universidade de aveiro deti departamento de eletrónica, telecomunicações e informática

Mini-Projeto de Otimização

Simulação e Otimização

Docentes Amaro Sousa e Nuno Lau
2022/2023

Mestrado em Engenharia Informática

Artur Romão, 98470
João Reis, 98474

Índice

1. Introdução

Este relatório tem como objetivo documentar as decisões na escolha de melhores parâmetros nos métodos implementados. Após essa justificação, serão indicados os resultados obtidos com esses os valores escolhidos. Para concluir, fazemos uma reflexão e uma comparação entre os métodos estudados, desde os seus resultados até aos seus tempos de execução.

2. GRASP (Greedy randomized adaptive search procedure)

2.1. Identificação e justificação dos melhores parâmetros

Para identificar qual seria o melhor valor para o parâmetro do método greedy randomized, k, a se utilizar, o programa foi executado para os valores (r=) 2, 3 e 4.

Os resultados para essas várias execuções demonstram que, comparando com o valor average das 10 exceções, o melhor valor de r é 3. Para r=2, o valor do average das soluções encontradas é de 156.325, para r=3 é de 155.968, e para r=4, é 156.142. Apesar de todos os valores serem muito próximos uns dos outros, o nosso critério de escolha é os melhores resultados obtidos, daí a nossa escolha.

Os resultados obtidos com este valor são analisados no próximo tópico.

2.2. Resultados obtidos

A seguir estão os resultados obtidos após executar este algoritmo meta-heurístico por 10 vezes, tal como representado na imagem seguinte. Para esta execução foi utilizado um parâmetro para o método greedy randomized, r, igual a 3.

```
>> GRASP
GRASP results among all 10 runs (execution time = 421.875556):
minimum = 155.465000
average = 155.968000
maximum = 157.270000
```

Fig. 1 - Resultados obtidos para o GRASP com r=3

Para uma visualização mais gráfica das soluções obtidas, os grafos com os SDN controllers escolhidos pelo algoritmo estão representados nas imagens seguintes. Apenas a solução com o valor do caminho curto médio (average shortest path) mínimo nas 10 execuções, neste caso, 155.465.

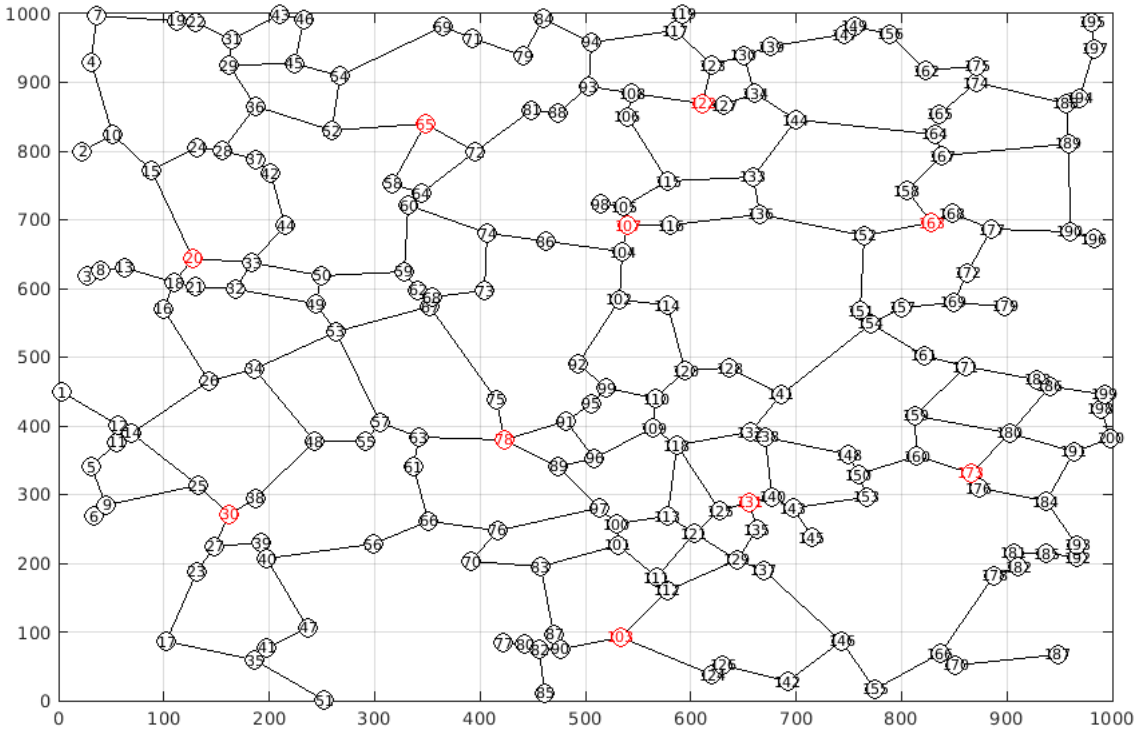

Fig 2 - Melhor solução obtida com um average shortest path de 155.465

3. GA (Genetic Algorithm)

3.1. Identificação e justificação dos melhores parâmetros

Para o algoritmo genético, os parâmetros que influenciam os resultados são o tamanho de uma população (no nosso programa, *P_size*) e a probabilidade de ocorrer mutação (no nosso programa, *q*). Analisamos os resultados obtidos para um *P_size* igual a 10, 15, 20, 25, 30, 40, 100, 200 e 500 e um *q* igual a 0.1 e 0.2. Em relação ao *q*, achamos que se usássemos valores maiores que 0.2, não estaríamos a ser coerentes com a "realidade" da genética, ou seja, seria uma probabilidade que na nossa ótica, já se torna grande para haver uma mutação.

Os melhores resultados obtidos para estes possíveis parâmetros são para um *q* = 0.2 e um *P_size* = 100, que estarão no próximo tópico com mais detalhe.

Passamos agora a justificar as nossas escolhas. Reparamos que quanto maior o *P_size*, ou seja, o tamanho da população, melhores resultados obtemos, porém o tempo de execução aumenta. O valor do average shortest path começa a estabilizar quando o tamanho da população é maior que 100. Como podemos observar nos seguintes gráficos.

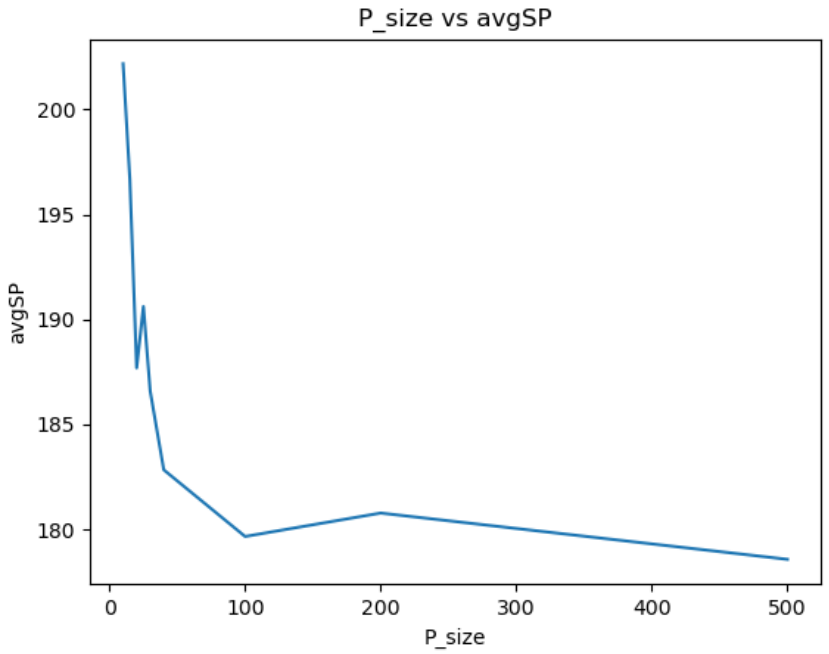

Fig. 3 - Influência do tamanho da população na procura pela melhor solução

Tal como referido anteriormente, é visível que quanto maior o tamanho da população melhores resultados obtemos, porém estabiliza perto P_size igual a 100. Este é o principal motivo para utilizarmos populações de tamanho 100.

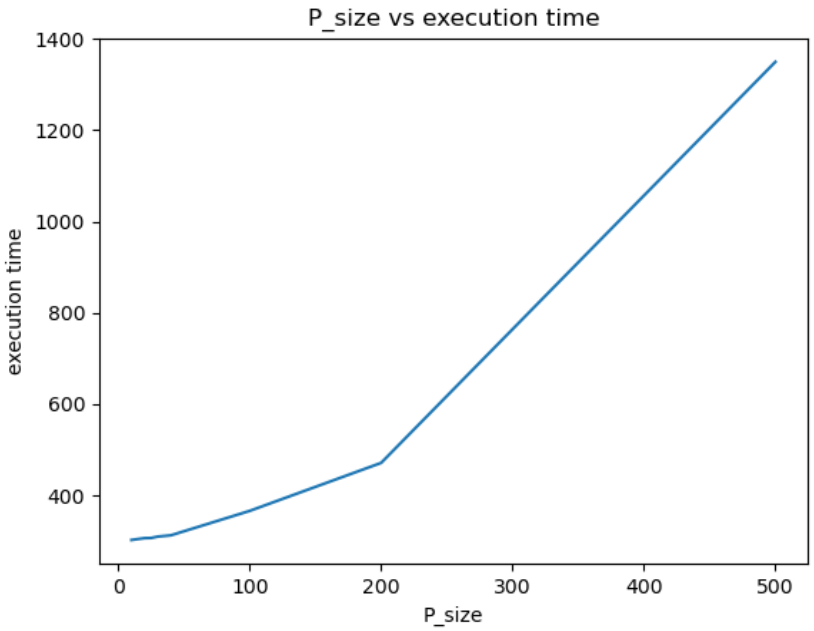

Fig. 4 - Tempo de execução mediante o aumento da população

É visível também que o tempo de execução é exponencial mediante o aumento do tamanho da população. Se olharmos para o P_size=100, onde o avgSP estabiliza, o tempo de execução não é demasiado alto, achamos adequado até.

Em relação ao parâmetro *q*, a probabilidade de mutação, foram analisados os resultados para um valor de 0.1 e de 0.2. Os resultados, isto é, o valor average das soluções encontradas, para *q*=0.1, é 180.179, e para *q*=0.2 é 177.657. Aplicando o critério de escolha do parâmetro com melhores resultados, o melhor valor de *q* é 0.2.

3.2. Resultados obtidos

A seguir estão os resultados obtidos após executar este algoritmo meta-heurístico por 10 vezes, tal como representado na imagem seguinte. Para esta execução foi utilizado os parâmetros *q* igual a 0.2 e *P_size* igual a 100.

```
>> GA
GA results among all 10 runs (execution time = 356.069795):
minimum = 170.535000
average = 177.657000
maximum = 187.405000
```

Fig. 5 - Resultados obtidos para o GA com *q*=0.1 e *P_size*=100

Para uma visualização mais gráfica das soluções obtidas, os grafos com os SDN controllers escolhidos pelo algoritmo estão representados nas imagens

seguintes. Apenas a solução com o valor do caminho curto médio (average shortest path) mínimo nas 10 execuções, neste caso, 170.535.

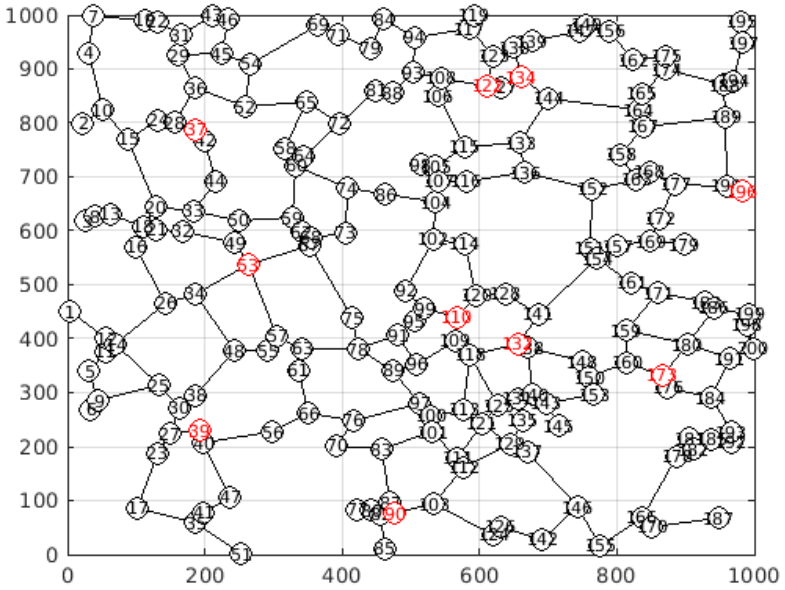

Fig. 6 - Melhor solução obtida com um average shortest path de 170.535

4. Exact Method (Integer Linear Programming)

4.1. Implementação e restrições a considerar

Este método exato é baseado em Integer Linear Programming (ILP). O objetivo deste exercício é usar o MATLAB para gerar um ficheiro *.lp* e, de seguida, correr esse mesmo ficheiro usando o IDE do *lpsolve*. Ao fim de um determinado tempo (*timeout*) o programa terá calculado uma solução ótima para o problema.

Passemos à explicação integral da construção do programa *.lp*. O objetivo do programa é **minimizar** o resultado da seguinte soma:

0 l1_1 + 622 l1_2 + … + 66 l200_199 + 0 l200_200

Em que 622 corresponde à distância entre o nó 1 e o nó 2, 66 à distância entre o nó 200 e o nó 199, etc, e l1_1, l1_2, …, correspondem a variáveis binárias das arestas (0 caso não seja considerada e 1 em caso contrário). No fundo, esta é a declaração do nosso objetivo, que no caso é minimizar o comprimento médio do caminho mais curto de cada *switch* para o seu *servidor* mais próximo.

Seguidamente, é necessário definir as nossas restrições, ou *constraints*. Assim sendo, temos:

1. Na solução devem ser selecionados exatamente 10 nós.
2. Um nó só pode estar conectado a um servidor.
3. Pares de nós cuja distância entre si seja superior a ***Cmax*** não devem ser considerados para fazer parte duma mesma solução.
4. O *servidor* atribuído a cada nó, tem que ser um nó *servidor*.

A implementação da primeira restrição é simples de explicar. Uma vez que cada nó se trata de uma variável binária (0 caso o nó não seja escolhido e 1 caso seja), a soma de todas as variáveis binárias correspondentes aos nós, tem que ser igual a 10.

A segunda restrição acaba por seguir a mesma lógica da anterior. Como cada aresta do grafo também corresponde a uma variável binária e cada nó só poderá estar ligado a 1 *servidor*, isto significa que a soma das variáveis binárias para aquele nó, terá que ser igual a 1.

Na terceira restrição, percorremos a matriz de distâncias e verificamos se encontramos alguma distância superior a ***Cmax***. Se tal acontecer, nunca poderemos escolher ambos os nós em simultâneo, pelo que recorremos a uma inequação para modelar a restrição: a soma das variáveis binárias referentes aos nós em questão terá que ser menor ou igual a 1, isto é, a solução terá apenas um desses nós ou nenhum deles.

Por fim, a última restrição, que assenta na lógica da figura seguinte, retirada do acetato 23 dos slides "*Exact Optimization Methods based on Integer Linear Programming*":

$$g_s^i \leq z_i \qquad ,s \in N, i \in N$$ ← The server assigned to each node must be a server node

Fig. 7 - Inequação representativa da quarta restrição

Em *lpsolve*, esta inequação é alcançada fazendo a diferença entre a variável binária duma determinada aresta e a variável binária do primeiro nó (no caso da figura, nó *i*). Essa diferença terá que ser menor ou igual a 0. Desta maneira, garantimos que o *server* atribuído a um determinado nó, é um nó *servidor*.

4.2. Resultados obtidos

É importante referir que, para efeitos de demonstração, o programa foi corrido com diferentes *timeouts*: 300s (5 minutos), 900s (15 minutos), 1800s (30 minutos) e 3600s (1 hora). No entanto, como se trata de um método exato, os resultados obtidos na execução de 3600s contêm os resultados das execuções

anteriores e, por esse motivo, para além de mostrarmos os resultados da execução de 5 minutos (obrigatórios), mostraremos também os resultados desta última *run*.

```
Relaxed solution       28868.4999999 after      11050 iter is B&B base.

Feasible solution              33498 after      13852 iter,         4 nodes (gap 16.0%)
Improved solution              33023 after      20701 iter,        42 nodes (gap 14.4%)
Improved solution              32991 after      28482 iter,       152 nodes (gap 14.3%)
Improved solution              32984 after      28882 iter,       171 nodes (gap 14.3%)
Improved solution              32828 after      29436 iter,       179 nodes (gap 13.7%)
Improved solution              32781 after      33113 iter,       246 nodes (gap 13.6%)
Improved solution              32769 after      33277 iter,       253 nodes (gap 13.5%)
Improved solution              32672 after      33303 iter,       254 nodes (gap 13.2%)
Improved solution              32625 after      36101 iter,       300 nodes (gap 13.0%)
Improved solution              32613 after      36245 iter,       307 nodes (gap 13.0%)
Improved solution              32592 after      38642 iter,       347 nodes (gap 12.9%)
Improved solution              32551 after      39141 iter,       352 nodes (gap 12.8%)
Improved solution              32539 after      39151 iter,       353 nodes (gap 12.7%)
Improved solution              32528 after      39584 iter,       359 nodes (gap 12.7%)
Improved solution              32516 after      40309 iter,       374 nodes (gap 12.6%)
Improved solution              32395 after      41379 iter,       385 nodes (gap 12.2%)
Improved solution              32383 after      41538 iter,       390 nodes (gap 12.2%)
Improved solution              32372 after      43623 iter,       426 nodes (gap 12.1%)
Improved solution              32360 after      45186 iter,       447 nodes (gap 12.1%)
Improved solution              32341 after      50338 iter,       553 nodes (gap 12.0%)
Improved solution              32317 after      68338 iter,       877 nodes (gap 11.9%)
Improved solution              32286 after      68594 iter,       885 nodes (gap 11.8%)
Improved solution              32263 after      73347 iter,       981 nodes (gap 11.8%)
Improved solution              32249 after      85764 iter,      1222 nodes (gap 11.7%)

lp_solve optimization was stopped due to time-out.

Optimal solution               32249 after     108821 iter,      1690 nodes (gap 11.7%).

Relative numeric accuracy ||*|| = 1.55431e-015

 MEMO: lp_solve version 5.5.2.11 for 32 bit OS, with 64 bit REAL variables.
      In the total iteration count 108821, 14531 (13.4%) were bound flips.
      There were 946 refactorizations, 0 triggered by time and 0 by density.
       ... on average 99.7 major pivots per refactorization.
      The largest [LUSOL v2.2.1.0] fact(B) had 99524 NZ entries, 1.0x largest basis.
      The maximum B&B level was 80, 0.0x MIP order, 44 at the optimal solution.
      The constraint matrix inf-norm is 1, with a dynamic range of 1.
      Time to load data was 10.133 seconds, presolve used 2.874 seconds,
       ... 297.139 seconds in simplex solver, in total 310.146 seconds.
```

Fig. 8 - Resultados obtidos para a execução de 300 segundos (5 minutos)

Ao analisarmos a figura acima, conseguimos perceber que o objetivo da solução ótima encontrada (soma de todas as variáveis binárias multiplicadas pela respetiva distância entre nós), para o tempo de 300s, corresponde a 32249. Ora, para obtermos a distância média de um nó para o respetivo *servidor*, temos que dividir este valor pelo número total de nós (200). Assim sendo, obtemos o valor 32249 / 200 = 161.245.

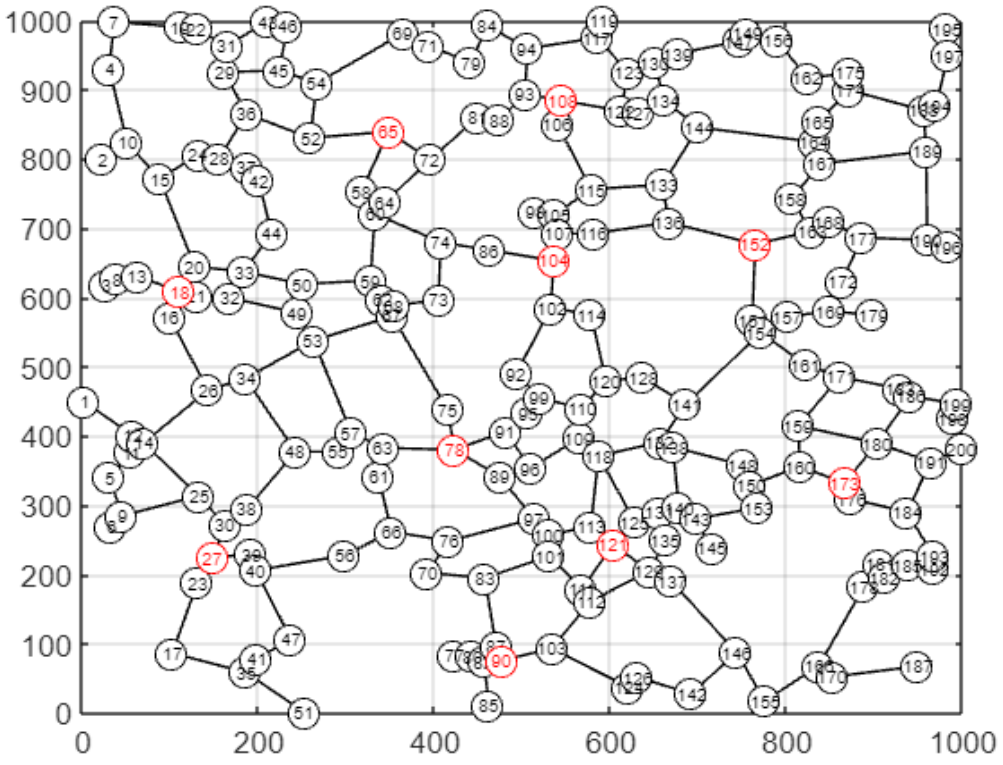

Fig. 9 - Melhor solução obtida com um average shortest path de 161.245 (*run* de 5 minutos)

```
Relaxed solution       28868.4999999 after      11050 iter is B&B base.

Feasible solution              33498 after      13852 iter,         4 nodes (gap 16.0%)
Improved solution              33023 after      20701 iter,        42 nodes (gap 14.4%)
Improved solution              32991 after      28482 iter,       152 nodes (gap 14.3%)
Improved solution              32984 after      28882 iter,       171 nodes (gap 14.3%)
Improved solution              32828 after      29436 iter,       179 nodes (gap 13.7%)
Improved solution              32781 after      33113 iter,       246 nodes (gap 13.6%)
Improved solution              32769 after      33277 iter,       253 nodes (gap 13.5%)
Improved solution              32672 after      33303 iter,       254 nodes (gap 13.2%)
Improved solution              32625 after      36101 iter,       300 nodes (gap 13.0%)
Improved solution              32613 after      36245 iter,       307 nodes (gap 13.0%)
Improved solution              32592 after      38642 iter,       347 nodes (gap 12.9%)
Improved solution              32551 after      39141 iter,       352 nodes (gap 12.8%)
Improved solution              32539 after      39151 iter,       353 nodes (gap 12.7%)
Improved solution              32528 after      39584 iter,       359 nodes (gap 12.7%)
Improved solution              32516 after      40309 iter,       374 nodes (gap 12.6%)
Improved solution              32395 after      41379 iter,       385 nodes (gap 12.2%)
Improved solution              32383 after      41538 iter,       390 nodes (gap 12.2%)
Improved solution              32372 after      43623 iter,       426 nodes (gap 12.1%)
Improved solution              32360 after      45186 iter,       447 nodes (gap 12.1%)
Improved solution              32341 after      50338 iter,       553 nodes (gap 12.0%)
Improved solution              32317 after      68338 iter,       877 nodes (gap 11.9%)
Improved solution              32286 after      68594 iter,       885 nodes (gap 11.8%)
Improved solution              32263 after      73347 iter,       981 nodes (gap 11.8%)
Improved solution              32249 after      85764 iter,      1222 nodes (gap 11.7%)
Improved solution              32243 after     327350 iter,      5995 nodes (gap 11.7%)
Improved solution              32130 after     796042 iter,     14335 nodes (gap 11.3%)
Improved solution              31987 after     798966 iter,     14351 nodes (gap 10.8%)
Improved solution              31877 after     799085 iter,     14353 nodes (gap 10.4%)
Improved solution              31858 after     799502 iter,     14363 nodes (gap 10.4%)
Improved solution              31857 after     799525 iter,     14364 nodes (gap 10.3%)
Improved solution              31856 after     799549 iter,     14365 nodes (gap 10.3%)
Improved solution              31855 after     799573 iter,     14366 nodes (gap 10.3%)
Improved solution              31831 after     799575 iter,     14367 nodes (gap 10.3%)
Improved solution              31773 after     799629 iter,     14368 nodes (gap 10.1%)
Improved solution              31622 after     803044 iter,     14422 nodes (gap 9.5%)
Improved solution              31466 after     803361 iter,     14431 nodes (gap 9.0%)
Improved solution              31310 after     804892 iter,     14460 nodes (gap 8.5%)

lp_solve optimization was stopped due to time-out.

Optimal solution               31310 after    1467835 iter,     26603 nodes (gap 8.5%).

Relative numeric accuracy ||*|| = 4.44089e-016

 MEMO: lp_solve version 5.5.2.11 for 32 bit OS, with 64 bit REAL variables.
      In the total iteration count 1467835, 170059 (11.6%) were bound flips.
      There were 13889 refactorizations, 0 triggered by time and 0 by density.
       ... on average 93.4 major pivots per refactorization.
      The largest [LUSOL v2.2.1.0] fact(B) had 99601 NZ entries, 1.0x largest basis.
      The maximum B&B level was 114, 0.0x MIP order, 36 at the optimal solution.
      The constraint matrix inf-norm is 1, with a dynamic range of 1.
      Time to load data was 11.933 seconds, presolve used 0.059 seconds,
       ... 3600.432 seconds in simplex solver, in total 3612.424 seconds.
```

Fig. 10 - Resultados obtidos para a execução de 3600 segundos (1 hora)

Seguindo a lógica anterior, para obtermos a distância média de um nó para o respetivo *servidor*, temos que dividir este valor pelo número total de nós (200). Assim sendo, obtemos o valor 31310 / 200 = 156.55.

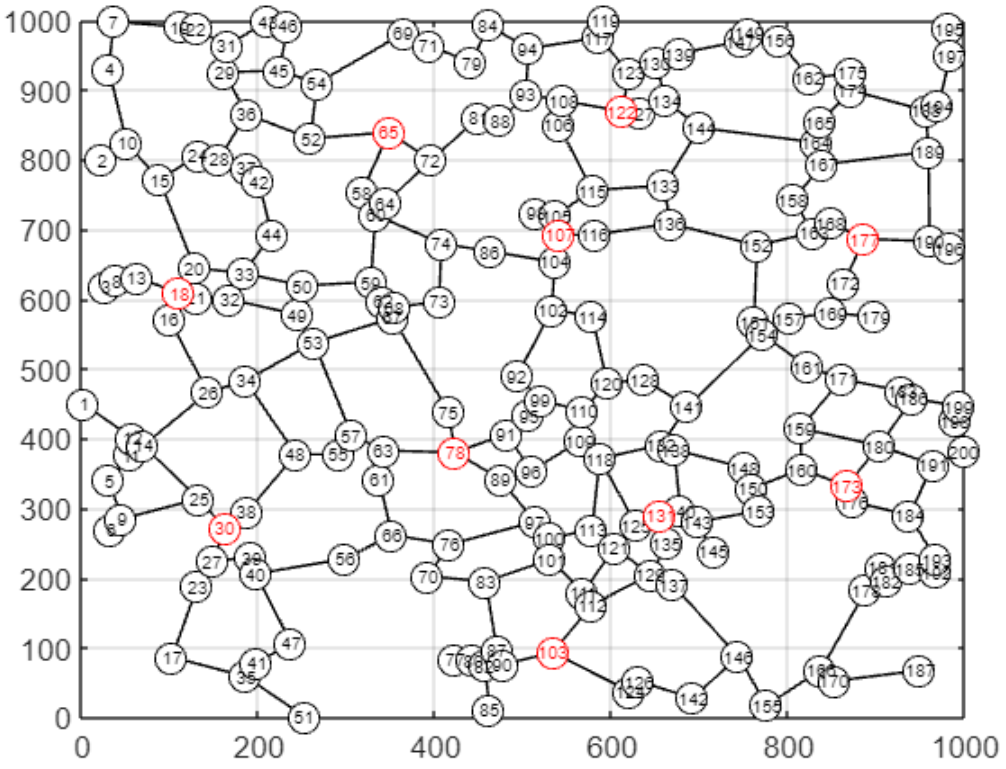

Fig. 11 - Melhor solução obtida com um average shortest path de 156.55 (*run* de 1 hora)

5. Comparação dos resultados

Em relação à descoberta de uma melhor solução, o método meta-heurístico GRASP é mais eficiente do que o método GA, uma vez que o valor average do shortest path das melhores soluções encontradas é menor.

Relativamente ao tempo de execução do GRASP foi de 421.88 segundos e o do GA foi de 339.47 segundos. Portanto, verificamos que o melhor método em questões de tempo de execução é o Genetic Algorithm.

Comparando também o esquema gráfico do grafo com os SDN controllers, figuras 2 e 4, repara-se que a solução do GRASP é sensivelmente melhor do que a do GA, uma vez que os SDN controllers estão mais espalhados pelo grafo.

Concluímos que o método GRASP se enquadra melhor na parte de encontrar uma solução mais próxima da exata.

Comparando os métodos anteriores com o método exato, tendo em conta que este último foi corrido para 5 minutos, podemos colocá-lo entre os dois métodos heurísticos em termos de qualidade de soluções ótimas encontradas (GRASP - 155.465, ILP - 161.245, GA - 170.535). Ainda assim, é percetível que o método GRASP se destaca dos restantes, uma vez que, em apenas 421.88 segundos, conseguiu encontrar uma solução melhor que a encontrada na *run* de 1 hora do ILP (GRASP - 155.465, ILP - 156.55).

6. Conclusão

Em conclusão, este documento apresenta uma análise dos resultados obtidos através da aplicação de diferentes métodos meta-heurísticos e de otimização num problema específico. Os métodos utilizados foram o Greedy Randomized Adaptive Search Procedure (GRASP), o Algoritmo Genético (GA) e o Método Exato baseado em Programação Linear Inteira (ILP).

Para o método GRASP, foi realizado um estudo para determinar o melhor valor para o parâmetro r, considerando os valores 2, 3 e 4. Os resultados indicaram que o valor ótimo de r é 3, com um average shortest path de 155.968.

No caso do Algoritmo Genético, os parâmetros investigados foram o tamanho da população (*P_size*) e a probabilidade de ocorrência de mutação (q). Após avaliação dos resultados para diferentes valores de *P_size* (10, 15, 20, 25, 30, 40, 100, 200 e 500) e *q* (0.1 e 0.2), concluiu-se que os melhores resultados foram obtidos com *P_size* igual a 100 e *q* igual a 0.2.

Por fim, o Método Exato baseado em Programação Linear Inteira (ILP) foi implementado para encontrar a solução ótima do problema. Foram definidas as restrições necessárias, como o número exato de nós selecionados, a conexão de cada nó a um único servidor e a distância máxima permitida entre nós. Os resultados obtidos após a execução do ILP com diferentes tempos limite (5 minutos e 1 hora) foram apresentados, mostrando a solução ótima encontrada e a distância média resultante.

Em suma, o documento destaca a análise comparativa dos métodos GRASP, GA e ILP, para o problema de minimizar a distância entre nós e os seus *servidores*. Cada método apresentou resultados distintos em termos de qualidade das soluções encontradas e tempo de execução. A escolha do método mais adequado depende dos objetivos do problema e das restrições de tempo e recursos disponíveis.