
                melhorVizinho.secao.status <- visitado

                SE melhorVizinho.demanda + veic.demandaAtendida <= cargaIdeal FAÇA
                        veic.distancia <- veic.distancia + melhorVizinho.distancia
                        veic.carga <- veic.carga + melhorVizinho.demanda
                        veic.secoes <- veic.secoes U melhorVizinho.secao
                        veic.vertices = veic.vertices U melhorVizinho.vertice

                        obtemRotaParcial(veic)
        SENÃO
                veic.vertices = veic.vertices U [1]
                veic.distancia <- veic.distancia + distanciaDoUltimoVerticeAoCentro

                PARA CADA secao:
                        SE secao.status = visitado E secao pertence a veic.secoes FAÇA
                                secao.status <- inativo
                        ELSE
                                secao.status <- ativo
```

```
obtemRotas():
        PARA CADA veiculo FAÇA:
                obtemRotaParcial(veiculo)

        secoesNaoVisitadas <- [secao | secao.status = ativo]

        ENQUANTO tamanho(secoesNaoVisitadas) > 0 FAÇA
                secoesProximas <- obtemSecaoMaisProxima(secoesNaoVisitadas[0])
                secaoIncluida <- Falso
                num_veic <- 0

                ENQUANTO EXISTE secoesProximas E secaoIncluida = Falso FAÇA
                        ENQUANTO secoesProximas[0].secao PERTENCER A secoesNaoVisitadas
                                retira primeiro elemento de secoesProximas

                SE secoesProximas.secao PERTENCE A veiculo[num_veic].secoes FAÇA
                        SE veiculo[num_veic].carga + secoesNaoVisitadas[0].demanda > capacidadeDoVeiculo FAÇA
                                veiculo[num_veic].secoes = veiculo[num_veic].secoes U secoesNaoVisitadas[0]
                                veiculo[num_veic].vertices = veiculo[num_veic].vertices U secoesProximas[0].vertice
                                veiculo[num_veic].carga = veiculo[num_veic].carga + secoesNaoVisitadas[0].demanda

                                atualizaDistancia(num_veic)
                                secoesNaoVisitadas[0].status <- inativo
                                Retira primeiro elemento de secoesNaoVisitadas
                                secaoIncluida <- Verdadeiro
                SENÃO FAÇA
                        num_veic <- num_veic + 1
                        SE num_veic == qtdVeiculos FAÇA
                                num_veic <- 0
                                Retira primeiro elemento de secaoProxima
```

## 3. TESTES COMPUTACIONAIS

A tabela 1 mostra os valores da melhor solução conhecida e seu tempo em segundos (“bkv” e “tempo(s)”), da solução dada pelo algoritmo desenvolvido pelo grupo e seu tempo em segundos (“sol” e “tSol(s)”) e do desvio entre as duas soluções (“desvio”), calculado pela fórmula dada pelo professor: ((sol[i] − bkv[i]) / bkv[i]) × 100).

As soluções bkv foram obtidas a partir de um código C++11 e compilado utilizando o compilador GCC, versão 6.2 e uma máquina Intel Core i7-980 3.33GHz (1 thread). Já as soluções do grupo foram obtidas a partir de um código em Python3 e compilado utilizando o compilador python, versão 3.7.3 e uma máquina Intel Core i5-3337U 1.80GHz)

| Problema | GRUPO 1 | | | | |
|---|---|---|---|---|---|
| | **bkv** | **tempo(s)** | **Sol** | **tSol(s)** | **desvio** |
| **1** | 519 | 3 | 764,13 | 0 | 47,23 |
| **2** | 451 | 2 | 641,56 | 0 | 42,25 |
| **3** | 603 | 7 | 937,30 | 0,01 | 55,44 |
| **4** | 690 | 5 | 926,27 | 0,01 | 34,24 |
| **5** | 997 | 1440 | 1654,16 | 0,04 | 65,91 |
| **6** | 441 | 4 | 558,02 | 0,01 | 26,54 |
| **7** | 719 | 21600 | 974,25 | 0,05 | 35,50 |
| **8** | 659 | 21600 | 1145,21 | 0,14 | 73,78 |
| **9** | 405 | 314 | 622,17 | 0,02 | 53,62 |
| **10** | 455 | 3280 | 837,76 | 0,04 | 84,12 |
| | **GRUPO 2** | | | | |
| | **bkv** | **tempo(s)** | **Sol** | **tSol(s)** | **Desvio** |
| **11** | 386 | 1 | 577,46 | 0 | 49,60 |
| **12** | 315 | 0 | 431,68 | 0 | 37,04 |
| **13** | 599 | 3 | 865,66 | 0,01 | 44,52 |
| **14** | 452 | 3 | 657,15 | 0,01 | 45,39 |
| **15** | 117 | 0 | 150,01 | 0 | 28,22 |
| **16** | 117 | 0 | 165,87 | 0 | 41,77 |
| **17** | 292 | 1 | 468,96 | 0,01 | 60,60 |
| **18** | 309 | 8 | 477,25 | 0 | 54,45 |
| **19** | 385 | 9 | 594,28 | 0,01 | 54,36 |
| **20** | 370 | 2853 | 716,54 | 0,02 | 93,66 |
| **21** | 3249 | 21600 | 5765,01 | 0,34 | 77,44 |
| **22** | 2476 | 21600 | 4628,40 | 0,18 | 86,93 |

## 4. CONCLUSÃO

O grupo obteve resultados um pouco piores dos que eram esperados durante o planejamento do algoritmo. O desvio médio das soluções foi de 54,21%, alto em comparação com o esperado, porém os tempos de execução para todos os problemas foram baixíssimos.

Na visão do grupo, o desempenho do algoritmo deve-se, principalmente, ao método de calcular o custo benefício para encontrar o melhor vizinho não só pela distância, mas também pelo fluxo necessário a ser carregado pelo veículo e e pela demanda da próxima seção. Entretanto, o grupo pensa que a função utilizada poderia ser modulada de forma diferente, e teve dificuldade em executar essa melhoria.

É importante ressaltar a importância do trabalho para o aprendizado da disciplina. A partir dele, criou-se o desafio de usar os conhecimentos de códigos e definições em um problema prático não trivial. Outro conhecimento adquirido durante o desafio foi o aprendizado da linguagem Python.

Durante o planejamento do código, o grupo obteve dificuldades em escolher os melhores métodos propostos durante o semestre para tentar alcançar uma solução ótima, como por exemplo, se o caminho aumentante seria definido por conceitos de busca em largura ou em profundidade. Durante a execução do código, um desafio foi a linguagem, uma vez que nenhum dos integrantes do grupo tiveram contato com Python3 antes do trabalho, porém a maior dificuldade foi pensar em formas de mudar a estratégia e maximizar a solução, uma vez que foram necessárias várias mudanças durante o processo.