Problema de Roteamento de Veículos Capacitado (CVRP)

Nome: Arthur Cavalcante Gomes Coelho
Matrícula: 1313099
Professora: Renatha Capua

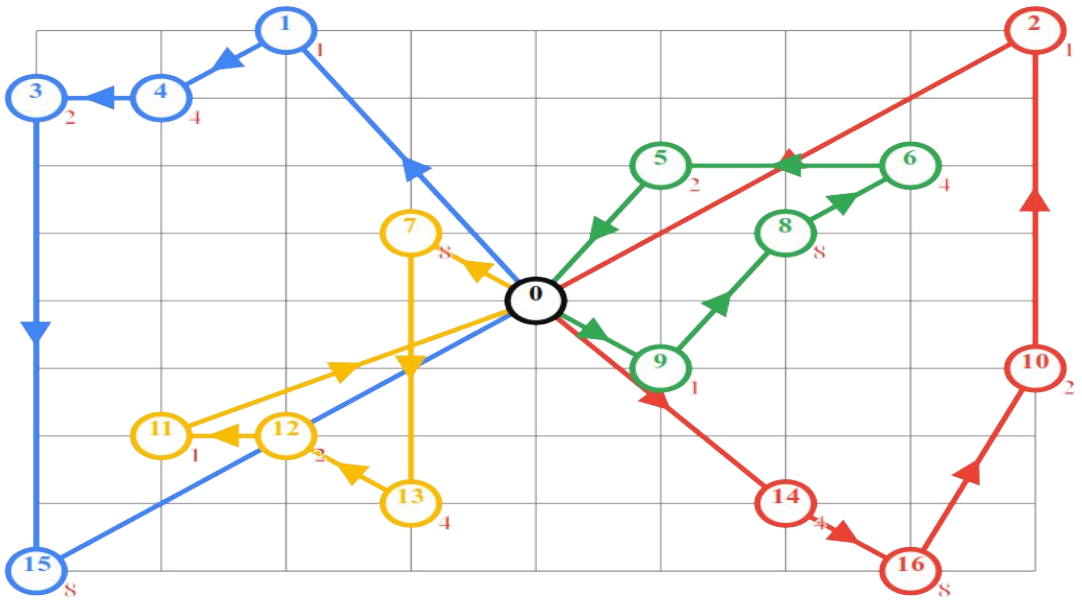
Introdução

O problema de roteamento de veículos é um dos mais estudados problemas na área da otimização combinatória. Consiste no atendimento de um conjunto de consumidores por intermédio de uma frota de veículos, que partem de um ou mais pontos (no atual projeto, foi tratado o caso de que todos os veículos partem do mesmo ponto) denominados depósitos. A restrição presente no PRVC é que cada veículo possui uma capacidade C (no caso do atual projeto, essa capacidade é uniforme para todos os veículos) e o somatório de todas as demandas dos consumidores atendidos por um veículo não pode ultrapassar essa capacidade C.

O PRV, apesar do seu enunciado relativamente simples, apresenta elevada complexidade computacional, pelo que é interessante como problema no teste de diversas heurísticas.

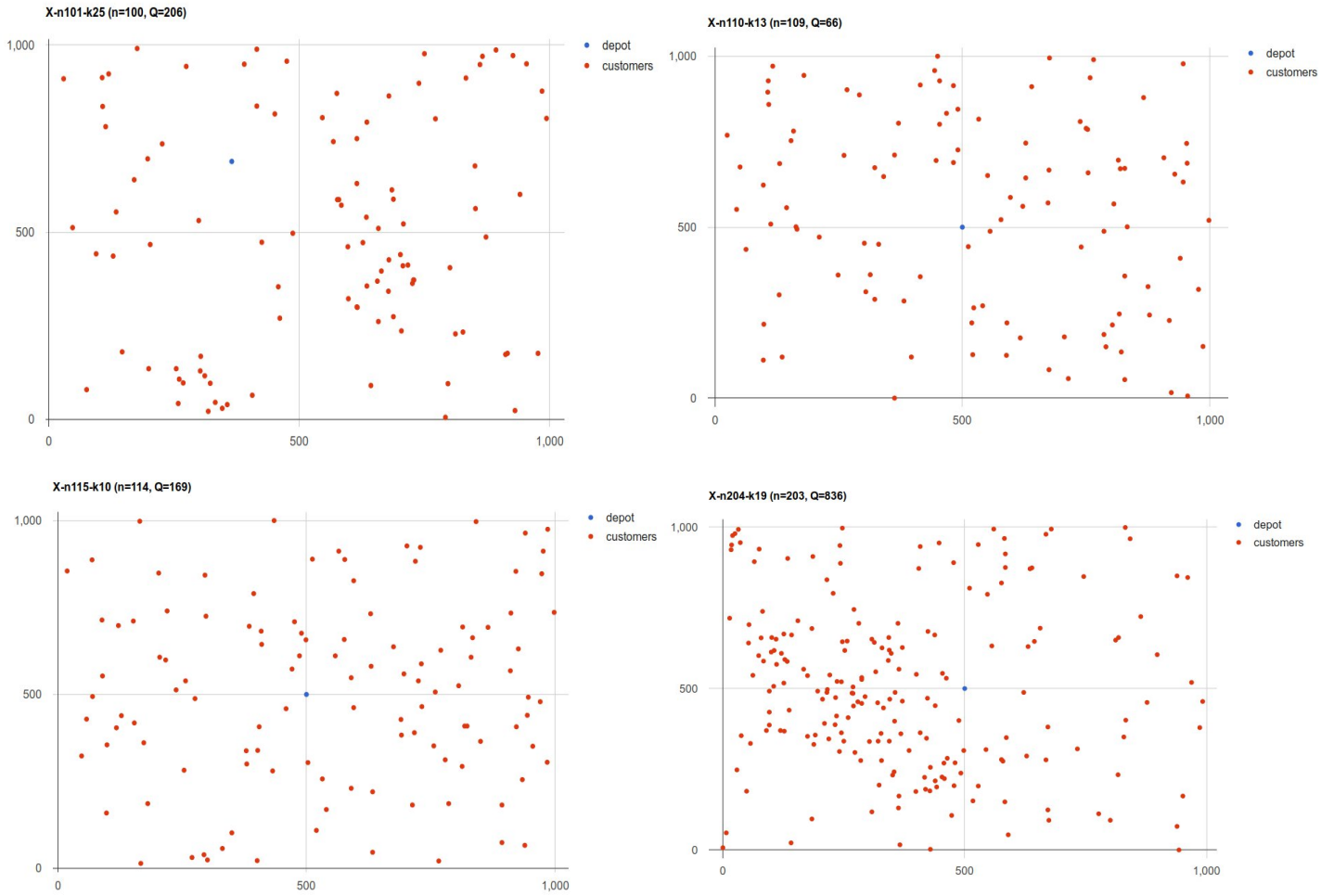
Na literatura científica, Dantzig e Ramser foram os primeiros autores a formular o PRV, em 1959, quando estudaram a aplicação real na distribuição de gasolina para estações de venda de combustíveis no trabalho .

Abaixo, podemos observar uma solução para o CVRP de um grafo que contém 16 consumidores e 1 ponto de partida dos veículos. Cada rota possui uma cor diferente, inicia e finaliza no ponto de partida 0.



Definição do Problema

No projeto desenvolvido nos foi dado quatro instâncias de um problema parecido com o da figura acima, contudo, a quantidade de consumidores, e, consequentemente, rotas possíveis, era consideravelmente maior. As figuras abaixo representam tais instâncias:



Foi disponibilizado, como representações desses grafos, quatro arquivos de texto de extensão .vrp que contém informações pertinentes quanto às instâncias do problema. Conclui-se que, primeiramente, é necessário se realizar o parsing do arquivo de texto referente à instância estudada, de modo que seja possível guardar as coordenadas dos nós, suas respectivas demandas além da capacidade uniforme dos veículos.

A forma como as informações são armazenadas no programa é uma questão de design, decidi que seria razoável criar uma classe Graph que armazenasse todas as informações obtidas no arquivo de texto. O construtor da classe recebe como único parâmetro a localização do arquivo .vrp, realiza o parsing e armazena as informações pertinentes em sua estrutura. Um dos métodos da classe Graph que é importante de se notar se chama: “initialize_distance_matrix();” , esse método é responsável por criar a matriz de distância entre todos os pontos da instância, matriz essa que é armazenada na estrutura da classe Graph.

Com a classe devidamente criada, progredimos para a resolução do problema.

Metodologia

Como o CVRP se trata de um problema NP-Difícil, uma solução exata, em que percorreríamos todas as possibilidades de caminhos do grafo se torna inviável. Com isto em mente, a metodologia de desenvolvimento para a resolução do problema escolhida foi a de utilização de heurísticas que possibilitem a chegada à uma solução relativamente próxima da solução ideal mais conhecida (BKS, best known solution).

- Solução Inicial
  - Para aplicarmos algoritmos que otimizem o caminho dos caminhões é preciso primeiramente de uma solução inicial. Pensei em uma heurística inicial relativamente simples, mas que ao mesmo tempo gerasse uma solução viável ao problema.
- Heurística Inicial
  - A heurística inicial escolhida é comumente chamada de “vizinho mais próximo”; partindo do nó inicial(S), vá para o nó imediatamente mais próximo, contanto que a demanda do nó mais próximo não ultrapasse a capacidade uniforme máxima do veículo. Capacidade essa que é subtraída pela demanda toda vez que visitamos um novo nó. Assim que a capacidade do nó mais próximo ultrapassar a capacidade do veículo encerramos a rota, não incluímos o nó que estoura a capacidade e começamos outra rota pelo nó inicial. Para este método funcionar é preciso guardar uma lista de nós já visitados, para que o nó inicial não os visite em uma nova rota.
  - Um problema imediato dessa heurística, no entanto, é que convergimos para o mínimo local, quando o ideal seria buscarmos o mínimo global por meio da avaliação das vizinhanças dos nós.
  - O resultado inicial em todas as instâncias se revelou bastante coerente com a qualidade da heurística inicial.
- Otimização
  - Primeira Melhoria
    - Com a solução inicial em mãos, progredimos para o próximo passo; buscar a melhora dessa solução por meio de heurísticas de otimização e buscas locais.
    - Um dos métodos mais famosos de otimização de caminhos se trata do k-opt, elaborado por Lin e Kernighan em 1973, Nesta proposta, *k* arcos são substituídos, no circuito, por outros *k* arcos com o objectivo de diminuir a distância total percorrida. Quanto maior for o valor de *k*, melhor é a precisão do método, mas maior é o esforço computacional.
    - Por limite de tempo e visando a simplicidade do algoritmo optei por implementar o algoritmo onde k=2, popularmente denominado 2-opt. O algoritmo utiliza as rotas obtidas com a solução inicial e as percorre novamente, buscando alguma melhora no caminho. O resultado encontrado foi novamente coerente, houve uma melhora na distância percorrida pelos veículos, porém, como o 2-opt percorre uma rota de cada vez, não há qualquer avaliação da vizinhança dos nós até o momento, portanto a melhora das rotas é limitada.

- Segunda Melhoria
  - Como já obtemos a melhoria das rotas criadas com a primeira heurística utilizando o 2-opt, seria interessante explorar a criação de outras rotas de modo que essas novas rotas melhorem o resultado da primeira heurística.
  - Porém seria interessante poder utilizar a solução inicial já encontrada para encontrar essas novas rotas. Com isto em mente, concluí que poderia utilizar as rotas já obtidas e realizar pequenas mudanças entre as mesmas e checar se houve melhora, se houver melhora, realizamos as mudanças nas devidas rotas. Dois métodos que realizam essa tarefa são: Delete and Insert e Exchange
  - Delete and Insert permite deletar um nó de alguma rota, e reinserir ele em qualquer outra rota, na melhor posição possível.
  - Exchange troca vértices de duas rotas diferentes, se houver melhora, substitui as rotas antigas pelas rotas novas.
  - Primeiramente utilizei a meta-heurística Delete and Insert para realizar alterações na rota obtida pela solução inicial. Com as rotas redefinidas pela Delete/Insert é o momento de explorar a troca de nós pela rota obtida com a meta-heurística Exchange. Depois de melhorarmos ainda mais a solução com Exchange, podemos reutilizar a busca local 2-opt para melhorar ainda mais a solução da junção das meta-heurísticas.
  - O resultado da junção das meta-heurísticas Delete/Insert e Exchange combinados com o 2-opt foi o esperado, obtivemos uma melhora considerável da solução inicial e da solução obtida apenas com o 2-opt.

O programa

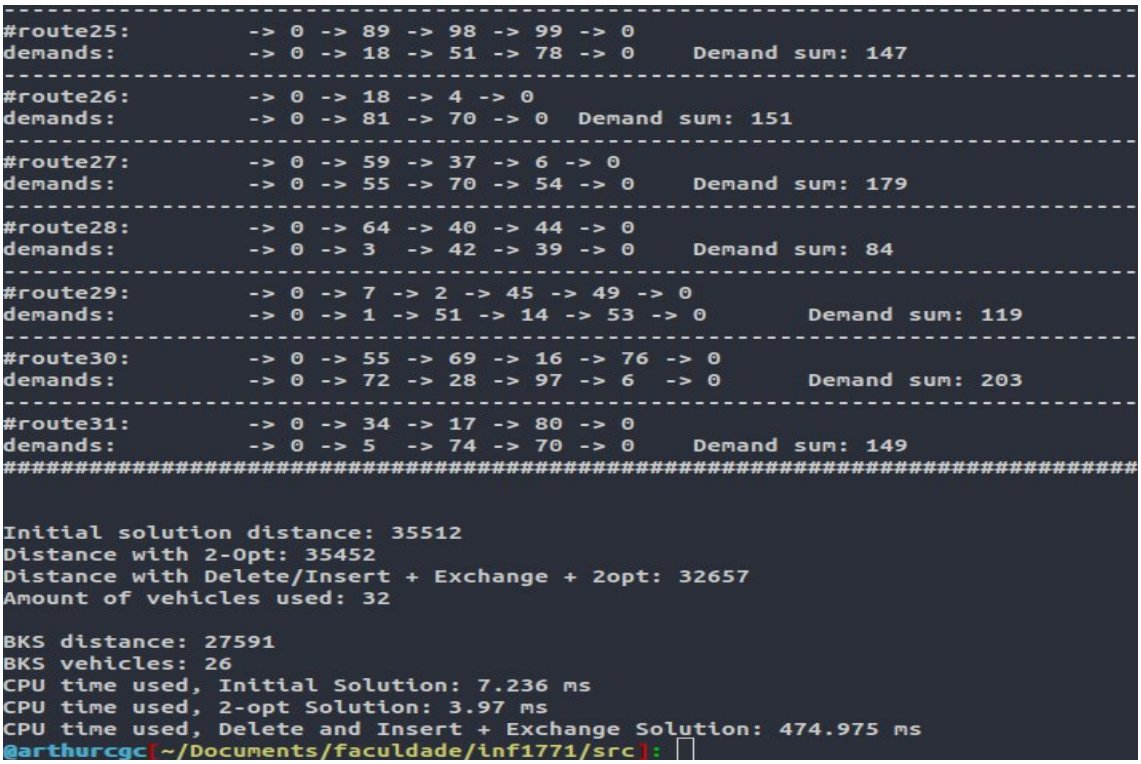
O programa funciona de forma simples, é preciso passar um par de argumentos na linha de comando, o primeiro contendo a localização em forma plain text do <nome_do_arquivo>.vrp o segundo se refere à localização do arquivo texto da BKS da solução <nome_do_arquivo>.sol. Foram criados scripts da shell do linux(<nome_da_instancia>.sh) para testar as 4 instâncias, assim como um script contendo as intruções de compilação do programa (compile.sh). Em todo caso, colocarei abaixo os comandos que devem ser rodados na pastar “src” do programa:

- compile.sh → faz o build do programa
  - /usr/bin/g++ -g main.cpp -o main graph.cpp initial.cpp optimization.cpp
- X-n101-k25.sh → roda a primeira instancia
  - ./main ../instances/X-n101-k25.vrp ../instances/X-n101-k25.sol
- X-n110-k13.sh →roda a segunda instancia
  - ./main ../instances/X-n110-k13.vrp ../instances/X-n110-k13.sol
- X-n115-k10.sh → roda a terceira instancia
  - ./main ../instances/X-n115-k10.vrp ../instances/X-n115-k10.sol
- X-n204-k19.sh → roda a quarta instancia

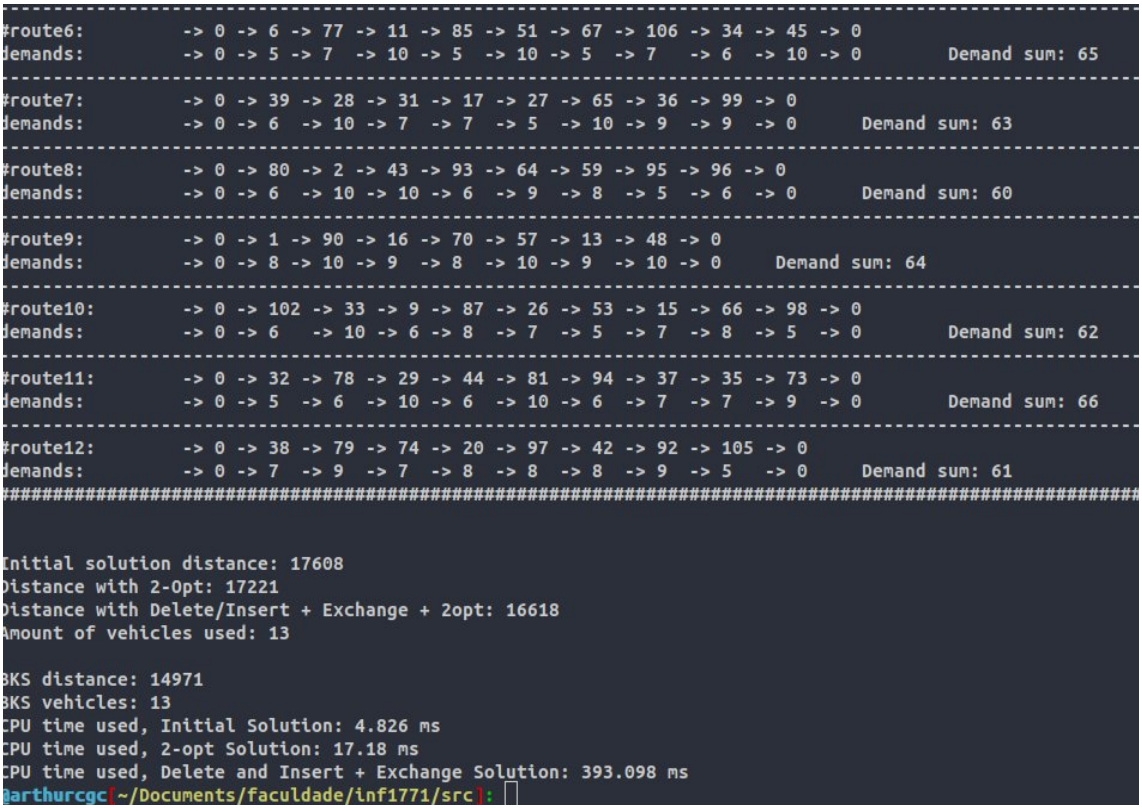
- ./main ../instances/X-n204-k19.vrp ../instances/X-n204-k19.sol

Resultados

É impresso o melhor conjunto de rotas obtido em cada execução.



./X-n101-k25.sh



./X-n110-k13.sh

```
#route6:        -> 0 -> 102 -> 24 -> 71 -> 37 -> 90 -> 39 -> 67 -> 40 -> 43 -> 68 -> 14 -> 8 -> 83 -> 75 -> 84 -> 0
demands:        -> 0 -> 6   -> 6  -> 3  -> 7  -> 2  -> 9  -> 6  -> 1  -> 2  -> 3  -> 10 -> 62 -> 9  -> 5  -> 3  -> 0   Demand sum: 134
-----------------------------------------------------------------------------------------------------------------------------------------
#route7:        -> 0 -> 53 -> 9 -> 42 -> 47 -> 79 -> 3 -> 0
demands:        -> 0 -> 2  -> 60 -> 2  -> 1  -> 10 -> 89 -> 0  Demand sum: 164
-----------------------------------------------------------------------------------------------------------------------------------------
#route8:        -> 0 -> 106 -> 73 -> 87 -> 86 -> 60 -> 48 -> 6 -> 108 -> 19 -> 0
demands:        -> 0 -> 10  -> 1  -> 6  -> 6  -> 1  -> 1  -> 87 -> 5   -> 1  -> 0      Demand sum: 118
-----------------------------------------------------------------------------------------------------------------------------------------
#route9:        -> 0 -> 13 -> 76 -> 55 -> 80 -> 97 -> 38 -> 12 -> 45 -> 96 -> 22 -> 51 -> 25 -> 0
demands:        -> 0 -> 6  -> 1  -> 3  -> 6  -> 5  -> 8  -> 79 -> 7  -> 8  -> 4  -> 4  -> 3  -> 0      Demand sum: 134
-----------------------------------------------------------------------------------------------------------------------------------------
#route10:       -> 0 -> 89 -> 5 -> 0
demands:        -> 0 -> 2  -> 74 -> 0  Demand sum: 76
#########################################################################################################################################


Initial solution distance: 15791
Distance with 2-Opt: 15438
Distance with Delete/Insert + Exchange + 2opt: 15043
Amount of vehicles used: 11

BKS distance: 12747
BKS vehicles: 10
CPU time used, Initial Solution: 23.108 ms
CPU time used, 2-opt Solution: 39.122 ms
CPU time used, Delete and Insert + Exchange Solution: 592.012 ms
@arthurcgc[~/Documents/faculdade/inf1771/src]: ./X-n115-k10.sh
```

./X-n115-k10.sh

```
#route13:       -> 0 -> 92 -> 85 -> 69 -> 6 -> 86 -> 29 -> 95 -> 67 -> 7 -> 84 -> 0
demands:        -> 0 -> 79 -> 96 -> 55 -> 76 -> 85 -> 74 -> 97 -> 94 -> 51 -> 64 -> 0  Demand sum: 771
-----------------------------------------------------------------------------------------------------------------------------------------
#route14:       -> 0 -> 32 -> 78 -> 100 -> 15 -> 34 -> 53 -> 111 -> 187 -> 9 -> 112 -> 171 -> 0
demands:        -> 0 -> 83 -> 60 -> 72  -> 93 -> 76 -> 81 -> 75  -> 55  -> 65 -> 51  -> 80  -> 0       Demand sum: 791
-----------------------------------------------------------------------------------------------------------------------------------------
#route15:       -> 0 -> 26 -> 19 -> 37 -> 60 -> 35 -> 82 -> 20 -> 46 -> 80 -> 27 -> 52 -> 0
demands:        -> 0 -> 66 -> 52 -> 54 -> 99 -> 62 -> 93 -> 86 -> 97 -> 98 -> 65 -> 56 -> 0    Demand sum: 828
-----------------------------------------------------------------------------------------------------------------------------------------
#route16:       -> 0 -> 115 -> 168 -> 175 -> 130 -> 161 -> 107 -> 178 -> 108 -> 196 -> 97 -> 0
demands:        -> 0 -> 71  -> 71  -> 87  -> 80  -> 94  -> 80  -> 81  -> 68  -> 83  -> 72 -> 0 Demand sum: 787
-----------------------------------------------------------------------------------------------------------------------------------------
#route17:       -> 0 -> 56 -> 40 -> 94 -> 11 -> 54 -> 41 -> 31 -> 59 -> 4 -> 23 -> 24 -> 28 -> 0
demands:        -> 0 -> 53 -> 59 -> 50 -> 63 -> 91 -> 73 -> 77 -> 94 -> 58 -> 59 -> 79 -> 70 -> 0      Demand sum: 826
-----------------------------------------------------------------------------------------------------------------------------------------
#route18:       -> 0 -> 147 -> 57 -> 163 -> 184 -> 180 -> 103 -> 158 -> 146 -> 191 -> 138 -> 76 -> 0
demands:        -> 0 -> 99  -> 82 -> 65  -> 51  -> 85  -> 54  -> 86  -> 74  -> 79  -> 50  -> 82 -> 0   Demand sum: 807
-----------------------------------------------------------------------------------------------------------------------------------------
#route19:       -> 0 -> 39 -> 33 -> 64 -> 88 -> 0
demands:        -> 0 -> 99 -> 50 -> 89 -> 95 -> 0      Demand sum: 333
#########################################################################################################################################


Initial solution distance: 24991
Distance with 2-Opt: 24365
Distance with Delete/Insert + Exchange + 2opt: 22985
Amount of vehicles used: 20

BKS distance: 19565
BKS vehicles: 19
CPU time used, Initial Solution: 26.62 ms
CPU time used, 2-opt Solution: 76.208 ms
CPU time used, Delete and Insert + Exchange Solution: 3134.97 ms
@arthurcgc[~/Documents/faculdade/inf1771/src]: ./X-n204-k19.sh
```

./X-n204-k19.sh

Conclusão

Os resultados obtidos estão coerentes com os métodos aplicados e com as observações feitas durante o relatório. Além da distância total e do tempo de CPU das soluções, também é impresso o caminho ótimo encontrado, é possível ver parte dele nas imagens dos resultados.

É interessante perceber a melhora considerável que se obtém quando introduzimos os métodos Delete/Insert e Exchange, podemos perceber que quando utilizamos esses métodos o efeito do mínimo local é minimizado, pois introduzimos variações às rotas.

Referências

- **Vidal T, Crainic TG, Gendreau M, Prins C (2013). "Heuristics for multi-attribute vehicle routing problems: A survey and synthesis". *European Journal of Operational Research*. 231 (1): 1–21. doi:10.1016/j.ejor.2013.02.053**

- **Oliveira, H.C.B.de; Vasconcelos, G.C. (2008). "A hybrid search method for the vehicle routing problem with time windows". *Annals of Operations Research*. 180: 125–144. doi:10.1007/s10479-008-0487-y**

- **Lin, Shen; Kernighan, B. W. (1973). "An Effective Heuristic Algorithm for the Traveling-Salesman Problem". *Operations Research*. 21 (2): 498–516. doi:10.1287/opre.21.2.498.**