

# Mini Projeto 2

Universidade de Aveiro

Mariana Pinto, Raquel Pinto



# Mini Projeto 2

Modelação e Desempenho de Redes e Serviços  
Departamento de Eletrónica, Telecomunicações e Informática  
Universidade de Aveiro

Mariana Pinto, Raquel Pinto  
(84792) mariana17@ua.pt, (92948) raq.milh@ua.pt

Data de Entrega: 2 Fevereiro 2022

# Conteúdo

<b>Assignment</b>	<b>1</b>
<b>1 Task 1</b>	<b>2</b>
1.a Task 1.a - Enunciado . . . . .	2
1.a.1 Código . . . . .	2
1.a.2 Resultados e Conclusões . . . . .	4
1.b Task 1.b - Enunciado . . . . .	5
1.b.1 Código . . . . .	5
1.b.2 Resultados e Conclusões . . . . .	8
1.c Task 1.c - Enunciado . . . . .	10
1.c.1 Código . . . . .	10
1.c.2 Resultados e Conclusões . . . . .	14
1.d Task 1.d - Enunciado . . . . .	16
1.d.1 Código . . . . .	16
1.d.2 Resultados e Conclusões . . . . .	22
1.e Task 1.e - Enunciado . . . . .	23
1.e.1 Código . . . . .	23
1.e.2 Resultados e Conclusões . . . . .	32
<b>2 Task 2</b>	<b>35</b>
2.a Task 2.a - Enunciado . . . . .	35
2.a.1 Código . . . . .	35
2.a.2 Resultados e Conclusões . . . . .	39
2.b Task 2.b - Enunciado . . . . .	41
2.b.1 Código . . . . .	41
2.b.2 Resultados e Conclusões . . . . .	46
2.c Task 2.c - Enunciado . . . . .	49
2.c.1 Código . . . . .	49
2.c.2 Resultados e Conclusões . . . . .	56
2.d Task 2.d - Enunciado . . . . .	58
2.d.1 Código . . . . .	58
2.d.2 Resultados e Conclusões . . . . .	70
<b>3 Task 3</b>	<b>73</b>
3.a Task 3.a - Enunciado . . . . .	73
3.a.1 Código . . . . .	73
3.a.2 Resultados e Conclusões . . . . .	75
3.b Task 3.b - Enunciado . . . . .	76
3.b.1 Código . . . . .	76
3.b.2 Resultados e Conclusões . . . . .	78
3.c Task 3.c - Enunciado . . . . .	78
3.c.1 Código . . . . .	78
3.c.2 Resultados e Conclusões . . . . .	81
3.d Task 3.d - Enunciado . . . . .	81

3.d.1	Código . . . . .	81
3.d.2	Resultados e Conclusões . . . . .	84
3.e	Task 3.e - Enunciado . . . . .	84
3.e.1	Resultados e Conclusões . . . . .	84
<b>4</b>	<b>Task 4</b>	<b>86</b>
4.a	Task 4.a - Enunciado . . . . .	86
4.a.1	Código . . . . .	86
4.a.2	Resultados e Conclusões . . . . .	89
<b>5</b>	<b>Contribuições dos autores</b>	<b>90</b>

# Assignment

Consider the MPLS (Multi-Protocol Label Switching) network of an ISP (Internet Service Provider) with the following topology composed by 10 nodes and 16 links and defined over a rectangle with 600 Km by 400 Km:

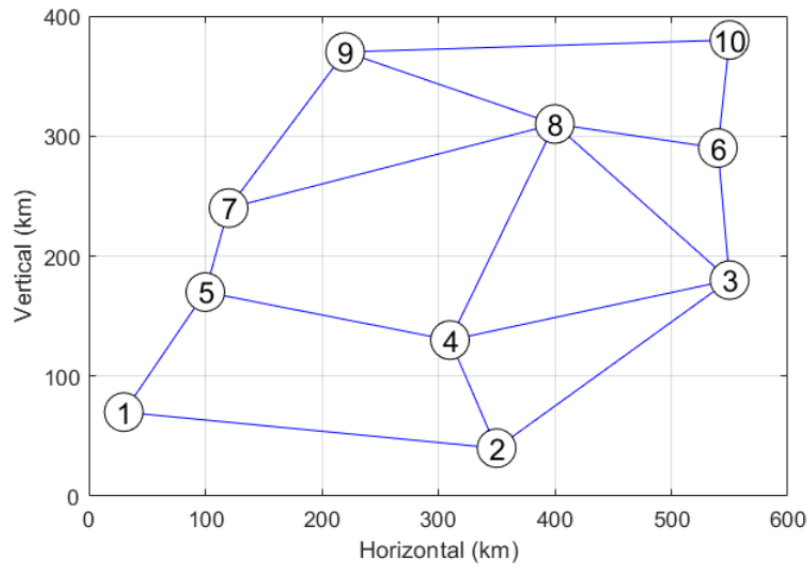


Figura 1: MPSL Network

The length of all links is provided by the square matrix  $L$ . The capacity of all links is 10 Gbps in each direction. Consider a unicast service defined with the following 9 flows (throughput values  $b_t$  and  $\underline{b}_t$  in Gbps):

$t$	$o_t$	$d_t$	$b_t$	$\underline{b}_t$
1	1	3	1.0	1.0
2	1	4	0.7	0.5
3	2	7	2.4	1.5
4	3	4	2.4	2.1
5	4	9	1.0	2.2
6	5	6	1.2	1.5
7	5	8	2.1	2.2
8	5	9	1.6	1.9
9	6	10	1.4	1.6

# Capítulo 1

## Task 1

In this task, the aim is to compute a symmetrical single path routing solution to support the unicast service which minimizes the resulting worst link load.

### 1.a Task 1.a - Enunciado

With a k-shortest path algorithm (using the lengths of the links), compute the number of different routing paths provided by the network to each traffic flow. What do you conclude?

#### 1.a.1 Código

Neste exercício, foi pedido para calcular o número de diferentes caminhos de encaminhamento fornecidos pela rede de cada fluxo de tráfego, usando um algoritmo de caminho mais curto.

Para isso calculou-se as distâncias entre os nós, a capacidade de cada link (10 Gbps para todos os links) e o tamanho de cada link. Através da função *calculatePaths* calculou-se o caminho mais curto para cada link, ficando com duas variáveis (sP que armazena os caminhos e nSp que armazena os custos dos caminhos sP).

Com estas variáveis calculou-se o tráfego de cada link usando a função *calculateLinkLoads*. O número de diferentes caminhos de encaminhamento fornecidos pela rede de cada fluxo de tráfego é dado pelo máximo do tráfego de cada link.

O código gerado para a resolução do exercício é o seguinte:

```
1 fprintf("Task 1 - Alinea A\n");
2 Nodes= [30 70
3         350 40
4         550 180
5         310 130
6         100 170
7         540 290
8         120 240
9         400 310
10        220 370
11        550 380];
12 Links= [1 2
13         1 5
14         2 3
15         2 4
16         3 4
17         3 6
18         3 8
```

```

19         4 5
20         4 8
21         5 7
22         6 8
23         6 10
24         7 8
25         7 9
26         8 9
27         9 10];
28 T= [1 3 1.0 1.0
29      1 4 0.7 0.5
30      2 7 2.4 1.5
31      3 4 2.4 2.1
32      4 9 1.0 2.2
33      5 6 1.2 1.5
34      5 8 2.1 2.5
35      5 9 1.6 1.9
36      6 10 1.4 1.6];
37 nNodes= 10;
38 nLinks= size(Links,1);
39 nFlows= size(T,1);
40 B= 625; %Average packet size in Bytes
41 co= Nodes(:,1)+j*Nodes(:,2);
42 L= inf(nNodes); %Square matrix with arc lengths (in Km)
43 for i=1:nNodes
44     L(i,i)= 0;
45 end
46 C= zeros(nNodes); %Square matrix with arc capacities (in Gbps)
47 for i=1:nLinks
48     C(Links(i,1),Links(i,2))= 10; %Gbps
49     C(Links(i,2),Links(i,1))= 10; %Gbps
50     d= abs(co(Links(i,1))-co(Links(i,2)));
51     L(Links(i,1),Links(i,2))= d+5; %Km
52     L(Links(i,2),Links(i,1))= d+5; %Km
53 end
54 L= round(L); %Km
55 % Compute up to 100 paths for each flow:
56 n= 100;
57 [sP nSP]= calculatePaths(L,T,n);
58 for i = 1:nFlows
59     aux = size(sP{i});
60     tmp = aux(:,2);
61     for j=1:numel(tmp)
62         fprintf('Fluxo %d -> Number of different routing paths: %d\n',i,
63                 tmp(j));
64     end
65 end

```

## 1.a.2 Resultados e Conclusões

Sabe-se que o número de diferentes caminhos de encaminhamento fornecidos pela rede de cada fluxo de tráfego é dado pelo máximo do tráfego de cada link.

Observando a Figura 1.1 sabe-se que o fluxo 1 (origem = 1, destino = 3) assim como o fluxo 2 (origem = 1 destino = 4) conseguem ter 32 caminhos de encaminhamento diferentes, ou seja, ter 32 de tráfego. O fluxo 3 (origem = 2, destino = 7) consegue ter 38 de tráfego. O fluxo 4 (origem = 3 , destino = 4) consegue ter 24 de tráfego. O fluxo 5 (origem = 4, destino = 9) consegue ter 36 de tráfego. O fluxo 6 (origem = 5, destino = 6) consegue ter 37 de tráfego. O fluxo 7 (origem = 5, destino = 8) consegue ter 25 de tráfego. O fluxo 8 (origem = 5, destino = 9) consegue ter 41 de tráfego e o fluxo 9 (origem = 6, destino = 10) consegue ter 28 de tráfego.

Também se pode observar que o fluxo 3 é o que tem maior tráfego, ou seja, tem mais caminhos de encaminhamento diferentes quando comparado com os outros fluxos. Isto significa que o fluxo 3 suporta mais tráfego que os outros fluxos.

```
Task 1 - Alinea A
Fluxo 1 -> Number of different routing paths: 32
Fluxo 2 -> Number of different routing paths: 32
Fluxo 3 -> Number of different routing paths: 38
Fluxo 4 -> Number of different routing paths: 24
Fluxo 5 -> Number of different routing paths: 36
Fluxo 6 -> Number of different routing paths: 37
Fluxo 7 -> Number of different routing paths: 25
Fluxo 8 -> Number of different routing paths: 41
Fluxo 9 -> Number of different routing paths: 28
```

Figura 1.1: Número de caminhos de encaminhamento fornecidos pela rede de cada fluxo de tráfego.



## 1.b Task 1.b - Enunciado

Run a random algorithm during 10 seconds in three cases: (i) using all possible routing paths, (ii) using the 1 shortest routing paths and (iii) using the 5 shortest routing paths. For each case, register the worst link load value of the best solution, the number of solutions generated by the algorithm and the average quality of all solutions. On a single figure, plot for the three cases the worst link load values of all solutions in an increasing order. Take conclusions on the influence of the number of routing paths in the efficiency of the random algorithm.

### 1.b.1 Código

Foi pedido para se executar um algoritmo random durante 10 segundos para 3 casos:

- (1) - Utilizando todas as rotas possíveis;
- (2) - Utilizando as 10 rotas mais curtas;
- (3) - Utilizando as 5 rotas mais curtas.

Para cada caso tem que se registar o pior valor de carga de ligação da melhor solução, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções.

Para o primeiro caso, tal como no exercício anterior, calculou-se as distâncias entre os nós, a capacidade de cada link (10 Gbps para todos os links) e o tamanho de cada link. Através da função *calculatePaths* calculou-se o caminho mais curto para cada link, ficando com duas variáveis (sP que armazena os caminhos e nSp que armazena os custos dos caminhos sP).

Com estas variáveis usou-se o algoritmo de otimização random (dura 10 segundos) onde após seleccionar um custo random para cada fluxo, calcula-se o tráfego do link com igual custo ao atribuído anteriormente através da função *calculateLinkLoads*. A seguir seleccionou-se o maior tráfego calculado para se adicionar à matriz *allValues*. Assim através de um if simples, conseguiu-se saber qual é o caminho com menor tráfego, ou seja, melhor para enviar pacotes.

Para o segundo caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 10 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 10 percursos. Para isso viu-se quais caminhos tinham os 10 menores custos no *nSp*. É importante referir que se quer os caminhos com 10 menores custos, mas se se tiver menos que 10 caminhos, então quer-se os caminhos todos. O resto do programa é igual ao primeiro caso.

Para o terceiro caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 5 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 5 percursos. Para isso viu-se quais caminhos tinham os 5 menores custos no *nSp*. É importante referir que se quer os caminhos com 5 menores custos, mas se se tiver menos que 5 caminhos, então quer-se os caminhos todos, tal como no caso anterior.

O código gerado para a resolução do exercício é o seguinte:

```
1 clear all;
2 close all;
3
4 fprintf('Task 1 - Alinea B\n');
5
6 Nodes= [30 70
7         350 40
8         550 180
9         310 130
10        100 170
11        540 290
```

```

12         120 240
13         400 310
14         220 370
15         550 380];
16
17 Links= [1 2
18         1 5
19         2 3
20         2 4
21         3 4
22         3 6
23         3 8
24         4 5
25         4 8
26         5 7
27         6 8
28         6 10
29         7 8
30         7 9
31         8 9
32         9 10];
33
34 T= [1 3 1.0 1.0
35     1 4 0.7 0.5
36     2 7 2.4 1.5
37     3 4 2.4 2.1
38     4 9 1.0 2.2
39     5 6 1.2 1.5
40     5 8 2.1 2.5
41     5 9 1.6 1.9
42     6 10 1.4 1.6];
43
44 nNodes= 10;
45
46 nLinks= size(Links,1);
47 nFlows= size(T,1);
48
49 B= 625; %Average packet size in Bytes
50
51 co= Nodes(:,1)+j*Nodes(:,2); %calculo para a distancia de cada no
52
53 L= inf(nNodes); %Square matrix with arc lengths (in Km)
54 for i=1:nNodes
55     L(i,i)= 0;
56 end
57 C= zeros(nNodes); %Square matrix with arc capacities (in Gbps)
58 for i=1:nLinks
59     C(Links(i,1),Links(i,2))= 10; %Gbps
60     C(Links(i,2),Links(i,1))= 10; %Gbps
61     d= abs(co(Links(i,1))-co(Links(i,2)));
62     L(Links(i,1),Links(i,2))= d+5; %Km
63     L(Links(i,2),Links(i,1))= d+5; %Km
64 end
65 L= round(L); %Km
66

```

```

67 % Compute up to 100 paths for each flow:
68 n= 100;
69 [sP nSP]= calculatePaths(L,T,n);
70
71 %Optimization algorithm resorting to the random strategy:
72 fprintf('\nSolution random with all possible routing paths\n');
73 t= tic;
74 bestLoad= inf;
75 sol= zeros(1,nFlows);
76 allValues= [];
77 while toc(t)<10
78     for i= 1:nFlows
79         sol(i)= randi(nSP(i)); %selecionar um custo random para cada fluxo
80     end
81     %trafego do link com o custo selecionado
82     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
83     %seleciona o que tem maior trafego
84     load= max(max(Loads(:,3:4)));
85     allValues= [allValues load];
86     %ver qual o caminho com o menor trafego
87     if load<bestLoad
88         bestSol= sol;
89         bestLoad= load;
90     end
91 end
92 figure(1)
93 grid on
94 plot(sort(allValues));
95 title('Random Algorithm – Task 1.B')
96
97 fprintf('Worst link load value of the best solution = %.2f Gbps\n',
98     bestLoad);
99 fprintf('Number of solutions generated = %d \n', length(allValues));
100 fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
101     allValues));
102
103 fprintf('\nSolution random with 10 shortest routing paths\n');
104 t= tic;
105 bestLoad= inf;
106 sol= zeros(1,nFlows);
107 allValues= [];
108 while toc(t)<10
109     for i= 1:nFlows
110         n = min(10,nSP(i));
111         sol(i)= randi(n);
112     end
113     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
114     load= max(max(Loads(:,3:4)));
115     allValues= [allValues load];
116     if load<bestLoad
117         bestSol= sol;
118         bestLoad= load;
119     end
120 end
121 hold on

```

```

120 grid on
121 plot(sort(allValues));
122
123 fprintf('Worst link load value of the best solution = %.2f Gbps\n',
    bestLoad);
124 fprintf('Number of solutions generated = %d \n', length(allValues));
125 fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
    allValues));
126
127
128 fprintf('\nSolution random with 5 shortest routing paths\n');
129 t= tic;
130 bestLoad= inf;
131 sol= zeros(1,nFlows);
132 allValues= [];
133 while toc(t)<10
134     for i= 1:nFlows
135         n = min(5,nSP(i));
136         sol(i)= randi(n);
137     end
138     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
139     load= max(max(Loads(:,3:4)));
140     allValues= [allValues load];
141     if load<bestLoad
142         bestSol= sol;
143         bestLoad= load;
144     end
145 end
146 hold on
147 grid on
148 plot(sort(allValues));
149 legend('Random with all possible','Random with 10 shortest','Random with 5
    shortest',Location="southeast");
150 fprintf('Worst link load value of the best solution = %.2f Gbps\n',
    bestLoad);
151 fprintf('Number of solutions generated = %d \n', length(allValues));
152 fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
    allValues));

```

### 1.b.2 Resultados e Conclusões

Perante este código pode-se esperar que ao utilizar número limitado das rotas mais curtas, se tenha melhores resultados com mais qualidade, pois mesmo se a rede for pequena, ao reduzir-se os caminhos curtos (*shorted path*) as soluções ficam melhores. O que acontece é retirar-se soluções, mas só se descarta as últimas soluções (soluções maiores), ficando assim com as melhores soluções que são também as primeiras (mais curtas). As boas soluções costumam estar sempre nos primeiros *shorted path* por isso ao reduzir está-se a selecionar as melhores soluções.

O nosso objetivo no caso 2 e 3 é gerar poucas soluções com resultados bons em vez de gerar muitas soluções e com resultados piores.

Tal como se pode ver na Figura 1.2, no primeiro caso (utilizando todas as rotas possíveis) obteve-se 4.90 Gbps como pior valor de carga de ligação da melhor solução enquanto que se obteve 4.40 Gbps e 4.00 Gbps para os casos 2 (utilizando as 10 rotas mais curtas) e 3 (utilizando as 5 rotas mais curtas), respetivamente. Pode-se ver também que existe melhor qualidade de soluções no caso 3 do que nos outros casos, sendo as piores soluções as do caso 1.

Foi gerado sempre maior número de soluções quanto menores forem os caminhos, com valores de

145688, 147649 e 147930 para o caso 1, 2 e 3, respetivamente. Isto acontece porque os caminhos são mais pequenos e assim o algoritmo demora menos tempo a calcular a solução, tendo o limite máximo de 10 segundos em todos os programas é de esperar que gere mais soluções para caminhos mais curtos do que para os maiores.

```
Task 1 - Alinea B

Solution random with all possible routing paths
Worst link load value of the best solution = 4.90 Gbps
Number of solutions generated = 145688
Average quality of all solutions generated = 10.32 Gbps

Solution random with 10 shortest routing paths
Worst link load value of the best solution = 4.40 Gbps
Number of solutions generated = 147649
Average quality of all solutions generated = 8.51 Gbps

Solution random with 5 shortest routing paths
Worst link load value of the best solution = 4.00 Gbps
Number of solutions generated = 147930
Average quality of all solutions generated = 7.74 Gbps
```

Figura 1.2: Resultados numéricos para a alínea B nos diferentes casos.

Como se pode ver no Gráfico 1.3, tem-se melhores resultados se se reduzir o tamanho das soluções geradas (caso 2 e 3) quando comparado com o caso 1, pois ao limitar o número de rotas mais curtas, reduz-se os caminhos existentes a caminhos mais curtos o que faz com que se fique com as melhores soluções. Pode-se comprovar isto observando o Gráfico 1.3 consegue-se ver que as soluções finais são próximas de caso para caso.

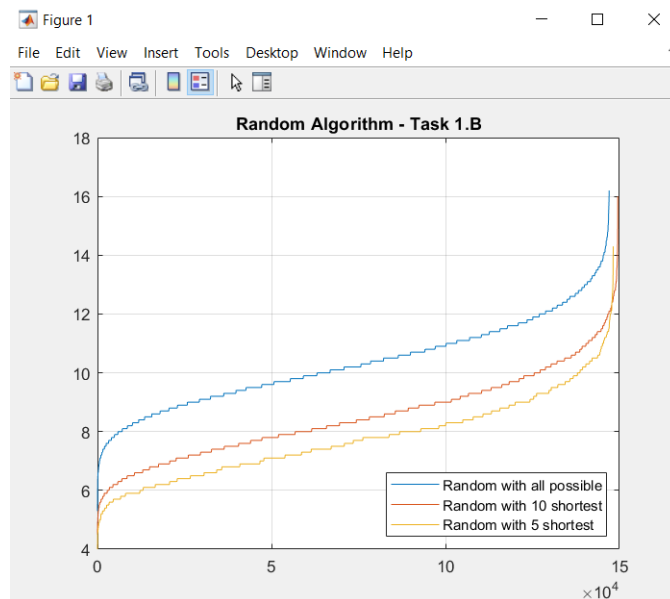


Figura 1.3: Resultados gráficos para a alínea B nos diferentes casos.

A diferença computacionalmente do caso 1 para os casos 2 e 3 é praticamente nula (visto que só se diminui-o o número de rotas a utilizar), mas existe uma melhoria significativa dos casos 2 e 3 quando comparados com o caso 1.

## 1.c Task 1.c - Enunciado

Repeat experiment 1.b but now using a greedy randomized algorithm instead of the random algorithm. Take conclusions on the influence of the number of routing paths in the efficiency of the greedy randomized algorithm.

### 1.c.1 Código

Neste exercício, foi pedido para se executar um algoritmo greedy randomized durante 10 segundos para 3 casos:

- (1) - Utilizando todas as rotas possíveis;
- (2) - Utilizando as 10 rotas mais curtas;
- (3) - Utilizando as 5 rotas mais curtas.

Para cada caso tem que se registar o pior valor de carga de ligação da melhor solução, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções.

Para o primeiro caso, tal como no exercício anterior, calculou-se as distâncias entre os nós, a capacidade de cada link (10 Gbps para todos os links) e o tamanho de cada link. Através da função *calculatePaths* calculou-se o caminho mais curto para cada link, ficando com duas variáveis (sP que armazena os caminhos e nSp que armazena os custos dos caminhos sP).

Com estas variáveis usou-se o algoritmo de otimização greedy randomized (com duração de 10 segundos) onde se criou um array (*ax2*) com números aleatórios de tamanho *nFlows*. Depois disto calculou-se o custo mínimo para valores aleatórios dos fluxos. Calculou-se o tráfego de cada link através da função *calculateLinkLoads*. Viu-se qual era o máximo do tráfego que existe para se conseguir obter o caminho com menor tráfego (melhor para enviar pacotes) e o custo desse caminho. A seguir calculou-se o tráfego para o melhor custo como na solução do algoritmo random, ou seja, calculou-se o tráfego do link com o custo atribuído anteriormente através da função *calculateLinkLoads*. A seguir seleciona-se o maior tráfego calculado para se adicionar à matriz *allValues*. Assim através de um if simples, consegue-se saber qual é o caminho com menor tráfego, ou seja, melhor para enviar pacotes.

Para o segundo caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 10 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 10 percursos. Para isso viu-se quais caminhos tinham os 10 menores custos no array *nSp*. É importante referir que tal como no exercício anterior, que se quer os caminhos com 10 menores custos, mas se se tiver menos que 10 caminhos, então seleciona-se os caminhos todos. O resto do programa é igual ao primeiro caso.

Para o terceiro caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 5 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 5 percursos. Para isso viu-se quais caminhos tinham os 5 menores custos no *nSp*. É importante referir que se quer os caminhos com 5 menores custos, mas se se tiver menos que 5 caminhos, então seleciona-se os caminhos todos, tal como no caso anterior. O resto do programa é igual ao primeiro caso.

O código gerado para a resolução do exercício é o seguinte:

```
1 clear all;
2 close all;
3
4 fprintf('Task 1 - Alinea C\n');
5 Nodes= [30 70
6         350 40
7         550 180
8         310 130
9         100 170
```

```

10         540 290
11         120 240
12         400 310
13         220 370
14         550 380];
15
16 Links= [1 2
17         1 5
18         2 3
19         2 4
20         3 4
21         3 6
22         3 8
23         4 5
24         4 8
25         5 7
26         6 8
27         6 10
28         7 8
29         7 9
30         8 9
31         9 10];
32
33 T= [1 3 1.0 1.0
34     1 4 0.7 0.5
35     2 7 2.4 1.5
36     3 4 2.4 2.1
37     4 9 1.0 2.2
38     5 6 1.2 1.5
39     5 8 2.1 2.5
40     5 9 1.6 1.9
41     6 10 1.4 1.6];
42
43 nNodes= 10;
44
45 nLinks= size(Links,1);
46 nFlows= size(T,1);
47
48 B= 625; %Average packet size in Bytes
49
50 co= Nodes(:,1)+j*Nodes(:,2);
51
52 L= inf(nNodes); %Square matrix with arc lengths (in Km)
53 for i=1:nNodes
54     L(i,i)= 0;
55 end
56 C= zeros(nNodes); %Square matrix with arc capacities (in Gbps)
57 for i=1:nLinks
58     C(Links(i,1),Links(i,2))= 10; %Gbps
59     C(Links(i,2),Links(i,1))= 10; %Gbps
60     d= abs(co(Links(i,1))-co(Links(i,2)));
61     L(Links(i,1),Links(i,2))= d+5; %Km
62     L(Links(i,2),Links(i,1))= d+5; %Km
63 end
64 L= round(L); %Km

```

```

65
66 % Compute up to 100 paths for each flow:
67 n= 100;
68 [sP nSP]= calculatePaths(L,T,n);
69
70 fprintf('\nSolution greedy randomized using all possible routing paths\n');
71 %Optimization algorithm resorting to the greedy randomized strategy:
72 t= tic;
73 bestLoad= inf;
74 sol= zeros(1,nFlows);
75 allValues= [];
76 while toc(t)<10
77     ax2 = randperm(nFlows);
78     sol= zeros(1,nFlows);
79     for i= ax2
80         k_best = 0;
81         best = inf;
82         %calculo do custo minimo para os valores aleatorios do fluxo
83         for k = 1:nSP(i)
84             sol(i)= k;
85             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
86             load= max(max(Loads(:,3:4)));
87             if load < best
88                 k_best = k;
89                 best = load;
90             end
91         end
92         sol(i) = k_best; %melhor custo para o fluxo i
93     end
94     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
95     load= max(max(Loads(:,3:4)));
96     allValues= [allValues load];
97     if load<bestLoad
98         bestSol= sol;
99         bestLoad= load;
100     end
101 end
102
103 hold on
104 grid on
105 plot(sort(allValues));
106 title('Greedy Randomized – Task 1.C')
107 fprintf('Worst link load value of the best solution = %.2f Gbps\n',
        bestLoad);
108 fprintf('Number of solutions generated = %d \n', length(allValues));
109 fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
        allValues));
110
111 fprintf('\nSolution greedy randomized using 10 shortest routing paths\n');
112 t= tic;
113 bestLoad= inf;
114 sol= zeros(1,nFlows);
115 allValues= [];
116 while toc(t)<10
117     ax2 = randperm(nFlows);

```



```

118     sol= zeros(1,nFlows);
119     for i= ax2
120         k_best = 0;
121         best = inf;
122         n = min(10,nSP(i)); % correr so 10 vezes , 10 primeiros
123         for k = 1:n
124             sol(i)= k;
125             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
126             load= max(max(Loads(:,3:4)));
127             if load < best
128                 k_best = k;
129                 best = load;
130             end
131         end
132         sol(i) = k_best;
133     end
134     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
135     load= max(max(Loads(:,3:4)));
136     allValues= [allValues load];
137     if load<bestLoad
138         bestSol= sol;
139         bestLoad= load;
140     end
141 end
142 hold on
143 grid on
144 plot(sort(allValues));
145 fprintf('Worst link load value of the best solution = %.2f Gbps\n',
        bestLoad);
146 fprintf('Number of solutions generated = %d \n', length(allValues));
147 fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
        allValues));
148
149
150 fprintf('\nSolution greedy randomized using 5 shortest routing paths\n');
151 t= tic;
152 bestLoad= inf;
153 sol= zeros(1,nFlows);
154 allValues= [];
155 while toc(t)<10
156     ax2 = randperm(nFlows);
157     sol= zeros(1,nFlows);
158     for i= ax2
159         k_best = 0;
160         best = inf;
161         n = min(5,nSP(i)); %correr so 5 vezes , 5 primeiros
162         for k = 1:n
163             sol(i)= k;
164             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
165             load= max(max(Loads(:,3:4)));
166             if load < best
167                 k_best = k;
168                 best = load;
169             end
170         end

```

```

171         sol(i) = k_best;
172     end
173     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
174     load= max(max(Loads(:,3:4)));
175     allValues= [allValues load];
176     if load<bestLoad
177         bestSol= sol;
178         bestLoad= load;
179     end
180 end
181 hold on
182 grid on
183 plot(sort(allValues));
184 legend('Greedy randomized using all possible','Greedy randomized using 10
        shortest','Greedy randomized using 5 shortest',Location="southeast");
185 fprintf('Worst link load value of the best solution = %.2f Gbps\n',
        bestLoad);
186 fprintf('Number of solutions generated = %d \n', length(allValues));
187 fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
        allValues));

```

## 1.c.2 Resultados e Conclusões

Neste exercício tal como no anterior, é de esperar que ao utilizar número limitado das rotas mais curtas, se tenha melhores resultados com mais qualidade, pois mesmo se a rede for pequena, ao reduzir-se os caminhos curtos (*shorted path*) as soluções ficam melhores. O que acontece é retirar-se soluções, mas só se descarta as últimas soluções (soluções maiores), ficando assim com as melhores soluções que são também as primeiras (mais curtas). As boas soluções costumam estar sempre nos primeiros *shorted path* por isso ao reduzir está-se a selecionar as melhores soluções.

O nosso objetivo no caso 2 e 3 é gerar poucas soluções, mas com resultados bons em vez de gerar muitas e com resultados piores.

Tal como se pode ver na Figura 1.4, no primeiro caso (utilizando todas as rotas possíveis) obteve-se 3.70 Gbps como pior valor de carga de ligação da melhor solução enquanto que se obteve 3.70 Gbps e 4.53 Gbps para os casos 2 (utilizando as 10 rotas mais curtas) e 3 (utilizando as 5 rotas mais curtas), respetivamente. Pode-se ver também que existe melhor qualidade de soluções no caso 3 do que nos outros casos, sendo as piores soluções as do caso 1.

Foi gerado sempre maior número de soluções quanto menores forem os caminhos, com valores de 5335, 17854 e 34302 para o caso 1, 2 e 3, respetivamente. Isto acontece porque os caminhos são mais pequenos e assim o algoritmo demora menos tempo a calcular a solução, tendo o limite máximo de 10 segundos em todos os programas é de esperar que gere mais soluções para caminhos mais curtos do que para os maiores.

```

Task 1 - Alinea C

Solution greedy randomized using all possible routing paths
Worst link load value of the best solution = 3.70 Gbps
Number of solutions generated = 5335
Average quality of all solutions generated = 4.54 Gbps

Solution greedy randomized using 10 shortest routing paths
Worst link load value of the best solution = 3.70 Gbps
Number of solutions generated = 17854
Average quality of all solutions generated = 4.53 Gbps

Solution greedy randomized using 5 shortest routing paths
Worst link load value of the best solution = 4.00 Gbps
Number of solutions generated = 34302
Average quality of all solutions generated = 5.06 Gbps

```

Figura 1.4: Resultados numéricos para a alínea C nos diferentes casos.

Como podemos ver no Gráfico 1.5, os melhores resultados se se reduzir o tamanho das soluções geradas (caso 2 e 3) quando comparado com o caso 1, pois ao limitar o número de rotas mais curtas, reduz-se os caminhos existentes a caminhos mais curtos o que faz com que no final se fique com as melhores soluções. Pode-se comprovar isto observando o Gráfico 1.3 onde se consegue ver que as soluções são próximas de caso para caso.

Também é importante referir que se encontra a solução mais rapidamente quando se tem o número de rotas limitado, isto acontece porque existe menos rotas e essas rotas são as melhores soluções.

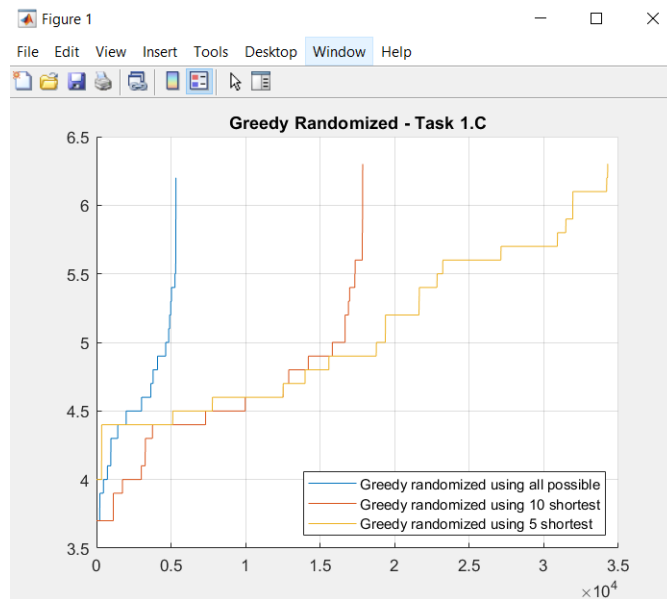


Figura 1.5: Resultados gráficos para a alínea C nos diferentes casos.

A diferença computacionalmente do caso 1 para os casos 2 e 3 é praticamente nula visto que só se diminui o número de rotas a utilizar, mas existe uma melhoria significativa dos casos 2 e 3 quando comparados com o caso 1.

## 1.d Task 1.d - Enunciado

Repeat experiment 1.b but now using a multi start hill climbing algorithm instead of the random algorithm. Take conclusions on the influence of the number of routing paths in the efficiency of the multi start hill climbing algorithm.

### 1.d.1 Código

Neste exercício, foi pedido para se executar um algoritmo hill climbing durante 10 segundos para 3 casos:

- (1) - Utilizando todas as rotas possíveis;
- (2) - Utilizando as 10 rotas mais curtas;
- (3) - Utilizando as 5 rotas mais curtas.

Para cada caso tem que se registar o pior valor de carga de ligação da melhor solução, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções.

O algoritmo hill climbing, é uma técnica de otimização que usa um algoritmo iterativo que começa com uma solução arbitrária para um problema e depois tenta encontrar uma solução melhor fazendo uma alteração incremental na solução. Se a alteração produzir uma solução melhor, outra alteração incremental será feita na nova solução e assim sucessivamente até que não sejam encontradas mais melhorias.

Para este exercício, foi usado o código do professor, pois só se conseguiu perceber que usa-se o algoritmo greedy anterior para construir uma solução, não conseguindo implementar a escolha da solução com o hill climbing. Este código foi alterado para se conseguir executar o caso 2 e 3, como se verifica a seguir.

Para o segundo caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 10 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 10 percursos. Para isso viu-se quais caminhos tinham os 10 menores custos no array *nSp*. É importante referir que se quer os caminhos com 10 menores custos, mas se se tiver menos que 10 caminhos, então seleciona-se os caminhos todos. O resto do programa é igual ao primeiro caso.

Para o terceiro caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 5 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 5 percursos. Para isso viu-se quais caminhos tinham os 5 menores custos no *nSp*. É importante referir que se quer os caminhos com 5 menores custos, mas se se tiver menos que 5 caminhos, então seleciona-se os caminhos todos, tal como no caso anterior. O resto do programa é igual ao primeiro caso.

O código gerado para a resolução do exercício é o seguinte:

```
1 clear all;
2 close all;
3
4 fprintf("Task 1 - Alinea D\n");
5 Nodes= [30 70
6         350 40
7         550 180
8         310 130
9         100 170
10        540 290
11        120 240
12        400 310
13        220 370
14        550 380];
15
16 Links= [1 2
```

```

17         1 5
18         2 3
19         2 4
20         3 4
21         3 6
22         3 8
23         4 5
24         4 8
25         5 7
26         6 8
27         6 10
28         7 8
29         7 9
30         8 9
31         9 10];
32
33 T= [1 3 1.0 1.0
34     1 4 0.7 0.5
35     2 7 2.4 1.5
36     3 4 2.4 2.1
37     4 9 1.0 2.2
38     5 6 1.2 1.5
39     5 8 2.1 2.5
40     5 9 1.6 1.9
41     6 10 1.4 1.6];
42
43 nNodes= 10;
44
45 nLinks= size(Links,1);
46 nFlows= size(T,1);
47
48 co= Nodes(:,1)+j*Nodes(:,2);
49 L= inf(nNodes); %Square matrix with arc lengths (in Km)
50 for i=1:nNodes
51     L(i,i)= 0;
52 end
53 for i=1:nLinks
54     d= abs(co(Links(i,1))-co(Links(i,2)));
55     L(Links(i,1),Links(i,2))= d+5; %Km
56     L(Links(i,2),Links(i,1))= d+5; %Km
57 end
58 L= round(L); %Km
59
60 % Compute up to n paths for each flow:
61 n= inf;
62 [sP nSP]= calculatePaths(L,T,n);
63
64 tempo= 10;
65
66 fprintf('\nSolution hill climbing using all possible routing paths\n');
67 %Optimization algorithm with multi start hill climbing:
68 t= tic;
69 bestLoad= inf;
70 allValues= [];
71 contadortotal= [];

```

```

72 while toc(t)<tempo
73
74 %GREEDY RANDOMIZED:
75 %construir uma solucao
76 ax2= randperm(nFlows);
77 sol= zeros(1,nFlows);
78 for i= ax2
79     k_best= 0;
80     best= inf;
81     for k= 1:nSP(i)
82         sol(i)= k;
83         Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
84         load= max(max(Loads(:,3:4)));
85         if load<best
86             k_best= k;
87             best= load;
88         end
89     end
90     sol(i)= k_best;
91 end
92 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
93 load= best;
94
95 %HILL CLIMBING:
96 %pegar na solucao do greedy e escolher a solucao com o hill climbing
97 continuar= true;
98 while continuar
99     i_best= 0;
100     k_best= 0;
101     best= load;
102     for i= 1:nFlows %cada fluxo
103         for k= 1:nSP(i) %cada percurso -> vamos a cada fluxo e a cada
percurso do fluxo
104             if k~=sol(i) %se o percurso for diferente do atualmente
escolhido
105                 aux= sol(i);
106                 sol(i)= k;
107                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol); %
calcula cargas
108                 load1= max(max(Loads(:,3:4))); %calcula carga maxima
109                 if load1<best
110                     i_best= i;
111                     k_best= k;
112                     best= load1;
113                 end
114                 sol(i)= aux; %repor o fluxo original
115             end
116         end
117     end
118     %quando nenhuma das solucoes for melhor, ele para -> i_best=0 (nao
119     %houve troca dentro dos fores)
120     if i_best>0
121         sol(i_best)= k_best;
122         load= best;
123     else

```

```

124         continuar= false;
125     end
126 end
127 allValues= [allValues load];
128 if load<bestLoad
129     bestSol= sol;
130     bestLoad= load;
131 end
132 end
133 figure(1);
134 grid on
135 plot(sort(allValues));
136 title('Multi Start Hill Climbing – Task 1.D');
137
138 fprintf('    Best load = %.2f Gbps\n',bestLoad);
139 fprintf('    No. of solutions = %d\n',length(allValues));
140 fprintf('    Av. quality of solutions = %.2f Gbps\n',mean(allValues));
141
142
143 fprintf('\nSolution hill climbing using 10 shortest routing paths\n');
144 t= tic;
145 bestLoad= inf;
146 allValues= [];
147 contadortotal= [];
148 while toc(t)<tempo
149
150     %GREEDY RANDOMIZED:
151     ax2= randperm(nFlows);
152     sol= zeros(1,nFlows);
153     for i= ax2
154         k_best= 0;
155         best= inf;
156         n = min(10,nSP(i));
157         for k= 1:n
158             sol(i)= k;
159             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
160             load= max(max(Loads(:,3:4)));
161             if load<best
162                 k_best= k;
163                 best= load;
164             end
165         end
166         sol(i)= k_best;
167     end
168     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
169     load= best;
170
171     %HILL CLIMBING:
172     continuar= true;
173     while continuar
174         i_best= 0;
175         k_best= 0;
176         best= load;
177         for i= 1:nFlows
178             for k= 1:nSP(i)

```

```

179         if k~=sol(i)
180             aux= sol(i);
181             sol(i)= k;
182             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
183             load1= max(max(Loads(:,3:4)));
184             if load1<best
185                 i_best= i;
186                 k_best= k;
187                 best= load1;
188             end
189             sol(i)= aux;
190         end
191     end
192 end
193 if i_best>0
194     sol(i_best)= k_best;
195     load= best;
196 else
197     continuar= false;
198 end
199 end
200 allValues= [allValues load];
201 if load<bestLoad
202     bestSol= sol;
203     bestLoad= load;
204 end
205 end
206 hold on
207 grid on
208 plot(sort(allValues));
209
210 fprintf(' Best load = %.2f Gbps\n',bestLoad);
211 fprintf(' No. of solutions = %d\n',length(allValues));
212 fprintf(' Av. quality of solutions = %.2f Gbps\n',mean(allValues));
213
214
215 fprintf('\nSolution hill climbing using 5 shortest routing paths\n');
216 t= tic;
217 bestLoad= inf;
218 allValues= [];
219 contadortotal= [];
220 while toc(t)<tempo
221
222     %GREEDY RANDOMIZED:
223     ax2= randperm(nFlows);
224     sol= zeros(1,nFlows);
225     for i= ax2
226         k_best= 0;
227         best= inf;
228         n = min(5,nSP(i));
229         for k= 1:n
230             sol(i)= k;
231             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
232             load= max(max(Loads(:,3:4)));
233             if load<best

```



```

234         k_best= k;
235         best= load;
236     end
237 end
238     sol(i)= k_best;
239 end
240 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
241 load= best;
242
243 %HILL CLIMBING:
244 continuar= true;
245 while continuar
246     i_best= 0;
247     k_best= 0;
248     best= load;
249     for i= 1:nFlows
250         for k= 1:nSP(i)
251             if k~=sol(i)
252                 aux= sol(i);
253                 sol(i)= k;
254                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
255                 load1= max(max(Loads(:,3:4)));
256                 if load1<best
257                     i_best= i;
258                     k_best= k;
259                     best= load1;
260                 end
261                 sol(i)= aux;
262             end
263         end
264     end
265
266     if i_best>0
267         sol(i_best)= k_best;
268         load= best;
269     else
270         continuar= false;
271     end
272 end
273 allValues= [allValues load];
274 if load<bestLoad
275     bestSol= sol;
276     bestLoad= load;
277 end
278 end
279 hold on
280 grid on
281 plot(sort(allValues));
282 legend('Hill climbing using all possible','Hill climbing using 10 shortest',
        'Hill climbing using 5 shortest',Location="southeast");
283 fprintf('    Best load = %.2f Gbps\n',bestLoad);
284 fprintf('    No. of solutions = %d\n',length(allValues));
285 fprintf('    Av. quality of solutions = %.2f Gbps\n',mean(allValues));

```

### 1.d.2 Resultados e Conclusões

Com este código, ao usar o algoritmo hill climbing é de esperar que ao utilizar número limitado das rotas mais curtas, se tenha melhores resultados, pois as boas soluções costumam estar sempre nos primeiros *shorted path* por isso ao reduzir está-se a selecionar as melhores soluções. É de notar que o hill climbing encontra rapidamente a melhor solução, logo, é de esperar que apesar de ter melhores resultados para o caso de se limitar o número de rotas mais curtas, este resultado não seja muito diferente de quando não se limita o número de rotas. Ou seja, com o hill climbing encontra-se muito rapidamente a melhor solução então os resultados obtidos de caso para caso (caso 1, 2 e 3) não devem ser muito diferentes.

Tal como se pode ver na Figura 1.6, no primeiro caso (utilizando todas as rotas possíveis) teve-se 3.70 Gbps como pior valor de carga de ligação da melhor solução para todos os casos. Pode-se ver também que a qualidade das soluções geradas não varia muito entre os casos. Isto acontece como já foi dito, porque o algoritmo hill climbing encontra muito rapidamente a melhor solução.

Quanto ao número de soluções geradas, pode-se ver que também não houve muita alteração de caso para caso, isto acontece pelo mesmo motivo já referido.

```
Task 1 - Alinea D

Solution hill climbing using all possible routing paths
Best load = 3.70 Gbps
No. of solutions = 1589
Av. quality of solutions = 4.29 Gbps

Solution hill climbing using 10 shortest routing paths
Best load = 3.70 Gbps
No. of solutions = 1963
Av. quality of solutions = 4.21 Gbps

Solution hill climbing using 5 shortest routing paths
Best load = 3.70 Gbps
No. of solutions = 1766
Av. quality of solutions = 4.26 Gbps
```

Figura 1.6: Resultados numéricos para a alínea D nos diferentes casos.

Como podemos ver no Gráfico 1.7, os resultados de caso para caso, são muito próximos uns dos outros, logo comprova-se que as melhores soluções estão nos primeiros *shorted path*. É de notar que com a diminuição do número de rotas mais curtas, o algoritmo encontra mais rapidamente a melhor solução, pois existe menos rotas para ele analisar, mas esta diferença não é tão significativa como nos algoritmos random e greedy randomized.

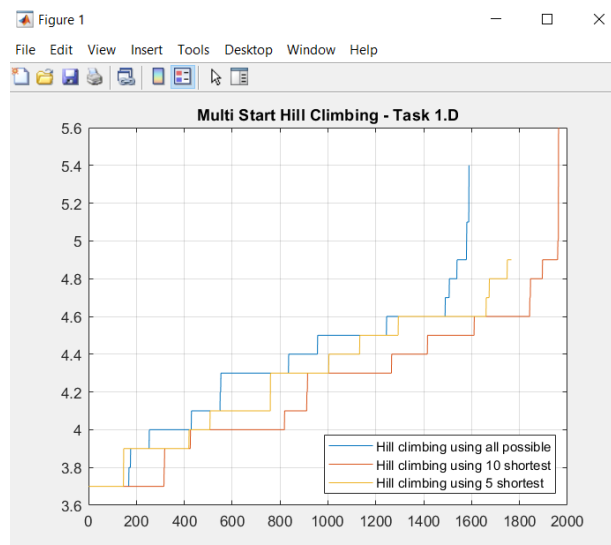


Figura 1.7: Resultados gráficos para a alínea D nos diferentes casos.

## 1.e Task 1.e - Enunciado

Compare the efficiency of the three heuristic algorithms based on the results obtained in 1.b, 1.c and 1.d.

### 1.e.1 Código

Como foi pedido para se comparar os resultados dos diferentes algoritmos, criou-se um programa que nos dá os diferentes gráficos.

O código gerado para a resolução do exercício é o seguinte:

```
1 clear all;
2 close all;
3
4 fprintf('Task 1 - Alinea E\n');
5
6 Nodes= [30 70
7         350 40
8         550 180
9         310 130
10        100 170
11        540 290
12        120 240
13        400 310
14        220 370
15        550 380];
16
17 Links= [1 2
18         1 5
19         2 3
20         2 4
21         3 4
22         3 6
23         3 8
24         4 5
25         4 8
26         5 7
27         6 8
28         6 10
29         7 8
30         7 9
31         8 9
32         9 10];
33
34 T= [1 3 1.0 1.0
35     1 4 0.7 0.5
36     2 7 2.4 1.5
37     3 4 2.4 2.1
38     4 9 1.0 2.2
39     5 6 1.2 1.5
40     5 8 2.1 2.5
41     5 9 1.6 1.9
42     6 10 1.4 1.6];
43
44 nNodes= 10;
```

```

45
46 nLinks= size(Links,1);
47 nFlows= size(T,1);
48
49 B= 625; %Average packet size in Bytes
50
51 co= Nodes(:,1)+j*Nodes(:,2); %calculo para a distancia de cada no
52
53 L= inf(nNodes); %Square matrix with arc lengths (in Km)
54 for i=1:nNodes
55     L(i,i)= 0;
56 end
57 C= zeros(nNodes); %Square matrix with arc capacities (in Gbps)
58 for i=1:nLinks
59     C(Links(i,1),Links(i,2))= 10; %Gbps
60     C(Links(i,2),Links(i,1))= 10; %Gbps
61     d= abs(co(Links(i,1))-co(Links(i,2)));
62     L(Links(i,1),Links(i,2))= d+5; %Km
63     L(Links(i,2),Links(i,1))= d+5; %Km
64 end
65 L= round(L); %Km
66
67 % Compute up to 100 paths for each flow:
68 n= 100;
69 [sP nSP]= calculatePaths(L,T,n);
70
71 %Optimization algorithm resorting to the random strategy:
72 fprintf('\nSolution random with all possible routing paths\n');
73 t= tic;
74 bestLoad= inf;
75 sol= zeros(1,nFlows);
76 allValues= [];
77 while toc(t)<10
78     for i= 1:nFlows
79         sol(i)= randi(nSP(i));
80     end
81     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
82     load= max(max(Loads(:,3:4)));
83     allValues= [allValues load];
84     if load<bestLoad
85         bestSol= sol;
86         bestLoad= load;
87     end
88 end
89 figure(1)
90 grid on
91 plot(sort(allValues));
92 title('Compare the algorithms – Task 1.E')
93
94 fprintf('Worst link load value of the best solution = %.2f Gbps\n',
95     bestLoad);
96 fprintf('Number of solutions generated = %d \n', length(allValues));
97 fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
98     allValues));
99

```

```

98
99 fprintf('\nSolution greedy randomized using all possible routing paths\n');
100 %Optimization algorithm resorting to the greedy randomized strategy:
101 t= tic;
102 bestLoad= inf;
103 sol= zeros(1,nFlows);
104 allValues= [];
105 while toc(t)<10
106     ax2 = randperm(nFlows);
107     sol= zeros(1,nFlows);
108     for i= ax2
109         k_best = 0;
110         best = inf;
111         for k = 1:nSP(i)
112             sol(i)= k;
113             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
114             load= max(max(Loads(:,3:4)));
115             if load < best
116                 k_best = k;
117                 best = load;
118             end
119         end
120         sol(i) = k_best;
121     end
122     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
123     load= max(max(Loads(:,3:4)));
124     allValues= [allValues load];
125     if load<bestLoad
126         bestSol= sol;
127         bestLoad= load;
128     end
129 end
130
131 hold on
132 grid on
133 plot(sort(allValues));
134 fprintf('Worst link load value of the best solution = %.2f Gbps\n',
        bestLoad);
135 fprintf('Number of solutions generated = %d \n', length(allValues));
136 fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
        allValues));
137
138 tempo= 10;
139 fprintf('\nSolution hill climbing using all possible routing paths\n');
140 %Optimization algorithm with multi start hill climbing:
141 t= tic;
142 bestLoad= inf;
143 allValues= [];
144 contadortotal= [];
145 while toc(t)<tempo
146
147     %GREEDY RANDOMIZED:
148     ax2= randperm(nFlows);
149     sol= zeros(1,nFlows);
150     for i= ax2

```

```

151     k_best= 0;
152     best= inf;
153     for k= 1:nSP(i)
154         sol(i)= k;
155         Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
156         load= max(max(Loads(:,3:4)));
157         if load<best
158             k_best= k;
159             best= load;
160         end
161     end
162     sol(i)= k_best;
163 end
164 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
165 load= best;
166
167 %HILL CLIMBING:
168 continuar= true;
169 while continuar
170     i_best= 0;
171     k_best= 0;
172     best= load;
173     for i= 1:nFlows
174         for k= 1:nSP(i)
175             if k~=sol(i)
176                 aux= sol(i);
177                 sol(i)= k;
178                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
179                 load1= max(max(Loads(:,3:4)));
180                 if load1<best
181                     i_best= i;
182                     k_best= k;
183                     best= load1;
184                 end
185                 sol(i)= aux;
186             end
187         end
188     end
189     if i_best>0
190         sol(i_best)= k_best;
191         load= best;
192     else
193         continuar= false;
194     end
195 end
196 allValues= [allValues load];
197 if load<bestLoad
198     bestSol= sol;
199     bestLoad= load;
200 end
201 end
202 hold on
203 grid on
204 plot(sort(allValues));
205 legend('Random using all possible','Greedy randomized using all possible',

```

```

        Hill climbing using all possible',Location="southeast");
206
207 fprintf('    Best load = %.2f Gbps\n',bestLoad);
208 fprintf('    No. of solutions = %d\n',length(allValues));
209 fprintf('    Av. quality of solutions = %.2f Gbps\n',mean(allValues));
210
211
212 fprintf('\nSolution random with 10 shortest routing paths\n');
213 t= tic;
214 bestLoad= inf;
215 sol= zeros(1,nFlows);
216 allValues= [];
217 while toc(t)<10
218
219     for i= 1:nFlows
220         n = min(10,nSP(i));
221         sol(i)= randi(n);
222     end
223     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
224     load= max(max(Loads(:,3:4)));
225     allValues= [allValues load];
226     if load<bestLoad
227         bestSol= sol;
228         bestLoad= load;
229     end
230 end
231 figure(2);
232 grid on
233
234 plot(sort(allValues));
235 title('Compare the algorithms – Task 1.E')
236 fprintf('Worst link load value of the best solution = %.2f Gbps\n',
        bestLoad);
237 fprintf('Number of solutions generated = %d \n', length(allValues));
238 fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
        allValues));
239
240 fprintf('\nSolution greedy randomized using 10 shortest routing paths\n');
241 t= tic;
242 bestLoad= inf;
243 sol= zeros(1,nFlows);
244 allValues= [];
245 while toc(t)<10
246     ax2 = randperm(nFlows);
247     sol= zeros(1,nFlows);
248     for i= ax2
249         k_best = 0;
250         best = inf;
251         n = min(10,nSP(i));
252         for k = 1:n
253             sol(i)= k;
254             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
255             load= max(max(Loads(:,3:4)));
256             if load < best
257                 k_best = k;

```

```

258         best = load;
259     end
260 end
261     sol(i) = k_best;
262 end
263     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
264     load= max(max(Loads(:,3:4)));
265     allValues= [allValues load];
266     if load<bestLoad
267         bestSol= sol;
268         bestLoad= load;
269     end
270 end
271 hold on
272 grid on
273 plot(sort(allValues));
274 fprintf('Worst link load value of the best solution = %.2f Gbps\n',
        bestLoad);
275 fprintf('Number of solutions generated = %d \n', length(allValues));
276 fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
        allValues));
277
278 fprintf('\nSolution hill climbing using 10 shortest routing paths\n');
279 t= tic;
280 bestLoad= inf;
281 allValues= [];
282 contadortotal= [];
283 while toc(t)<tempo
284
285     %GREEDY RANDOMIZED:
286     ax2= randperm(nFlows);
287     sol= zeros(1,nFlows);
288     for i= ax2
289         k_best= 0;
290         best= inf;
291         n = min(10,nSP(i));
292         for k= 1:n
293             sol(i)= k;
294             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
295             load= max(max(Loads(:,3:4)));
296             if load<best
297                 k_best= k;
298                 best= load;
299             end
300         end
301         sol(i)= k_best;
302     end
303     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
304     load= best;
305
306     %HILL CLIMBING:
307     continuar= true;
308     while continuar
309         i_best= 0;
310         k_best= 0;

```



```

311         best= load;
312         for i= 1:nFlows
313             for k= 1:nSP(i)
314                 if k~=sol(i)
315                     aux= sol(i);
316                     sol(i)= k;
317                     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
318                     load1= max(max(Loads(:,3:4)));
319                     if load1<best
320                         i_best= i;
321                         k_best= k;
322                         best= load1;
323                     end
324                     sol(i)= aux;
325                 end
326             end
327         end
328
329         if i_best>0
330             sol(i_best)= k_best;
331             load= best;
332         else
333             continuar= false;
334         end
335     end
336     allValues= [allValues load];
337     if load<bestLoad
338         bestSol= sol;
339         bestLoad= load;
340     end
341 end
342 hold on
343 grid on
344 plot(sort(allValues));
345 legend('Random with 10 shortest','Greedy randomized with 10 shortest','Hill
    climbing with 10 shortest',Location="southeast");
346
347 fprintf('    Best load = %.2f Gbps\n',bestLoad);
348 fprintf('    No. of solutions = %d\n',length(allValues));
349 fprintf('    Av. quality of solutions = %.2f Gbps\n',mean(allValues));
350
351
352 fprintf('\nSolution random with 5 shortest routing paths\n');
353 t= tic;
354 bestLoad= inf;
355 sol= zeros(1,nFlows);
356 allValues= [];
357 while toc(t)<10
358     for i= 1:nFlows
359         n = min(5,nSP(i));
360         sol(i)= randi(n);
361     end
362     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
363     load= max(max(Loads(:,3:4)));
364     allValues= [allValues load];

```

```

365         if load<bestLoad
366             bestSol= sol;
367             bestLoad= load;
368         end
369     end
370     figure(3)
371     grid on
372     plot(sort(allValues));
373     title('Compare the algorithms – Task 1.E')
374
375     fprintf('Worst link load value of the best solution = %.2f Gbps\n',
        bestLoad);
376     fprintf('Number of solutions generated = %d \n', length(allValues));
377     fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
        allValues));
378
379     fprintf('\nSolution greedy randomized using 5 shortest routing paths\n');
380     t= tic;
381     bestLoad= inf;
382     sol= zeros(1,nFlows);
383     allValues= [];
384     while toc(t)<10
385         ax2 = randperm(nFlows);
386         sol= zeros(1,nFlows);
387         for i= ax2
388             k_best = 0;
389             best = inf;
390             n = min(5,nSP(i));
391             for k = 1:n
392                 sol(i)= k;
393                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
394                 load= max(max(Loads(:,3:4)));
395                 if load < best
396                     k_best = k;
397                     best = load;
398                 end
399             end
400             sol(i) = k_best;
401         end
402         Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
403         load= max(max(Loads(:,3:4)));
404         allValues= [allValues load];
405         if load<bestLoad
406             bestSol= sol;
407             bestLoad= load;
408         end
409     end
410     hold on
411     grid on
412     plot(sort(allValues));
413     fprintf('Worst link load value of the best solution = %.2f Gbps\n',
        bestLoad);
414     fprintf('Number of solutions generated = %d \n', length(allValues));
415     fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
        allValues));

```

```

416
417
418 fprintf('\nSolution hill climbing using 5 shortest routing paths\n');
419 t= tic;
420 bestLoad= inf;
421 allValues= [];
422 contadortotal= [];
423 while toc(t)<tempo
424
425     %GREEDY RANDOMIZED:
426     ax2= randperm(nFlows);
427     sol= zeros(1,nFlows);
428     for i= ax2
429         k_best= 0;
430         best= inf;
431         n = min(5,nSP(i));
432         for k= 1:n
433             sol(i)= k;
434             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
435             load= max(max(Loads(:,3:4)));
436             if load<best
437                 k_best= k;
438                 best= load;
439             end
440         end
441         sol(i)= k_best;
442     end
443     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
444     load= best;
445
446     %HILL CLIMBING:
447     continuar= true;
448     while continuar
449         i_best= 0;
450         k_best= 0;
451         best= load;
452         for i= 1:nFlows
453             for k= 1:nSP(i)
454                 if k~=sol(i)
455                     aux= sol(i);
456                     sol(i)= k;
457                     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
458                     load1= max(max(Loads(:,3:4)));
459                     if load1<best
460                         i_best= i;
461                         k_best= k;
462                         best= load1;
463                     end
464                     sol(i)= aux;
465                 end
466             end
467         end
468         if i_best>0
469             sol(i_best)= k_best;
470             load= best;

```

```

471         else
472             continuar= false;
473         end
474     end
475     allValues= [allValues load];
476     if load<bestLoad
477         bestSol= sol;
478         bestLoad= load;
479     end
480 end
481 hold on
482 grid on
483 plot(sort(allValues));
484 legend('Random with 5 shortest','Greedy randomized with 5 shortest','Hill
         climbing with 5 shortest',Location="southeast");
485 fprintf('    Best load = %.2f Gbps\n',bestLoad);
486 fprintf('    No. of solutions = %d\n',length(allValues));
487 fprintf('    Av. quality of solutions = %.2f Gbps\n',mean(allValues));

```

## 1.e.2 Resultados e Conclusões

Tendo este código, é de esperar que o algoritmo hill climbing seja mais rápido e mais eficiente que o algoritmo greedy randomized e random, encontrando mais rapidamente a melhor solução e tendo melhores soluções. Isto acontece porque este algoritmo consegue encontrar rapidamente a melhor solução, tal como explicado no exercício anterior. O algoritmo greedy quando comparado com o algoritmo random é de esperar que seja bastante melhor, pois gera soluções mais cuidadas, como explicado nos exercícios anteriores enquanto que o algoritmo random geram soluções random, gerando assim muitas soluções até encontrar a melhor. Isto faz com que precise de muito mais tempo para encontrar a melhor solução que os outros dois algoritmos.

Task 1 - Alinea E	
Solution random with all possible routing paths Worst link load value of the best solution = 5.30 Gbps Number of solutions generated = 142334 Average quality of all solutions generated = 10.32 Gbps	Solution greedy randomized using 10 shortest routing paths Worst link load value of the best solution = 3.70 Gbps Number of solutions generated = 16981 Average quality of all solutions generated = 4.53 Gbps
Solution greedy randomized using all possible routing paths Worst link load value of the best solution = 3.70 Gbps Number of solutions generated = 5493 Average quality of all solutions generated = 4.54 Gbps	Solution hill climbing using 10 shortest routing paths Best load = 3.70 Gbps No. of solutions = 2113 Av. quality of solutions = 4.23 Gbps
Solution hill climbing using all possible routing paths Best load = 3.70 Gbps No. of solutions = 1623 Av. quality of solutions = 4.30 Gbps	Solution random with 5 shortest routing paths Worst link load value of the best solution = 4.00 Gbps Number of solutions generated = 146297 Average quality of all solutions generated = 7.75 Gbps
Solution random with 10 shortest routing paths Worst link load value of the best solution = 4.10 Gbps Number of solutions generated = 142419 Average quality of all solutions generated = 8.51 Gbps	Solution greedy randomized using 5 shortest routing paths Worst link load value of the best solution = 4.00 Gbps Number of solutions generated = 33260 Average quality of all solutions generated = 5.06 Gbps
Solution greedy randomized using 10 shortest routing paths Worst link load value of the best solution = 3.70 Gbps Number of solutions generated = 16981 Average quality of all solutions generated = 4.53 Gbps	Solution hill climbing using 5 shortest routing paths Best load = 3.70 Gbps No. of solutions = 1743 Av. quality of solutions = 4.27 Gbps

Figura 1.8: Resultados numéricos para a alínea E nos diferentes casos.

Observando a Figura 1.8 pode-se confirmar o que foi dito anteriormente, onde se obteve como pior valor de carga de ligação da melhor solução valores de 5.30 Gbps, 3.70 Gbps e 3.70 Gbps para os algoritmos random, greedy e hill climbing, respetivamente, quando se utiliza todas as rotas possíveis. Aqui pode-se

observar que a solução random obtém valores piores que os algoritmos de greedy randomized e de hill climbing, como o esperado. Também se pode observar que se teve 142334 soluções geradas pelo algoritmo random, 5493 para o algoritmo greedy randomized e 1623 para o algoritmo hill climbing. Com isto conclui-se duas coisas:

- Primeiro que o algoritmo hill climbing é mais complexo computacionalmente que o greedy randomized que por sua vez é mais complexo que o algoritmo random, pois deu-se a todos eles 10 segundos de execução. A solução random teve mais soluções que o algoritmo greedy que por sua vez teve mais soluções que o algoritmo hill climbing.

- Segundo, também se concluir que apesar do algoritmo ter gerado mais soluções do que os outros algoritmos, o seu pior valor de carga de ligação da melhor solução é pior que a solução dos restantes algoritmos. Pode-se observar que as melhores soluções geradas pelo algoritmo greedy randomized e pelo hill climbing têm o mesmo valor no pior valor de carga de ligação da melhor solução, mas as soluções geradas no algoritmo hill climbing são menos que no greedy randomized. O que significa que o algoritmo hill climbing gera menos soluções para chegar à mesma melhor solução que o greedy randomized, logo daqui pode se dizer que o algoritmo hill climbing é mais eficiente que o greedy randomized, tal como previsto.

Com isto pode-se esperar que o algoritmo hill climbing terá melhor qualidade das soluções que o greedy randomized que por sua vez terá melhores soluções que o algoritmo random. Isto é comprovado pelos resultados obtidos (10.32 Gbps para o algoritmo random, 4.54 Gbps para o algoritmo greedy randomized e 4.30 Gbps para o algoritmo hill climbing).

Também se pode esperar que o algoritmo hill climbing encontre a melhor solução mais rapidamente o algoritmo greedy randomized e que este a encontre mais rápido que o algoritmo random. Isto pode-se observar através do Gráfico 1.8.

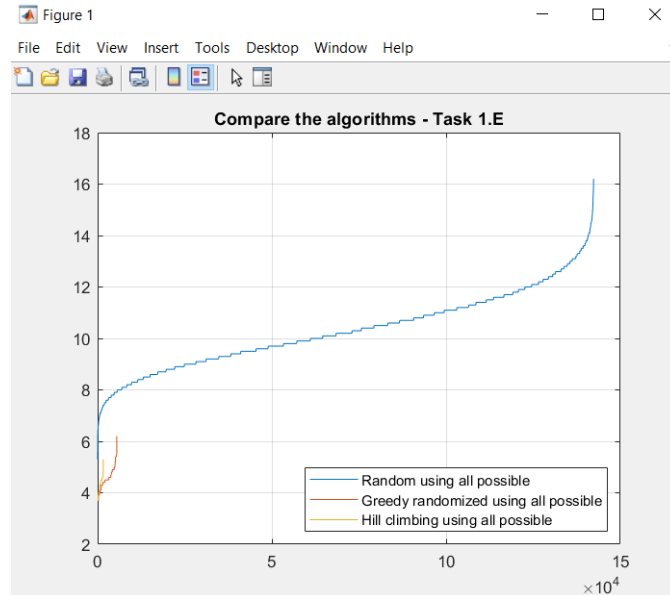


Figura 1.9: Resultados gráficos para a alínea E no caso de ter todas as rotas para os diferentes algoritmos.

Quando se utiliza 10 ou 5 rotas mais curtas, como dito nos exercícios anteriores é de esperar que quanto mais pequenas forem as rotas melhores serão os valores de pior valor de carga de ligação da melhor solução e mais rapidamente se encontrará a melhor solução como pode ser comprovado através da Figura 1.8 e dos Gráficos 1.9, 1.10 e 1.11.

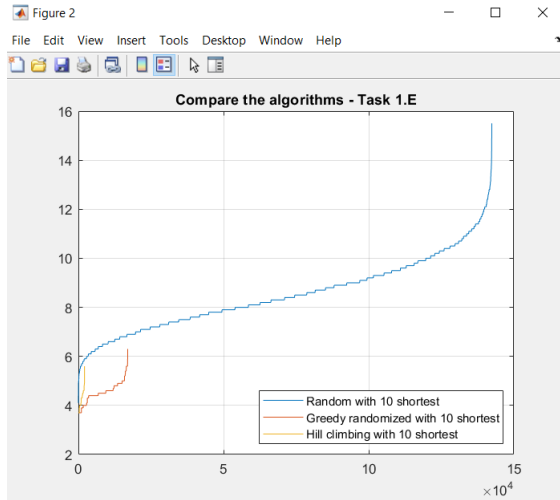


Figura 1.10: Resultados gráficos para a alínea E no caso de ter 10 rotas mais curtas para os diferentes algoritmos.

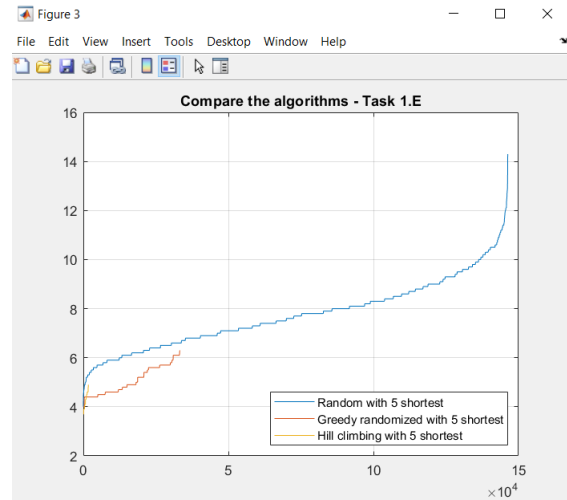


Figura 1.11: Resultados gráficos para a alínea E no caso de ter 5 rotas mais curtas para os diferentes algoritmos.

## Capítulo 2

### Task 2

Consider that the energy consumption of each link is proportional to its length. Consider also that a link not supporting traffic in any of its direction can be put in sleeping mode with no energy consumption. In this task, the aim is to compute a symmetrical single path routing solution to support the unicast service which minimizes the energy consumption of the network.

#### 2.a Task 2.a - Enunciado

Run a random algorithm during 10 seconds in three cases: (i) using all possible routing paths, (ii) using the 10 shortest routing paths, and (iii) using the 5 shortest routing paths. For each case, register the energy consumption value of the best solution, the number of solutions generated by the algorithm and the average quality of all solutions. On a single figure, plot for the three cases the worst link load values of all solutions in an increasing order. Take conclusions on the influence of the number of routing paths in the efficiency of the random algorithm.

##### 2.a.1 Código

Foi pedido para se executar um algoritmo random durante 10 segundos para 3 casos:

- (1) - Utilizando todas as rotas possíveis;
- (2) - Utilizando as 10 rotas mais curtas;
- (3) - Utilizando as 5 rotas mais curta.

Para cada um destes casos tem que se registar o valor do consumo de energia da melhor solução, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções.

Para o primeiro caso, calculou-se as distâncias entre os nós, a capacidade de cada link (10 Gbps para todos os links) e o tamanho de cada link. Através da função *calculatePaths* calculou-se o caminho mais curto para cada link, ficando com duas variáveis (sP que armazena os caminhos e nSp que armazena os custos dos caminhos sP).

Com estas variáveis usou-se o algoritmo de otimização random (dura 10 segundos) onde após selecionar um custo random para cada fluxo, calcula-se o tráfego do link com igual custo ao atribuído anteriormente através da função *calculateLinkLoads*. A seguir selecionou-se o maior tráfego calculado. Verificou-se se o tráfego não é maior que 10 Gbps (garantindo que não se escolhe uma solução se a capacidade do tráfego for maior que 10 Gbps (*energy = inf*)). Para isso percorreu-se os links para se verificar se existe tráfego a passar entre os links numa direção e na direção oposta (*Loads(a,3)* e *Loads(a,4)*). Se existir quer dizer que não esta no modo *sleeping mode*, logo pode-se calcular a energia total que é dada pela soma entre a energia e o comprimento do link entre dois nós (*Loads(a,1)* e *Loads(a,2)*). A seguir adiciona-se à matriz

allValues o valor da energia calculado. Feito isto, através de um if simples, sabe-se qual caminho que gasta menos energia.

Para o segundo caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 10 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 10 percursos. Para isso viu-se quais caminhos tinham os 10 menores custos no  $nSp$ . A partir daqui é igual ao primeiro caso.

Para o terceiro caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 5 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 5 percursos. Para isso viu-se quais caminhos tinham os 5 menores custos no  $nSp$ , tal como no caso 2. A partir daqui é igual ao primeiro caso.

O código gerado para a resolução do exercício é o seguinte:

```
1 clear all;
2 close all;
3
4 fprintf('Task 2 – Alinea A\n');
5 Nodes= [30 70
6         350 40
7         550 180
8         310 130
9         100 170
10        540 290
11        120 240
12        400 310
13        220 370
14        550 380];
15
16 Links= [1 2
17         1 5
18         2 3
19         2 4
20         3 4
21         3 6
22         3 8
23         4 5
24         4 8
25         5 7
26         6 8
27         6 10
28         7 8
29         7 9
30         8 9
31         9 10];
32
33 T= [1 3 1.0 1.0
34     1 4 0.7 0.5
35     2 7 2.4 1.5
36     3 4 2.4 2.1
37     4 9 1.0 2.2
38     5 6 1.2 1.5
39     5 8 2.1 2.5
40     5 9 1.6 1.9]
```



```

41     6 10 1.4 1.6];
42
43 nNodes= 10;
44
45 nLinks= size(Links,1);
46 nFlows= size(T,1);
47
48 B= 625; %Average packet size in Bytes
49
50 co= Nodes(:,1)+j*Nodes(:,2);
51
52 L= inf(nNodes); %Square matrix with arc lengths (in Km)
53 for i=1:nNodes
54     L(i,i)= 0;
55 end
56 C= zeros(nNodes); %Square matrix with arc capacities (in Gbps)
57 for i=1:nLinks
58     C(Links(i,1),Links(i,2))= 10; %Gbps
59     C(Links(i,2),Links(i,1))= 10; %Gbps
60     d= abs(co(Links(i,1))-co(Links(i,2)));
61     L(Links(i,1),Links(i,2))= d+5; %Km
62     L(Links(i,2),Links(i,1))= d+5; %Km
63 end
64 L= round(L); %Km
65
66 % Compute up to 100 paths for each flow:
67 n= 100;
68 [sP nSP]= calculatePaths(L,T,n);
69
70 fprintf('\nSolution random with all possible routing paths\n');
71 t= tic;
72 bestEnergy= inf;
73 sol= zeros(1,nFlows);
74 allValues= [];
75 %quanto mais comprimento do link mais energia, logo queremos o que tem
    menos comprimento, menos energia.
76 while toc(t)<10
77     for i= 1:nFlows
78         sol(i)= randi(nSP(i));
79     end
80     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
81     load= max(max(Loads(:,3:4)));
82     if load<=10
83         energy = 0;
84         for a=1:nLinks
85             if Loads(a,3)+Loads(a,4)>0 %se nao estiver em sleep mode
86                 energy = energy + L(Loads(a,1),Loads(a,2)); %a energia e
                    igual a energia + o comprimento do link
87             end
88         end
89     else
90         energy=inf; %garante que nao escolhe se a capacidade for maior que
            10 gigabits
91     end
92     %escolhemos o caminho que gasta menos energia

```

```

93     allValues= [allValues energy];
94     if energy<bestEnergy
95         bestSol= sol;
96         bestEnergy= energy;
97     end
98 end
99 figure(1)
100 grid on
101 plot(sort(allValues));
102 title('Random Algorithm – Task 2.A')
103 fprintf('Energy consumption value of the best solution = %.1f \n',
        bestEnergy);
104 fprintf('Number of solutions generated = %d \n', length(allValues));
105 fprintf('Average quality of all solutions generated = %.1f \n', mean(
        allValues));
106
107
108 fprintf('\nSolution random with 10 shortest routing paths\n');
109 t= tic;
110 bestEnergy= inf;
111 sol= zeros(1,nFlows);
112 allValues= [];
113 while toc(t)<10
114     for i= 1:nFlows
115         n = min(10,nSP(i)); %correr so 10 vezes, 10 primeiros
116         sol(i)= randi(n);
117     end
118     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
119     load= max(max(Loads(:,3:4)));
120     if load<=10
121         energy = 0;
122         for a=1:nLinks
123             if Loads(a,3)+Loads(a,4)>0
124                 energy = energy + L(Loads(a,1),Loads(a,2));
125             end
126         end
127     else
128         energy=inf;
129     end
130     allValues= [allValues energy];
131     if energy<bestEnergy
132         bestSol= sol;
133         bestEnergy= energy;
134     end
135 end
136
137 hold on
138 grid on
139 plot(sort(allValues));
140 fprintf('Energy consumption value of the best solution = %.1f \n',
        bestEnergy);
141 fprintf('Number of solutions generated = %d \n', length(allValues));
142 fprintf('Average quality of all solutions generated = %.1f \n', mean(
        allValues));
143

```

```

144
145 fprintf('\nSolution random with 5 shortest routing paths\n');
146 t= tic;
147 bestEnergy= inf;
148 sol= zeros(1,nFlows);
149 allValues= [];
150 while toc(t)<10
151     for i= 1:nFlows
152         n = min(5,nSP(i)); %correr so 5 vezes, 5 primeiros
153         sol(i)= randi(n);
154     end
155     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
156     load= max(max(Loads(:,3:4)));
157     if load<=10
158         energy = 0;
159         for a=1:nLinks
160             if Loads(a,3)+Loads(a,4)>0
161                 energy = energy + L(Loads(a,1),Loads(a,2));
162             end
163         end
164     else
165         energy=inf;
166     end
167     allValues= [allValues energy];
168     if energy<bestEnergy
169         bestSol= sol;
170         bestEnergy= energy;
171     end
172 end
173
174
175 hold on
176 grid on
177 plot(sort(allValues));
178 legend('Random with all possible','Random with 10 shortest','Random with 5
    shortest',Location="southeast");
179 fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergy);
180 fprintf('Number of solutions generated = %d \n', length(allValues));
181 fprintf('Average quality of all solutions generated = %.1f \n', mean(
    allValues));

```

## 2.a.2 Resultados e Conclusões

Perante este código pode-se esperar que ao utilizar número limitado das rotas mais curtas, se tenha melhores resultados, pois mesmo se a rede for pequena, ao reduzir-se os caminhos curtos (*shorted path*) as soluções ficam melhores. O que acontece é retirar-se soluções, mas só se descarta as últimas soluções (soluções maiores), ficando assim com as melhores soluções que são também as primeiras (mais curtas). As boas soluções costumam estar sempre nos primeiros *shorted path* por isso ao reduzir está-se a selecionar as melhores soluções.

O nosso objetivo no caso 2 e 3 é gerar poucas soluções com resultados bons em vez de gerar muitas soluções e com resultados piores.

Tal como se pode ver na Figura 2.1, no primeiro caso (utilizando todas as rotas possíveis) obteve-se 2137.0 como valor do consumo de energia da melhor solução enquanto que se obteve 1696.0 Gbps e 1422.0 Gbps para os casos 2 (utilizando as 10 rotas mais curtas) e 3 (utilizando as 5 rotas mais curtas),

respetivamente. Pode-se ver também que a média da qualidade das soluções geradas são sempre inf, isto significa que muitas das soluções geradas acontecem quando o tráfego é maior que 10 Gbps.

Foi gerado sempre maior número de soluções quanto menores forem os caminhos, com valores de 144765, 145360 e 145723 para o caso 1, 2 e 3, respetivamente. Isto acontece porque os caminhos são mais pequenos e assim o algoritmo demora menos tempo a calcular a solução, tendo o limite máximo de 10 segundos em todos os programas é de esperar que gere mais soluções para caminhos mais curtos do que para os maiores.

```
Task 2 - Alínea A

Solution random with all possible routing paths
Energy consumption value of the best solution = 2137.0
Number of solutions generated = 144765
Average quality of all solutions generated = Inf

Solution random with 10 shortest routing paths
Energy consumption value of the best solution = 1696.0
Number of solutions generated = 145360
Average quality of all solutions generated = Inf

Solution random with 5 shortest routing paths
Energy consumption value of the best solution = 1422.0
Number of solutions generated = 145723
Average quality of all solutions generated = Inf
```

Figura 2.1: Resultados numéricos para a alínea A nos diferentes casos.

Como podemos ver no Gráfico 2.2, tem-se melhores resultados se se reduzir o tamanho das soluções geradas (caso 2 e 3) quando comparado com o caso 1, pois ao limitar o número de rotas mais curtas, reduz-se os caminhos existentes a caminhos mais curtos o que faz com que se fique com as melhores soluções. Pode-se comprovar isto observando o Gráfico 2.2 consegue-se ver que as soluções iniciais são próximas de caso para caso.

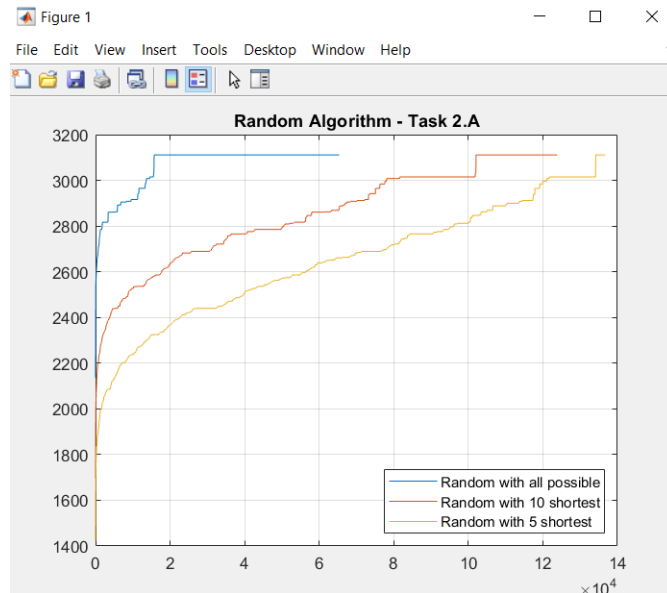


Figura 2.2: Resultados gráficos para a alínea A nos diferentes casos.

A diferença computacionalmente do caso 1 para os casos 2 e 3 é praticamente nula (visto que só se diminui-o o número de rotas a utilizar), mas existe uma melhoria significativa dos casos 2 e 3 quando comparados com o caso 1.

## 2.b Task 2.b - Enunciado

Repeat experiment 2.a but now using a greedy randomized algorithm instead of the random algorithm. Take conclusions on the influence of the number of routing paths in the efficiency of the greedy randomized algorithm.

### 2.b.1 Código

Neste exercício, foi pedido para se executar um algoritmo greedy randomized durante 10 segundos para 3 casos:

- (1) - Utilizando todas as rotas possíveis;
- (2) - Utilizando as 10 rotas mais curtas;
- (3) - Utilizando as 5 rotas mais curtas.

Para cada caso tem que se registar o valor do consumo de energia da melhor solução, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções.

Para o primeiro caso, tal como no exercício anterior, calculou-se as distâncias entre os nós, a capacidade de cada link (10 Gbps para todos os links) e o tamanho de cada link. Através da função *calculatePaths* calculou-se o caminho mais curto para cada link, ficando com duas variáveis (sP que armazena os caminhos e nSp que armazena os custos dos caminhos sP).

Com estas variáveis usou-se o algoritmo de otimização greedy randomized (com duração de 10 segundos) onde se criou um array (*ax2*) com números aleatórios de tamanho *nFlows*.

Depois disto tem que se calcular o melhor vizinho. Para isso, calculou-se o tráfego de cada link através da função *calculateLinkLoads*, viu-se qual era o máximo do tráfego que existe e garantiu-se que não se escolhe uma solução se o tráfego for maior que 10 Gbps. Percorreu-se os links para se verificar se existe tráfego a passar entre os links numa direção e na direção oposta (*Loads(a,3)* e *Loads(a,4)*). Se existir quer dizer que não está no modo *sleeping mode*, logo pode-se calcular a energia total que é dada pela soma entre a energia e o comprimento do link entre dois nós (*Loads(a,1)* e *Loads(a,2)*). Feito isto através de um if simples, sabe-se qual é o melhor vizinho, ou seja o que gasta menor energia.

Posto isto, tem que se ver para cada custo qual é a menor energia para o melhor vizinho calculado anteriormente. Isto faz-se calculando o tráfego de cada link através da função *calculateLinkLoads*, vendo-se qual é o máximo do tráfego e garantindo-se também que não se escolhe uma solução se o tráfego for maior que 10 Gbps. Percorreu-se os links para se verificar se existe tráfego a passar entre os links numa direção e na direção oposta (*Loads(a,3)* e *Loads(a,4)*). Se existir quer dizer que não está no modo *sleeping mode*, logo pode-se calcular a energia total que é dada pela soma entre a energia e o comprimento do link entre dois nós (*Loads(a,1)* e *Loads(a,2)*). A seguir adiciona-se à matriz *allValues* o valor da energia do melhor vizinho calculado. Feito isto, através de um if simples, sabe-se qual caminho que gasta menos energia.

Para o segundo caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 10 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 10 percursos. Para isso viu-se quais caminhos tinham os 10 menores custos no array *nSp*. Ou seja, calcula-se qual o melhor vizinho destes 10 caminhos. É importante referir que se quer os caminhos com 10 menores custos, mas se se tiver menos que 10 caminhos, então seleciona-se os caminhos todos. O resto do programa é igual ao primeiro caso.

Para o terceiro caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 5 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 5 percursos. Para isso viu-se quais caminhos tinham os 5 menores custos no *nSp*. Ou seja, calcula-se qual o melhor vizinho destes 5 caminhos. É importante referir que se quer os caminhos com 5 menores custos, mas se se tiver menos que 5 caminhos, então seleciona-se os caminhos todos, tal como no caso anterior. O resto do programa é igual ao primeiro caso.

O código gerado para a resolução do exercício é o seguinte:

```
1 clear all;
2 close all;
3
4 fprintf('Task 2 – Alinea B\n');
5 Nodes= [30 70
6         350 40
7         550 180
8         310 130
9         100 170
10        540 290
11        120 240
12        400 310
13        220 370
14        550 380];
15
16 Links= [1 2
17         1 5
18         2 3
19         2 4
20         3 4
21         3 6
22         3 8
23         4 5
24         4 8
25         5 7
26         6 8
27         6 10
28         7 8
29         7 9
30         8 9
31         9 10];
32
33 T= [1 3 1.0 1.0
34     1 4 0.7 0.5
35     2 7 2.4 1.5
36     3 4 2.4 2.1
37     4 9 1.0 2.2
38     5 6 1.2 1.5
39     5 8 2.1 2.5
40     5 9 1.6 1.9
41     6 10 1.4 1.6];
42
43 nNodes= 10;
44
45 nLinks= size(Links,1);
46 nFlows= size(T,1);
47
48 B= 625; %Average packet size in Bytes
49
50 co= Nodes(:,1)+j*Nodes(:,2);
51
52 L= inf(nNodes); %Square matrix with arc lengths (in Km)
53 for i=1:nNodes
54     L(i,i)= 0;
```

```

55 end
56 C= zeros(nNodes); %Square matrix with arc capacities (in Gbps)
57 for i=1:nLinks
58     C(Links(i,1),Links(i,2))= 10; %Gbps
59     C(Links(i,2),Links(i,1))= 10; %Gbps
60     d= abs(co(Links(i,1))-co(Links(i,2)));
61     L(Links(i,1),Links(i,2))= d+5; %Km
62     L(Links(i,2),Links(i,1))= d+5; %Km
63 end
64 L= round(L); %Km
65
66 % Compute up to 100 paths for each flow:
67 n= 100;
68 [sP nSP]= calculatePaths(L,T,n);
69
70 fprintf('\nSolution greedy randomized using all possible routing paths\n');
71
72 t= tic;
73 bestEnergyGreedy= inf;
74 sol= zeros(1,nFlows);
75 allValues= [];
76 while toc(t)<10
77     ax2 = randperm(nFlows); %array numa ordem aleatoria
78     sol= zeros(1,nFlows);
79     for i= ax2
80         k_best = 0;
81         best = inf;
82         %encontrar o melhor vizinho
83         for k = 1:nSP(i)
84             sol(i)= k;
85             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
86             load= max(max(Loads(:,3:4)));
87
88             if load<=10
89                 energy = 0;
90                 for a=1:nLinks
91                     if Loads(a,3)+Loads(a,4)>0
92                         energy = energy + L(Loads(a,1),Loads(a,2));
93                     end
94                 end
95             else
96                 energy=inf;
97             end
98
99             if energy < best
100                 k_best = k;
101                 best = energy;
102             end
103         end
104         sol(i) = k_best; % percurso com a melhor escolha
105     end
106     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
107     load= max(max(Loads(:,3:4)));
108     if load<=10
109         energy = 0;

```

```

110         for a=1:nLinks
111             if Loads(a,3)+Loads(a,4)>0
112                 energy = energy + L(Loads(a,1),Loads(a,2));
113             end
114         end
115     else
116         energy=inf;
117     end
118
119     allValues= [allValues energy];
120     if energy <bestEnergyGreedy
121         bestSol= sol;
122         bestEnergyGreedy= energy;
123     end
124 end
125 hold on
126 grid on
127 plot(sort(allValues));
128 title('Greedy Randomized – Task 2.B')
129 fprintf('Energy consumption value of the best solution = %.1f \n',
        bestEnergyGreedy);
130 fprintf('Number of solutions generated = %d \n', length(allValues));
131 fprintf('Average quality of all solutions generated = %.1f \n', mean(
        allValues));
132
133
134 fprintf('\nSolution greedy randomized using 10 shortest routing paths\n');
135 t= tic;
136 bestEnergyGreedy= inf;
137 sol= zeros(1,nFlows);
138 allValues= [];
139 while toc(t)<10
140     ax2 = randperm(nFlows); % array numa ordem aleatoria
141     sol= zeros(1,nFlows);
142     for i= ax2
143         k_best = 0;
144         best = inf;
145         %encontrar o melhor vizinho
146         n = min(10,nSP(i)); %correr so 10 vezes, 10 primeiros
147         for k = 1:n
148             sol(i)= k;
149             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
150             load= max(max(Loads(:,3:4)));
151
152             if load<=10
153                 energy = 0;
154                 for a=1:nLinks
155                     if Loads(a,3)+Loads(a,4)>0
156                         energy = energy + L(Loads(a,1),Loads(a,2));
157                     end
158                 end
159             else
160                 energy=inf;
161             end
162

```



```

163         if energy < best
164             k_best = k;
165             best = energy;
166         end
167     end
168     sol(i) = k_best; % percurso com a melhor escolha
169 end
170 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
171 load= max(max(Loads(:,3:4)));
172 if load<=10
173     energy = 0;
174     for a=1:nLinks
175         if Loads(a,3)+Loads(a,4)>0
176             energy = energy + L(Loads(a,1),Loads(a,2));
177         end
178     end
179 else
180     energy=inf;
181 end
182
183 allValues= [allValues energy];
184 if energy <bestEnergyGreedy
185     bestSol= sol;
186     bestEnergyGreedy= energy;
187 end
188 end
189
190 hold on
191 grid on
192 plot(sort(allValues));
193 fprintf('Energy consumption value of the best solution = %.1f \n',
194         bestEnergyGreedy);
195 fprintf('Number of solutions generated = %d \n', length(allValues));
196 fprintf('Average quality of all solutions generated = %.1f \n', mean(
197         allValues));
198
199 fprintf('\nSolution greedy randomized using 5 shortest routing paths\n');
200 t= tic;
201 bestEnergyGreedy= inf;
202 sol= zeros(1,nFlows);
203 allValues= [];
204 while toc(t)<10
205     ax2 = randperm(nFlows); %array numa ordem aleatoria
206     sol= zeros(1,nFlows);
207     for i= ax2
208         k_best = 0;
209         best = inf;
210         %encontrar o melhor vizinho
211         n = min(5,nSP(i)); %correr so 5 vezes , 5 primeiros
212         for k = 1:n
213             sol(i)= k;
214             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
215             load= max(max(Loads(:,3:4)));

```

```

216         if load <= 10
217             energy = 0;
218             for a=1:nLinks
219                 if Loads(a,3)+Loads(a,4)>0
220                     energy = energy + L(Loads(a,1),Loads(a,2));
221                 end
222             end
223         else
224             energy=inf;
225         end
226
227         if energy < best
228             k_best = k;
229             best = energy;
230         end
231     end
232     sol(i) = k_best; % percurso com a melhor escolha
233 end
234 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
235 load= max(max(Loads(:,3:4)));
236 if load <= 10
237     energy = 0;
238     for a=1:nLinks
239         if Loads(a,3)+Loads(a,4)>0
240             energy = energy + L(Loads(a,1),Loads(a,2));
241         end
242     end
243 else
244     energy=inf; s
245 end
246
247 allValues= [allValues energy];
248 if energy < bestEnergyGreedy
249     bestSol= sol;
250     bestEnergyGreedy= energy;
251 end
252 end
253
254 hold on
255 grid on
256 plot(sort(allValues));
257 legend('Greedy randomized using all possible','Greedy randomized using 10
258         shortest','Greedy randomized using 5 shortest',Location="southeast");
259 fprintf('Energy consumption value of the best solution = %.1f \n',
260         bestEnergyGreedy);
261 fprintf('Number of solutions generated = %d \n', length(allValues));
262 fprintf('Average quality of all solutions generated = %.1f \n', mean(
263         allValues));

```

## 2.b.2 Resultados e Conclusões

Neste exercício tal como no anterior, é de esperar que ao utilizar número limitado das rotas mais curtas, se tenha melhores resultados com mais qualidade, pois mesmo se a rede for pequena, ao reduzir-se os caminhos curtos (*shorted path*) as soluções ficam melhores. O que acontece é retirar-se soluções, mas só se descarta as últimas soluções (soluções maiores), ficando assim com as melhores soluções que são

também as primeiras (mais curtas). As boas soluções costumam estar sempre nos primeiros *shorted path* por isso ao reduzir está-se a selecionar as melhores soluções.

O nosso objetivo no caso 2 e 3 é gerar poucas soluções, mas com resultados bons em vez de gerar muitas e com resultados piores.

Tal como se pode ver na Figura 2.3, no primeiro caso (utilizando todas as rotas possíveis) obteve-se 1235.0 como valor do consumo da energia da melhor solução enquanto que obtive 1235.0 e 1303.0 para os casos 2 (utilizando as 10 rotas mais curtas) e 3 (utilizando as 5 rotas mais curtas), respetivamente. Pode-se ver também que existe melhor qualidade de soluções no caso 3 do que nos outros casos, sendo as piores soluções as do caso 1.

Foi gerado sempre maior número de soluções quanto menores forem os caminhos, com valores de 5188, 16522 e 32106 para o caso 1, 2 e 3, respetivamente. Isto acontece porque os caminhos são mais pequenos e assim o algoritmo demora menos tempo a calcular a solução, tendo o limite máximo de 10 segundos em todos os programas é de esperar que gere mais soluções para caminhos mais curtos do que para os maiores.

```
Task 2 - Alinea B

Solution greedy randomized using all possible routing paths
Energy consumption value of the best solution = 1235.0
Number of solutions generated = 5188
Average quality of all solutions generated = 1397.9

Solution greedy randomized using 10 shortest routing paths
Energy consumption value of the best solution = 1235.0
Number of solutions generated = 16522
Average quality of all solutions generated = 1415.4

Solution greedy randomized using 5 shortest routing paths
Energy consumption value of the best solution = 1303.0
Number of solutions generated = 32106
Average quality of all solutions generated = 1514.7
```

Figura 2.3: Resultados numéricos para a alínea B nos diferentes casos.

Como podemos ver no Gráfico 2.4, os melhores resultados se se reduzir o tamanho das soluções geradas (caso 2 e 3) quando comparado com o caso 1, pois ao limitar o número de rotas mais curtas, reduz-se os caminhos existentes a caminhos mais curtos o que faz com que no final se fique com as melhores soluções. Pode-se comprovar isto observando o Gráfico 2.4 onde se consegue ver que as soluções iniciais são próximas de caso para caso.

Também é importante referir que se encontra a solução mais rapidamente quando temos o número de rotas limitado, isto acontece porque existe menos rotas e essas rotas são as melhores soluções.

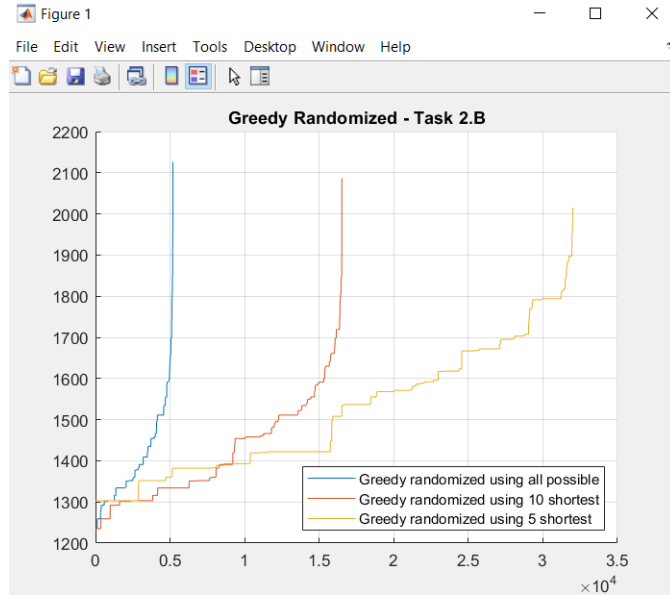


Figura 2.4: Resultados gráficos para a alínea B nos diferentes casos.

A diferença computacionalmente do caso 1 para os casos 2 e 3 é praticamente nula visto que só se diminui o número de rotas a utilizar, mas existe uma melhoria significativa dos casos 2 e 3 quando comparados com o caso 1. Do caso 1 para o caso 2, o valor do consumo da energia da melhor solução não aumentou, mas a média da qualidade das soluções aumentou assim como o número de soluções geradas.

## 2.c Task 2.c - Enunciado

Repeat experiment 2.a but now using a multi start hill climbing algorithm instead of the random algorithm. Take conclusions on the influence of the number of routing paths in the efficiency of the multi start hill climbing algorithm.

### 2.c.1 Código

Neste exercício, foi pedido para se executar um algoritmo hill climbing durante 10 segundos para 3 casos:

- (1) - Utilizando todas as rotas possíveis;
- (2) - Utilizando as 10 rotas mais curtas;
- (3) - Utilizando as 5 rotas mais curtas.

Para cada caso tem que se registar o valor do consumo de energia da melhor solução, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções.

Como dito anteriormente, o algoritmo hill climbing, é uma técnica de otimização que usa um algoritmo iterativo que começa com uma solução arbitrária para um problema e depois tenta encontrar uma solução melhor fazendo uma alteração incremental na solução. Se a alteração produzir uma solução melhor, outra alteração incremental será feita na nova solução e assim sucessivamente até que não sejam encontradas mais melhorias.

Para este exercício, foi usado o código do professor, pois é igual ao código do algoritmo hill climbing da task 1 adaptado para calcular a energia. Este código foi alterado para se conseguir executar o caso 2 e 3, como se verifica a seguir.

Para o segundo caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 10 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 10 percursos. Para isso viu-se quais caminhos tinham os 10 menores custos no array *nSp*. É importante referir que se quer os caminhos com 10 menores custos, mas se se tiver menos que 10 caminhos, então seleciona-se os caminhos todos. O resto do programa é igual ao primeiro caso.

Para o terceiro caso foi pedido para se usar o mesmo algoritmo do primeiro caso, mas agora utilizando apenas as 5 rotas mais curtas. Isto faz-se limitando-se a seleção do custo random para cada fluxo aos primeiros 5 percursos. Para isso viu-se quais caminhos tinham os 5 menores custos no *nSp*. É importante referir que se quer os caminhos com 5 menores custos, mas se se tiver menos que 5 caminhos, então seleciona-se os caminhos todos, tal como no caso anterior. O resto do programa é igual ao primeiro caso.

O código gerado para a resolução do exercício é o seguinte:

```
1 clear all;
2 close all;
3
4 fprintf('Task 2 - Alinea C\n');
5 Nodes= [30 70
6         350 40
7         550 180
8         310 130
9         100 170
10        540 290
11        120 240
12        400 310
13        220 370
14        550 380];
15
```

```

16 Links= [1 2
17          1 5
18          2 3
19          2 4
20          3 4
21          3 6
22          3 8
23          4 5
24          4 8
25          5 7
26          6 8
27          6 10
28          7 8
29          7 9
30          8 9
31          9 10];
32
33 T= [1 3 1.0 1.0
34      1 4 0.7 0.5
35      2 7 2.4 1.5
36      3 4 2.4 2.1
37      4 9 1.0 2.2
38      5 6 1.2 1.5
39      5 8 2.1 2.5
40      5 9 1.6 1.9
41      6 10 1.4 1.6];
42
43 nNodes= 10;
44 nLinks= size(Links,1);
45 nFlows= size(T,1);
46
47 co= Nodes(:,1)+j*Nodes(:,2);
48 L= inf(nNodes); %Square matrix with arc lengths (in Km)
49 for i=1:nNodes
50     L(i,i)= 0;
51 end
52 for i=1:nLinks
53     d= abs(co(Links(i,1))-co(Links(i,2)));
54     L(Links(i,1),Links(i,2))= d+5; %Km
55     L(Links(i,2),Links(i,1))= d+5; %Km
56 end
57 L= round(L); %Km
58
59 % Compute up to n paths for each flow:
60 n= inf;
61 [sP nSP]= calculatePaths(L,T,n);
62
63 tempo= 10;
64
65 fprintf('\nSolution hill climbing using all possible routing paths\n');
66 %Optimization algorithm with multi start hill climbing:
67 t= tic;
68 bestEnergy= inf;
69 allValues= [];
70 contadortotal= [];

```

```

71 while toc(t)<tempo
72     %GREEDY RANDOMIZED:
73     continuar= true;
74     while continuar
75         continuar= false;
76         ax2= randperm(nFlows);
77         sol= zeros(1,nFlows);
78         for i= ax2
79             k_best= 0;
80             best= inf;
81             for k= 1:nSP(i)
82                 sol(i)= k;
83                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
84                 load= max(max(Loads(:,3:4)));
85                 if load <= 10
86                     energy= 0;
87                     for a= 1:nLinks
88                         if Loads(a,3)+Loads(a,4)>0
89                             energy= energy + L(Loads(a,1),Loads(a,2));
90                         end
91                     end
92                 else
93                     energy= inf;
94                 end
95                 if energy<best
96                     k_best= k;
97                     best= energy;
98                 end
99             end
100             if k_best>0
101                 sol(i)= k_best;
102             else
103                 continuar= true;
104                 break;
105             end
106         end
107     end
108     energy= best;
109
110     %HILL CLIMBING:
111     continuar= true;
112     while continuar
113         i_best= 0;
114         k_best= 0;
115         best= energy;
116         for i= 1:nFlows
117             for k= 1:nSP(i)
118                 if k~=sol(i)
119                     aux= sol(i);
120                     sol(i)= k;
121                     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
122                     load1= max(max(Loads(:,3:4)));
123                     if load1 <= 10
124                         energy1= 0;
125                         for a= 1:nLinks

```

```

126         if Loads(a,3)+Loads(a,4)>0
127             energy1= energy1 + L(Loads(a,1),Loads(a,2))
128             ;
129         end
130     else
131         energy1= inf;
132     end
133     if energy1<best
134         i_best= i;
135         k_best= k;
136         best= energy1;
137     end
138     sol(i)= aux;
139 end
140 end
141 end
142 if i_best>0
143     sol(i_best)= k_best;
144     energy= best;
145 else
146     continuar= false;
147 end
148 end
149 allValues= [allValues energy];
150 if energy<bestEnergy
151     bestSol= sol;
152     bestEnergy= energy;
153 end
154 end
155 figure(1);
156 grid on
157 plot(sort(allValues));
158 title('Multi Start Hill Climbing – Task 2.C');
159 fprintf('    Best energy = %.1f\n',bestEnergy);
160 fprintf('    No. of solutions = %d\n',length(allValues));
161 fprintf('    Av. quality of solutions = %.1f\n',mean(allValues));
162
163 % o hill climbing nao e muito vantajoso para calcular a energia ,
164     normalmente ele nao
165
166 % consegue melhorar a solucao inicial so greeedy randomized
167
168 fprintf('\nSolution hill climbing using 10 shortest routing paths\n');
169 %Optimization algorithm with multi start hill climbing:
170 t= tic;
171 bestEnergy= inf;
172 allValues= [];
173 contadortotal= [];
174 while toc(t)<tempo
175     %GREEDY RANDOMIZED:
176     continuar= true;
177     while continuar
178         continuar= false;
179         ax2= randperm(nFlows);
180         sol= zeros(1,nFlows);

```



```

179     for i= ax2
180         k_best= 0;
181         best= inf;
182         n = min(10,nSP(i));
183         for k= 1:n
184             sol(i)= k;
185             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
186             load= max(max(Loads(:,3:4)));
187             if load <= 10
188                 energy= 0;
189                 for a= 1:nLinks
190                     if Loads(a,3)+Loads(a,4)>0
191                         energy= energy + L(Loads(a,1),Loads(a,2));
192                     end
193                 end
194             else
195                 energy= inf;
196             end
197             if energy<best
198                 k_best= k;
199                 best= energy;
200             end
201         end
202         if k_best>0
203             sol(i)= k_best;
204         else
205             continuar= true;
206             break;
207         end
208     end
209 end
210 energy= best;
211
212 %HILL CLIMBING:
213 continuar= true;
214 while continuar
215     i_best= 0;
216     k_best= 0;
217     best= energy;
218     for i= 1:nFlows
219         for k= 1:nSP(i)
220             if k~=sol(i)
221                 aux= sol(i);
222                 sol(i)= k;
223                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
224                 load1= max(max(Loads(:,3:4)));
225                 if load1 <= 10
226                     energy1= 0;
227                     for a= 1:nLinks
228                         if Loads(a,3)+Loads(a,4)>0
229                             energy1= energy1 + L(Loads(a,1),Loads(a,2))
230                             ;
231                         end
232                     end
233                 else

```

```

233         energy1= inf;
234     end
235     if energy1<best
236         i_best= i;
237         k_best= k;
238         best= energy1;
239     end
240     sol(i)= aux;
241 end
242 end
243 end
244 if i_best>0
245     sol(i_best)= k_best;
246     energy= best;
247 else
248     continuar= false;
249 end
250 end
251 allValues= [allValues energy];
252 if energy<bestEnergy
253     bestSol= sol;
254     bestEnergy= energy;
255 end
256 end
257 hold on
258 grid on
259 plot(sort(allValues));
260 fprintf('    Best energy = %.1f\n',bestEnergy);
261 fprintf('    No. of solutions = %d\n',length(allValues));
262 fprintf('    Av. quality of solutions = %.1f\n',mean(allValues));
263
264
265 fprintf('\nSolution hill climbing using 5 shortest routing paths\n');
266 %Optimization algorithm with multi start hill climbing:
267 t= tic;
268 bestEnergy= inf;
269 allValues= [];
270 contadortotal= [];
271 while toc(t)<tempo
272     %GREEDY RANDOMIZED:
273     continuar= true;
274     while continuar
275         continuar= false;
276         ax2= randperm(nFlows);
277         sol= zeros(1,nFlows);
278         for i= ax2
279             k_best= 0;
280             best= inf;
281             n = min(5,nSP(i));
282             for k= 1:n
283                 sol(i)= k;
284                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
285                 load= max(max(Loads(:,3:4)));
286                 if load <= 10
287                     energy= 0;

```

```

288         for a= 1:nLinks
289             if Loads(a,3)+Loads(a,4)>0
290                 energy= energy + L(Loads(a,1),Loads(a,2));
291             end
292         end
293     else
294         energy= inf;
295     end
296     if energy<best
297         k_best= k;
298         best= energy;
299     end
300 end
301 if k_best>0
302     sol(i)= k_best;
303 else
304     continuar= true;
305     break;
306 end
307 end
308 end
309 energy= best;
310
311 %HILL CLIMBING:
312 continuar= true;
313 while continuar
314     i_best= 0;
315     k_best= 0;
316     best= energy;
317     for i= 1:nFlows
318         for k= 1:nSP(i)
319             if k~=sol(i)
320                 aux= sol(i);
321                 sol(i)= k;
322                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
323                 load1= max(max(Loads(:,3:4)));
324                 if load1 <= 10
325                     energy1= 0;
326                     for a= 1:nLinks
327                         if Loads(a,3)+Loads(a,4)>0
328                             energy1= energy1 + L(Loads(a,1),Loads(a,2))
329                             ;
330                         end
331                     end
332                     energy1= inf;
333                 end
334                 if energy1<best
335                     i_best= i;
336                     k_best= k;
337                     best= energy1;
338                 end
339                 sol(i)= aux;
340             end
341         end

```

```

342         end
343         if i_best>0
344             sol(i_best)= k_best;
345             energy= best;
346         else
347             continuar= false;
348         end
349     end
350     allValues= [allValues energy];
351     if energy<bestEnergy
352         bestSol= sol;
353         bestEnergy= energy;
354     end
355 end
356 hold on
357 grid on
358 plot(sort(allValues));
359 legend('Hill climbing using all possible','Hill climbing using 10 shortest',
        , 'Hill climbing using 5 shortest',Location="southeast");
360 fprintf('    Best energy = %.1f\n',bestEnergy);
361 fprintf('    No. of solutions = %d\n',length(allValues));
362 fprintf('    Av. quality of solutions = %.1f\n',mean(allValues));

```

## 2.c.2 Resultados e Conclusões

Com este código, ao usar o algoritmo hill climbing é de esperar que ao utilizar número limitado das rotas mais curtas, se tenha melhores resultados, pois as boas soluções costumam estar sempre nos primeiros *shorted path* por isso ao reduzir está-se a selecionar as melhores soluções. É de notar que o hill climbing encontra rapidamente a melhor solução, logo, é de esperar que apesar de ter melhores resultados para o caso de se limitar o número de rotas mais curtas, este resultado não seja muito diferente de quando não se limita o número de rotas. Ou seja, com o hill climbing encontra-se muito rapidamente a melhor solução então os resultados obtidos de caso para caso (caso 1, 2 e 3) não devem ser muito diferentes.

Tal como se pode ver na Figura 2.5, no primeiro caso (utilizando todas as rotas possíveis) teve-se 1153.0 como valor do consumo da energia da melhor solução para todos os casos. Pode-se ver também que a qualidade das soluções geradas não varia muito entre os casos. Isto acontece como já foi dito, porque o algoritmo hill climbing encontra muito rapidamente a melhor solução.

Quanto ao número de soluções geradas, pode-se ver que também não houve muita alteração do caso para caso, isto acontece pelo mesmo motivo já referido.

```

Task 2 - Alinea C

Solution hill climbing using all possible routing paths
Best energy = 1153.0
No. of solutions = 7919
Av. quality of solutions = 1317.1

Solution hill climbing using 10 shortest routing paths
Best energy = 1153.0
No. of solutions = 7812
Av. quality of solutions = 1313.6

Solution hill climbing using 5 shortest routing paths
Best energy = 1153.0
No. of solutions = 7195
Av. quality of solutions = 1308.8

```

Figura 2.5: Resultados numéricos para a alínea C nos diferentes casos.

Como podemos ver no Gráfico 2.6, os resultados de caso para caso, são muito próximos uns dos outros, logo comprova-se que as melhores soluções estão nos primeiros *shorted path*. É de notar que com a diminuição do número de rotas mais curtas, o algoritmo encontra mais rapidamente a melhor solução (menos soluções geradas), pois existe menos rotas para ele analisar, mas esta diferença não é tão significativa como nos algoritmos random e greedy randomized.

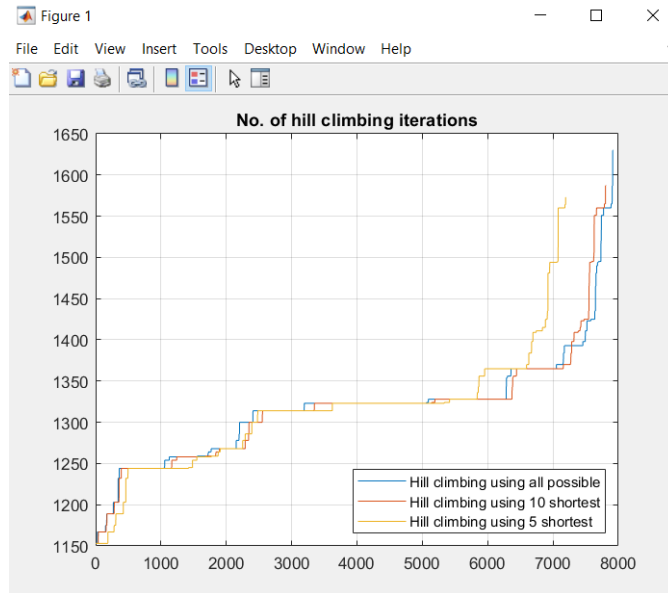


Figura 2.6: Resultados gráficos para a alínea C nos diferentes casos.

## 2.d Task 2.d - Enunciado

Compare the efficiency of the three heuristic algorithms based on the results obtained in 2.a, 2.b and 2.c.

### 2.d.1 Código

Como foi pedido para se comparar os resultados dos diferentes algoritmos, criou-se um programa que nos dá os diferentes gráficos.

O código gerado para a resolução do exercício é o seguinte:

```
1 clear all;
2 close all;
3
4 fprintf('Task 2 - Alinea D\n');
5 Nodes= [30 70
6         350 40
7         550 180
8         310 130
9         100 170
10        540 290
11        120 240
12        400 310
13        220 370
14        550 380];
15
16 Links= [1 2
17         1 5
18         2 3
19         2 4
20         3 4
21         3 6
22         3 8
23         4 5
24         4 8
25         5 7
26         6 8
27         6 10
28         7 8
29         7 9
30         8 9
31         9 10];
32
33 T= [1 3 1.0 1.0
34     1 4 0.7 0.5
35     2 7 2.4 1.5
36     3 4 2.4 2.1
37     4 9 1.0 2.2
38     5 6 1.2 1.5
39     5 8 2.1 2.5
40     5 9 1.6 1.9
41     6 10 1.4 1.6];
42
43 nNodes= 10;
44
```

```

45 nLinks= size(Links,1);
46 nFlows= size(T,1);
47 tempo= 10;
48
49 B= 625; %Average packet size in Bytes
50
51 co= Nodes(:,1)+j*Nodes(:,2);
52
53 L= inf(nNodes); %Square matrix with arc lengths (in Km)
54 for i=1:nNodes
55     L(i,i)= 0;
56 end
57 C= zeros(nNodes); %Square matrix with arc capacities (in Gbps)
58 for i=1:nLinks
59     C(Links(i,1),Links(i,2))= 10; %Gbps
60     C(Links(i,2),Links(i,1))= 10; %Gbps
61     d= abs(co(Links(i,1))-co(Links(i,2)));
62     L(Links(i,1),Links(i,2))= d+5; %Km
63     L(Links(i,2),Links(i,1))= d+5; %Km
64 end
65 L= round(L); %Km
66
67 % Compute up to 100 paths for each flow:
68 n= 100;
69 [sP nSP]= calculatePaths(L,T,n);
70
71 fprintf('\nSolution random with all possible routing paths\n');
72 t= tic;
73 bestEnergy= inf;
74 sol= zeros(1,nFlows);
75 allValues= [];
76 while toc(t)<10
77     for i= 1:nFlows
78         sol(i)= randi(nSP(i));
79     end
80     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
81     load= max(max(Loads(:,3:4)));
82     if load<=10
83         energy = 0;
84         for a=1:nLinks
85             if Loads(a,3)+Loads(a,4)>0
86                 energy = energy + L(Loads(a,1),Loads(a,2));
87             end
88         end
89     else
90         energy=inf;
91     end
92
93     allValues= [allValues energy];
94     if energy<bestEnergy
95         bestSol= sol;
96         bestEnergy= energy;
97     end
98 end
99 figure(1);

```

```

100 grid on
101 plot(sort(allValues));
102 title('Compare the algorithms – Task 2.D')
103 fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergy);
104 fprintf('Number of solutions generated = %d \n', length(allValues));
105 fprintf('Average quality of all solutions generated = %.1f \n', mean(
    allValues));
106
107
108 fprintf('\nSolution greedy randomized using all possible routing paths\n');
109 t= tic;
110 bestEnergyGreedy= inf;
111 sol= zeros(1,nFlows);
112 allValues= [];
113 while toc(t)<10
114     ax2 = randperm(nFlows);
115     sol= zeros(1,nFlows);
116     for i= ax2
117         k_best = 0;
118         best = inf;
119         %encontrar o melhor vizinho
120         for k = 1:nSP(i)
121             sol(i)= k;
122             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
123             load= max(max(Loads(:,3:4)));
124
125             if load<=10
126                 energy = 0;
127                 for a=1:nLinks
128                     if Loads(a,3)+Loads(a,4)>0
129                         energy = energy + L(Loads(a,1),Loads(a,2));
130                     end
131                 end
132             else
133                 energy=inf;
134             end
135
136             if energy < best
137                 k_best = k;
138                 best = energy;
139             end
140         end
141         sol(i) = k_best;
142     end
143
144     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
145     load= max(max(Loads(:,3:4)));
146     if load<=10
147         energy = 0;
148         for a=1:nLinks
149             if Loads(a,3)+Loads(a,4)>0
150                 energy = energy + L(Loads(a,1),Loads(a,2));
151             end
152         end

```



```

153     else
154         energy=inf;
155     end
156
157     allValues= [allValues energy];
158     if energy <bestEnergyGreedy
159         bestSol= sol;
160         bestEnergyGreedy= energy;
161     end
162 end
163 hold on
164 grid on
165 plot(sort(allValues));
166 fprintf('Energy consumption value of the best solution = %.1f \n',
        bestEnergyGreedy);
167 fprintf('Number of solutions generated = %d \n', length(allValues));
168 fprintf('Average quality of all solutions generated = %.1f \n', mean(
        allValues));
169
170 fprintf('\nSolution hill climbing using all possible routing paths\n');
171 %Optimization algorithm with multi start hill climbing:
172 t= tic;
173 bestEnergy= inf;
174 allValues= [];
175 contadortotal= [];
176 while toc(t)<tempo
177     %GREEDY RANDOMIZED:
178     continuar= true;
179     while continuar
180         continuar= false;
181         ax2= randperm(nFlows);
182         sol= zeros(1,nFlows);
183         for i= ax2
184             k_best= 0;
185             best= inf;
186             for k= 1:nSP(i)
187                 sol(i)= k;
188                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
189                 load= max(max(Loads(:,3:4)));
190                 if load <= 10
191                     energy= 0;
192                     for a= 1:nLinks
193                         if Loads(a,3)+Loads(a,4)>0
194                             energy= energy + L(Loads(a,1),Loads(a,2));
195                         end
196                     end
197                 else
198                     energy= inf;
199                 end
200                 if energy<best
201                     k_best= k;
202                     best= energy;
203                 end
204             end
205             if k_best>0

```

```

2206         sol(i)= k_best;
2207     else
2208         continuar= true;
2209         break;
2210     end
2211 end
2212 end
2213 energy= best;
2214
2215 %HILL CLIMBING:
2216 continuar= true;
2217 while continuar
2218     i_best= 0;
2219     k_best= 0;
2220     best= energy;
2221     for i= 1:nFlows
2222         for k= 1:nSP(i)
2223             if k~=sol(i)
2224                 aux= sol(i);
2225                 sol(i)= k;
2226                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
2227                 load1= max(max(Loads(:,3:4)));
2228                 if load1 <= 10
2229                     energy1= 0;
2230                     for a= 1:nLinks
2231                         if Loads(a,3)+Loads(a,4)>0
2232                             energy1= energy1 + L(Loads(a,1),Loads(a,2))
2233                                     ;
2234                         end
2235                     end
2236                     energy1= inf;
2237                 end
2238                 if energy1<best
2239                     i_best= i;
2240                     k_best= k;
2241                     best= energy1;
2242                 end
2243                 sol(i)= aux;
2244             end
2245         end
2246     end
2247     if i_best>0
2248         sol(i_best)= k_best;
2249         energy= best;
2250     else
2251         continuar= false;
2252     end
2253 end
2254 allValues= [allValues energy];
2255 if energy<bestEnergy
2256     bestSol= sol;
2257     bestEnergy= energy;
2258 end
2259 end

```

```

260 hold on
261 grid on
262 plot(sort(allValues));
263 legend('Random using all possible','Greedy randomized using all possible','
    Hill climbing using all possible',Location="southeast");
264 fprintf('    Best energy = %.1f\n',bestEnergy);
265 fprintf('    No. of solutions = %d\n',length(allValues));
266 fprintf('    Av. quality of solutions = %.1f\n',mean(allValues));
267
268
269 fprintf('\nSolution random with 10 shortest routing paths\n');
270 t= tic;
271 bestEnergy= inf;
272 sol= zeros(1,nFlows);
273 allValues= [];
274 while toc(t)<10
275     for i= 1:nFlows
276         n = min(10,nSP(i));
277         sol(i)= randi(n);
278     end
279     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
280     load= max(max(Loads(:,3:4)));
281     if load<=10
282         energy = 0;
283         for a=1:nLinks
284             if Loads(a,3)+Loads(a,4)>0
285                 energy = energy + L(Loads(a,1),Loads(a,2));
286             end
287         end
288     else
289         energy=inf;
290     end
291     allValues= [allValues energy];
292     if energy<bestEnergy
293         bestSol= sol;
294         bestEnergy= energy;
295     end
296 end
297 figure(2);
298 grid on
299 plot(sort(allValues));
300 title('Compare the algorithms – Task 2.D')
301 fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergy);
302 fprintf('Number of solutions generated = %d \n', length(allValues));
303 fprintf('Average quality of all solutions generated = %.1f \n', mean(
    allValues));
304
305 fprintf('\nSolution greedy randomized using 10 shortest routing paths\n');
306 t= tic;
307 bestEnergyGreedy= inf;
308 sol= zeros(1,nFlows);
309 allValues= [];
310 while toc(t)<10
311     ax2 = randperm(nFlows);

```

```

312     sol= zeros(1,nFlows);
313     for i= ax2
314         k_best = 0;
315         best = inf;
316         %encontrar o melhor vizinho
317         n = min(10,nSP(i));
318         for k = 1:n
319             sol(i)= k;
320             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
321             load= max(max(Loads(:,3:4)));
322
323             if load<=10
324                 energy = 0;
325                 for a=1:nLinks
326                     if Loads(a,3)+Loads(a,4)>0
327                         energy = energy + L(Loads(a,1),Loads(a,2));
328                     end
329                 end
330             else
331                 energy=inf;
332             end
333
334             if energy < best
335                 k_best = k;
336                 best = energy;
337             end
338         end
339         sol(i) = k_best;
340     end
341     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
342     load= max(max(Loads(:,3:4)));
343     if load<=10
344         energy = 0;
345         for a=1:nLinks
346             if Loads(a,3)+Loads(a,4)>0
347                 energy = energy + L(Loads(a,1),Loads(a,2));
348             end
349         end
350     else
351         energy=inf;
352     end
353
354     allValues= [allValues energy];
355     if energy <bestEnergyGreedy
356         bestSol= sol;
357         bestEnergyGreedy= energy;
358     end
359 end
360
361 hold on
362 grid on
363 plot(sort(allValues));
364 fprintf('Energy consumption value of the best solution = %.1f \n',
        bestEnergyGreedy);
365 fprintf('Number of solutions generated = %d \n', length(allValues));

```

```

366 fprintf('Average quality of all solutions generated = %.1f \n', mean(
    allValues));
367
368 fprintf('\nSolution hill climbing using 10 shortest routing paths\n');
369 %Optimization algorithm with multi start hill climbing:
370 t= tic;
371 bestEnergy= inf;
372 allValues= [];
373 contadortotal= [];
374 while toc(t)<tempo
375     %GREEDY RANDOMIZED:
376     continuar= true;
377     while continuar
378         continuar= false;
379         ax2= randperm(nFlows);
380         sol= zeros(1,nFlows);
381         for i= ax2
382             k_best= 0;
383             best= inf;
384             n = min(10,nSP(i));
385             for k= 1:n
386                 sol(i)= k;
387                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
388                 load= max(max(Loads(:,3:4)));
389                 if load <= 10
390                     energy= 0;
391                     for a= 1:nLinks
392                         if Loads(a,3)+Loads(a,4)>0
393                             energy= energy + L(Loads(a,1),Loads(a,2));
394                         end
395                     end
396                 else
397                     energy= inf;
398                 end
399                 if energy<best
400                     k_best= k;
401                     best= energy;
402                 end
403             end
404             if k_best>0
405                 sol(i)= k_best;
406             else
407                 continuar= true;
408                 break;
409             end
410         end
411     end
412     energy= best;
413
414     %HILL CLIMBING:
415     continuar= true;
416     while continuar
417         i_best= 0;
418         k_best= 0;
419         best= energy;

```

```

420     for i= 1:nFlows
421         for k= 1:nSP(i)
422             if k~=sol(i)
423                 aux= sol(i);
424                 sol(i)= k;
425                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
426                 load1= max(max(Loads(:,3:4)));
427                 if load1 <= 10
428                     energy1= 0;
429                     for a= 1:nLinks
430                         if Loads(a,3)+Loads(a,4)>0
431                             energy1= energy1 + L(Loads(a,1),Loads(a,2))
432                                     ;
433                         end
434                     else
435                         energy1= inf;
436                     end
437                     if energy1<best
438                         i_best= i;
439                         k_best= k;
440                         best= energy1;
441                     end
442                     sol(i)= aux;
443                 end
444             end
445         end
446         if i_best>0
447             sol(i_best)= k_best;
448             energy= best;
449         else
450             continuar= false;
451         end
452     end
453     allValues= [allValues energy];
454     if energy<bestEnergy
455         bestSol= sol;
456         bestEnergy= energy;
457     end
458 end
459 hold on
460 grid on
461 plot(sort(allValues));
462 legend('Random with 10 shortest','Greedy randomized with 10 shortest','Hill
         climbing with 10 shortest',Location="southeast");
463 fprintf('    Best energy = %.1f\n',bestEnergy);
464 fprintf('    No. of solutions = %d\n',length(allValues));
465 fprintf('    Av. quality of solutions = %.1f\n',mean(allValues));
466
467 fprintf('\nSolution random with 5 shortest routing paths\n');
468 t= tic;
469 bestEnergy= inf;
470 sol= zeros(1,nFlows);
471 allValues= [];
472 while toc(t)<10

```

```

473     for i= 1:nFlows
474         n = min(5,nSP(i));
475         sol(i)= randi(n);
476     end
477     Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
478     load= max(max(Loads(:,3:4)));
479     if load<=10
480         energy = 0;
481         for a=1:nLinks
482             if Loads(a,3)+Loads(a,4)>0
483                 energy = energy + L(Loads(a,1),Loads(a,2));
484             end
485         end
486     else
487         energy=inf;
488     end
489     allValues= [allValues energy];
490     if energy<bestEnergy
491         bestSol= sol;
492         bestEnergy= energy;
493     end
494
495 end
496 figure(3);
497 grid on
498 plot(sort(allValues));
499 title('Compare the algorithms – Task 2.D')
500 fprintf('Energy consumption value of the best solution = %.1f \n',
        bestEnergy);
501 fprintf('Number of solutions generated = %d \n', length(allValues));
502 fprintf('Average quality of all solutions generated = %.1f \n', mean(
        allValues));
503
504 fprintf('\nSolution greedy randomized using 5 shortest routing paths\n');
505 t= tic;
506 bestEnergyGreedy= inf;
507 sol= zeros(1,nFlows);
508 allValues= [];
509 while toc(t)<10
510     ax2 = randperm(nFlows);
511     sol= zeros(1,nFlows);
512     for i= ax2
513         k_best = 0;
514         best = inf;
515         %encontrar o melhor vizinho
516         n = min(5,nSP(i));
517         for k = 1:n
518             sol(i)= k;
519             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
520             load= max(max(Loads(:,3:4)));
521
522             if load<=10
523                 energy = 0;
524                 for a=1:nLinks
525                     if Loads(a,3)+Loads(a,4)>0

```

```

526             energy = energy + L(Loads(a,1),Loads(a,2));
527         end
528     end
529     else
530         energy=inf;
531     end
532
533     if energy < best
534         k_best = k;
535         best = energy;
536     end
537 end
538 sol(i) = k_best;
539 end
540 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
541 load= max(max(Loads(:,3:4)));
542 if load<=10
543     energy = 0;
544     for a=1:nLinks
545         if Loads(a,3)+Loads(a,4)>0
546             energy = energy + L(Loads(a,1),Loads(a,2));
547         end
548     end
549 else
550     energy=inf;
551 end
552
553 allValues= [allValues energy];
554 if energy <bestEnergyGreedy
555     bestSol= sol;
556     bestEnergyGreedy= energy;
557 end
558 end
559
560 hold on
561 grid on
562 plot(sort(allValues));
563 fprintf('Energy consumption value of the best solution = %.1f \n',
564         bestEnergyGreedy);
564 fprintf('Number of solutions generated = %d \n', length(allValues));
565 fprintf('Average quality of all solutions generated = %.1f \n', mean(
566         allValues));
566
567
568 fprintf('\nSolution hill climbing using 5 shortest routing paths\n');
569 %Optimization algorithm with multi start hill climbing:
570 t= tic;
571 bestEnergy= inf;
572 allValues= [];
573 contadortotal= [];
574 while toc(t)<tempo
575     %GREEDY RANDOMIZED:
576     continuar= true;
577     while continuar
578         continuar= false;

```



```

579     ax2= randperm(nFlows);
580     sol= zeros(1,nFlows);
581     for i= ax2
582         k_best= 0;
583         best= inf;
584         n = min(5,nSP(i));
585         for k= 1:n
586             sol(i)= k;
587             Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
588             load= max(max(Loads(:,3:4)));
589             if load <= 10
590                 energy= 0;
591                 for a= 1:nLinks
592                     if Loads(a,3)+Loads(a,4)>0
593                         energy= energy + L(Loads(a,1),Loads(a,2));
594                     end
595                 end
596             else
597                 energy= inf;
598             end
599             if energy<best
600                 k_best= k;
601                 best= energy;
602             end
603         end
604         if k_best>0
605             sol(i)= k_best;
606         else
607             continuar= true;
608             break;
609         end
610     end
611 end
612 energy= best;
613
614 %HILL CLIMBING:
615 continuar= true;
616 while continuar
617     i_best= 0;
618     k_best= 0;
619     best= energy;
620     for i= 1:nFlows
621         for k= 1:nSP(i)
622             if k~=sol(i)
623                 aux= sol(i);
624                 sol(i)= k;
625                 Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
626                 load1= max(max(Loads(:,3:4)));
627                 if load1 <= 10
628                     energy1= 0;
629                     for a= 1:nLinks
630                         if Loads(a,3)+Loads(a,4)>0
631                             energy1= energy1 + L(Loads(a,1),Loads(a,2))
632                             ;
633                         end

```

```

633         end
634     else
635         energy1= inf;
636     end
637     if energy1<best
638         i_best= i;
639         k_best= k;
640         best= energy1;
641     end
642     sol(i)= aux;
643 end
644 end
645 end
646 if i_best>0
647     sol(i_best)= k_best;
648     energy= best;
649 else
650     continuar= false;
651 end
652 end
653 allValues= [allValues energy];
654 if energy<bestEnergy
655     bestSol= sol;
656     bestEnergy= energy;
657 end
658 end
659 hold on
660 grid on
661 plot(sort(allValues));
662 legend('Random with 5 shortest','Greedy randomized with 5 shortest','Hill
        climbing with 5 shortest',Location="southeast");
663
664 fprintf('    Best energy = %.1f\n',bestEnergy);
665 fprintf('    No. of solutions = %d\n',length(allValues));
666 fprintf('    Av. quality of solutions = %.1f\n',mean(allValues));

```

## 2.d.2 Resultados e Conclusões

Através do código mostrado anteriormente, pode-se ver que o algoritmo hill climbing é bastante mais complexo que o greedy randomized que por sua vez é mais complexo do que o algoritmo random. É esperado que a solução hill climbing tenha a melhor solução mais rapidamente que o algoritmo greedy randomized que por sua vez tenha melhores resultados que o algoritmo random, como verificado na task anterior.

Quando se compara a solução random com a greedy randomized em redes grandes, apesar da solução greedy ser mais pesada computacionalmente que a estratégia random, esta complexidade extra compensa, pois os valores do algoritmo greedy randomized são melhores que os do algoritmo random. Isto também acontece para redes pequenas (neste caso só se tem 9 fluxos). Na Figura 2.7 pode-se observar que se obteve melhores resultados com melhor média da qualidade das soluções geradas no algoritmo greedy randomized (1235.0) que no algoritmo random (2189.0). Isto também pode ser visto através do Gráfico 2.8.

Quando se compara o algoritmo hill climbing com o algoritmo random, pode se observar que este melhorou a solução final entre estes dois algoritmos, mas por outro lado, é de notar que o algoritmo random em 10 segundos gerou 143474 soluções enquanto que o algoritmo hill climbing apenas gerou 1935. Isto comprava que o algoritmo hill climbing é bastante mais complexo que o random.

Finalmente, quando se compara o algoritmo greedy randomized com o hill climbing em redes grandes,

## Task 2 - Alinea D

Solution random with all possible routing paths  
 Energy consumption value of the best solution = 2189.0  
 Number of solutions generated = 143474  
 Average quality of all solutions generated = Inf

Solution greedy randomized using all possible routing paths  
 Energy consumption value of the best solution = 1235.0  
 Number of solutions generated = 5145  
 Average quality of all solutions generated = 1398.5

Solution hill climbing using all possible routing paths  
 Best energy = 1235.0  
 No. of solutions = 1935  
 Av. quality of solutions = 1374.3

Solution random with 10 shortest routing paths  
 Energy consumption value of the best solution = 1701.0  
 Number of solutions generated = 145087  
 Average quality of all solutions generated = Inf

Solution greedy randomized using 10 shortest routing paths  
 Energy consumption value of the best solution = 1235.0  
 Number of solutions generated = 15834  
 Average quality of all solutions generated = 1415.9

Solution hill climbing using 10 shortest routing paths  
 Best energy = 1235.0  
 No. of solutions = 2503  
 Av. quality of solutions = 1373.3

Solution random with 5 shortest routing paths  
 Energy consumption value of the best solution = 1394.0  
 Number of solutions generated = 145180  
 Average quality of all solutions generated = Inf

Solution greedy randomized using 5 shortest routing paths  
 Energy consumption value of the best solution = 1303.0  
 Number of solutions generated = 32163  
 Average quality of all solutions generated = 1512.1

Solution hill climbing using 5 shortest routing paths  
 Best energy = 1235.0  
 No. of solutions = 1806  
 Av. quality of solutions = 1388.6

Figura 2.7: Resultados numéricos para a alínea D nos diferentes casos.

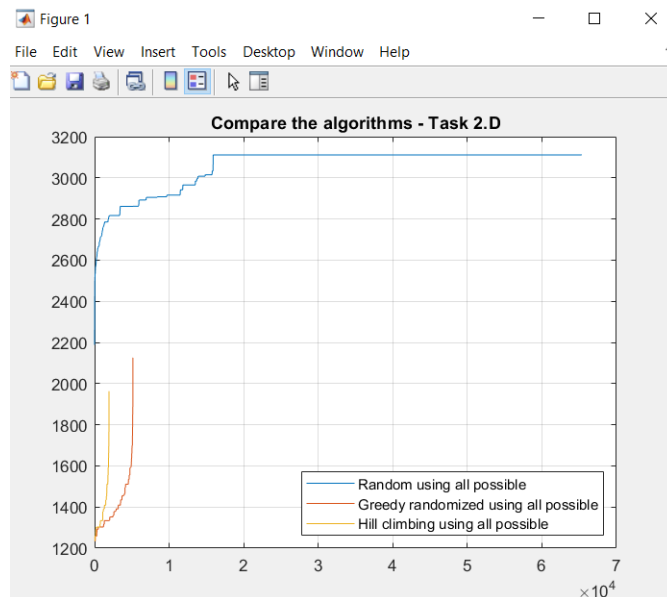


Figura 2.8: Resultados gráficos para a alínea D no caso de ter todas as rotas para os diferentes algoritmos.

apesar do algoritmo hill climbing ser mais complexo e pesado computacionalmente que o algoritmo greedy randomized, esta complexidade extra compensa, pois os valores do algoritmo hill climbing encontra a melhor solução mais rapidamente e os seus melhores valores são melhores que os do algoritmo greedy randomized. Neste caso, como é uma rede pequena (9 fluxos), o melhor valor do algoritmo hill climbing é igual ao do algoritmo greedy randomized, como se pode ver na Figura 2.7. Isto acontece, porque cada algoritmo só corre 10 segundos e como o algoritmo hill climbing é muito complexo, demora mais tempo a gerar soluções tendo menos soluções que o algoritmo greedy randomized. Também se pode ver que a média da qualidade de todas as soluções geradas no hill climbing é maior que no greedy randomized, pois o hill climbing precisa sempre de menos soluções para chegar á melhor solução.

Posto isto, pode-se concluir que para redes pequenas o algoritmo hill climbing é demasiado complexo,

não sendo vantajoso quando se compara com o algoritmo greedy randomized, pois gera menos soluções que o algoritmo greedy randomized e ao mesmo tempo obtém o mesmo resultado que o algoritmo greedy randomized.

Com isto tudo, pode-se concluir que o algoritmo random gera muito mais soluções que o greedy randomized e que o hill climbing. No algoritmo greedy randomized existe uma melhoria significativa na melhor solução quando comparado com o algoritmo random, ou seja a complexidade acrescida que o algoritmo greedy randomized tem em relação ao random, compensa. O algoritmo hill climbing não obteve melhoria quando comparado com o algoritmo greedy randomized. Como este é mais complexo que o greedy randomized pode-se dizer que o algoritmo hill climbing não traz vantagens para calcular a energia comparado com o algoritmo greedy randomized.

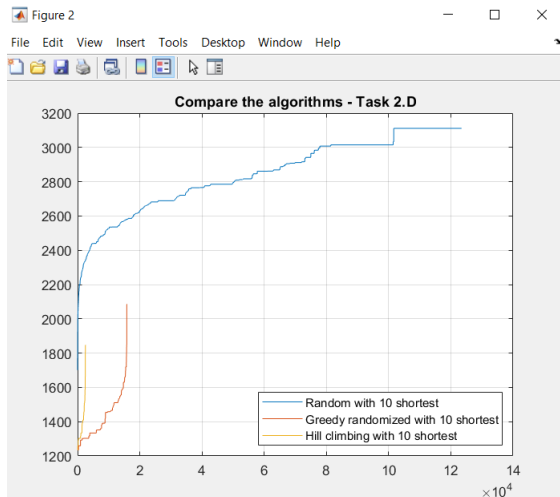


Figura 2.9: Resultados gráficos para a alínea D no caso de ter 10 rotas mais curtas para os diferentes algoritmos.

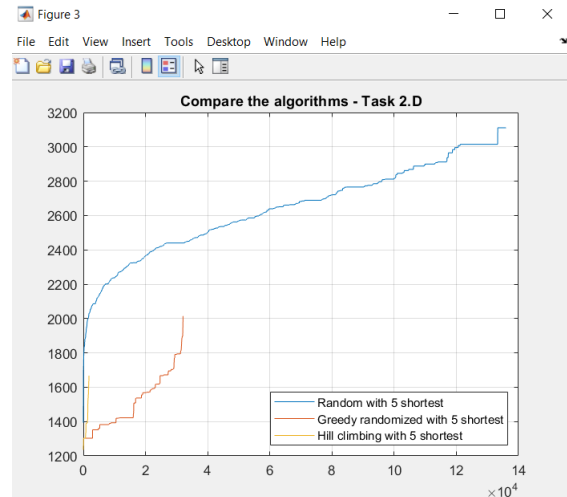


Figura 2.10: Resultados gráficos para a alínea D no caso de ter 5 rotas mais curtas para os diferentes algoritmos.

Através dos Gráficos 2.9 e 2.10 pode-se observar que utilizando rotas mais curtas as soluções de cada algoritmo ficam melhores (como dito nos exercícios anteriores), continuando-se com o mencionado anteriormente neste exercício.

# Capítulo 3

## Task 3

Assume that all routers are of very high availability (i.e., their availability is 1.0). Compute the availability of each link based on the length of the link assuming the model considered in J.-P. Vasseur, M. Pickavet and P. Demeester, “Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS”, Elsevier (2004). In this task, the aim is to compute a pair of symmetrical routing paths to support each flow of the unicast service.

### 3.a Task 3.a - Enunciado

For each flow, compute one of its routing paths given by the most available path.

#### 3.a.1 Código

Neste exercício foi pedido para se calcular uma das rotas dadas pelo caminho mais disponível, usado o modelo dado nas aulas.

Primeiramente, calculou-se o MTBF usando a expressão dada nas aulas, availability, a matriz A e um array que guarda os valores de sP, que armazena os caminhos, e nSp, que armazena os custos dos caminhos sP. Começou-se por inicializar a disponibilidade de cada fluxo a 1 como pede o enunciado e a construir um ciclo que percorre-se o número de fluxos existentes, guardando numa variável aux os valores de sP{i} em arrays para se conseguir aceder aos valores de cada caminho. De seguida, tirou-se o número de nós através do número de colunas do aux guardando numa variável chamada arr. Para cada fluxo, fez-se print do 1º caminho, percorrendo o tamanho de aux e calculou-se a sua disponibilidade.

O algoritmo de dijkstra para encontrar o caminho mais curto (ou de menor custo) desde nó de origem usa a ideia de que cada router trata-se como se fosse o router central. Como cada link tem uma métrica associada com o mesmo, o custo da rota desde a origem acaba por ser o sumatório de cada custo individual até se chegar ao nó pretendido. Como neste exercício pede o caminho de cada fluxo de maior probabilidade, ou seja, o 1º caminho encontrado em que cada fluxo tem maior probabilidade de cada elemento estar operacional em qualquer instante, calcula-se fazendo o produto das disponibilidades dos links que pertencem ao percurso calculado com menor custo.

Ao colocar  $n = 1$ , o kShortestPath dá-nos o percurso com custo mínimo, que é o chamado 1º percurso.

O código gerado para a resolução do exercício é o seguinte:

```
1 Nodes= [30 70
2         350 40
3         550 180
4         310 130
5         100 170
6         540 290
7         120 240
8         400 310
9         220 370
```

```

10         550 380];
11 Links= [1 2
12         1 5
13         2 3
14         2 4
15         3 4
16         3 6
17         3 8
18         4 5
19         4 8
20         5 7
21         6 8
22         6 10
23         7 8
24         7 9
25         8 9
26         9 10];
27 T= [1 3 1.0 1.0
28     1 4 0.7 0.5
29     2 7 2.4 1.5
30     3 4 2.4 2.1
31     4 9 1.0 2.2
32     5 6 1.2 1.5
33     5 8 2.1 2.5
34     5 9 1.6 1.9
35     6 10 1.4 1.6];
36 nNodes= 10;
37 nLinks= size(Links,1);
38 nFlows= size(T,1);
39 co= Nodes(:,1)+j*Nodes(:,2);
40 L= inf(nNodes); %Square matrix with arc lengths (in Km)
41 for i=1:nNodes
42     L(i,i)= 0;
43 end
44 for i=1:nLinks
45     d= abs(co(Links(i,1))-co(Links(i,2)));
46     L(Links(i,1),Links(i,2))= d+5; %Km
47     L(Links(i,2),Links(i,1))= d+5; %Km
48 end
49 L= round(L); %Km
50 fprintf("Task 3 - Alinea A\n");
51 MTBF = (450*360*24)./L;
52 A = MTBF./(MTBF+24); % a= availability
53 A(isnan(A))=0; % quando a matriz a tiver Nan mete essa posicao a 0 em vez
do Nan
54 [sP nSP]= calculatePaths(L,T,1); %retorna o 1 caminho para cada link que
e o melhor
55 %sP sao os caminhos e o nSP sao os custos dos caminhos sP
56 av = ones(1,nFlows); %avalibility tudo a 1 - inicializacao
57 for i = 1:nFlows
58     aux = cell2mat(sP{i}); %transforma o {...} num array para conseguirmos
aceder ao que esta dentro dos {}
59     arr = size(aux); % n de linhas n de colunas -> nos queremos os nos = n
de colunas
60     fprintf('Fluxo %d:\n',i);

```

```

61     fprintf('1º caminho: %d', aux(1));
62     for j= 2:length(aux)
63         fprintf('-%d', aux(j));
64     end
65     fprintf("\n");
66     for j = 1:arr(1,2)-1 %percorre o fluxo i -> o array aux
67         av(i) = av(i) * A(aux(j),aux(j+1));
68         %estao sempre em serie, pois da sempre o caminho mais curto no
69         %calculatePaths
70     end
71     aux = av(i)*100;
72     fprintf("Disponibilidade= %f = %f%%\n", av(i), aux);
73 end

```

### 3.a.2 Resultados e Conclusões

Com base nas explicações anteriormente dadas e perante este programa, obteve-se os resultados da Figura 3.1.

```

Task 3 - Alinea A
Fluxo 1:
1º caminho: 1-2-3
Disponibilidade= 0.996460 = 99.646010%
Fluxo 2:
1º caminho: 1-5-4
Disponibilidade= 0.997868 = 99.786769%
Fluxo 3:
1º caminho: 2-4-5-7
Disponibilidade= 0.997535 = 99.753514%
Fluxo 4:
1º caminho: 3-4
Disponibilidade= 0.998459 = 99.845917%
Fluxo 5:
1º caminho: 4-8-9
Disponibilidade= 0.997529 = 99.752928%
Fluxo 6:
1º caminho: 5-7-8-6
Disponibilidade= 0.996810 = 99.680974%
Fluxo 7:
1º caminho: 5-7-8
Disponibilidade= 0.997708 = 99.770809%
Fluxo 8:
1º caminho: 5-7-9
Disponibilidade= 0.998477 = 99.847713%
Fluxo 9:
1º caminho: 6-10
Disponibilidade= 0.999408 = 99.940776%

```

Figura 3.1: Resultados para a task 3 alínea A nos diferentes casos.

Conclui-se que todos os fluxos tiveram um 1º caminho disponível, com uma disponibilidade superior a 99% cada.

### 3.b Task 3.b - Enunciado

For each flow, compute another routing path given by the most available path which is link disjoint with the previously computed routing path. Compute the availability provided by each pair of routing paths. Present all pairs of routing paths of each flow and their availability. Present also the average service availability (i.e., the average availability value among all flows of the service).

#### 3.b.1 Código

Neste exercício foi pedido para se calcular outra das rotas dadas pelo caminho mais disponível, usando o modelo dado nas aulas, neste caso com pares de percursos disjuntos, em que cada fluxo (de um nó origem para um nó destino) é suportado por dois percursos disjuntos (ambos a iniciar no nó origem e a terminar no nó destino do fluxo).

Primeiramente, calculou-se o MTBF usando a expressão dada nas aulas, availability, a matriz A e um array que guarda os valores de sP, que armazena os primeiros caminhos, nSp, que armazena os custos dos primeiros caminhos sP, sP2, que armazena dos segundos caminhos, e nSp2, que armazena os custos dos segundos caminhos. Como fizemos no exercício anterior, começou-se por inicializar a disponibilidade de cada fluxo a 1 como pede o enunciado e por construir um ciclo onde se percorre o número de fluxos existentes, guardando numa variável aux os valores de sPi em arrays para se conseguir aceder aos valores de cada caminho, usando a variável aux e realizando as operações já realizadas anteriormente. Depois de encontrados os 1º caminhos, calculou-se os segundos caminhos, usando a mesma lógica: tirou-se o número de nós através do número de colunas do aux guardando numa variável chamada arr e, para cada fluxo, fez-se um print do 2º caminho, percorrendo o tamanho de aux e calculou-se a sua disponibilidade.

O código gerado para a resolução do exercício é o seguinte:

```
1 Nodes= [30 70
2         350 40
3         550 180
4         310 130
5         100 170
6         540 290
7         120 240
8         400 310
9         220 370
10        550 380];
11 Links= [1 2
12         1 5
13         2 3
14         2 4
15         3 4
16         3 6
17         3 8
18         4 5
19         4 8
20         5 7
21         6 8
22         6 10
23         7 8
24         7 9
25         8 9
26         9 10];
27 T= [1 3 1.0 1.0
28     1 4 0.7 0.5
29     2 7 2.4 1.5
```



```

30     3   4   2.4  2.1
31     4   9   1.0  2.2
32     5   6   1.2  1.5
33     5   8   2.1  2.5
34     5   9   1.6  1.9
35     6  10   1.4  1.6];
36 nNodes= 10;
37 nLinks= size(Links,1);
38 nFlows= size(T,1);
39 co= Nodes(:,1)+j*Nodes(:,2);
40 L= inf(nNodes);    %Square matrix with arc lengths (in Km)
41 for i=1:nNodes
42     L(i,i)= 0;
43 end
44 for i=1:nLinks
45     d= abs(co(Links(i,1))-co(Links(i,2)));
46     L(Links(i,1),Links(i,2))= d+5; %Km
47     L(Links(i,2),Links(i,1))= d+5; %Km
48 end
49 L= round(L); %Km
50 fprintf("Task 3 - Alinea B\n");
51 MTBF = (450*360*24)./L;
52 A = MTBF./(MTBF+24); % a= availability
53 A(isnan(A))=0; % quando a matriz a tiver Nan mete essa posicao a 0 em vez
    do Nan
54 AuxL = -log(A)*100;
55 [sP nSP sP2 nSP2]= calculateDisjointPaths(AuxL,T);
56 for i=1:nFlows
57     aux = cell2mat(sP{1});
58     arr = size(aux);
59     for j=2:length(aux)
60         AuxL(aux(j),aux(j-1))= inf;
61         AuxL(aux(j-1),aux(j))= inf;
62     end
63 end
64 %sP sao os caminhos e o nSp sao os custos dos caminhos sP
65 av = ones(1,nFlows); %avalibility tudo a 1 - inicializacao
66 for i = 1:nFlows
67     aux = cell2mat(sP2{i}); %transforma o {...} num array para conseguirmos
        aceder ao que esta dentro dos {}
68     arr = size(aux); % n de linhas n de colunas -> nos queremos os nos = n
        de colunas
69     fprintf('Fluxo %d:',i);
70     if ~isempty(sP2{i}{1})
71         fprintf('\nSegundo caminho: %d',sP2{i}{1}(1));
72         for j= 2:length(sP2{i}{1})
73             fprintf('-%d',sP2{i}{1}(j));
74         end
75         for j = 1:arr(1,2)-1 %percorre o fluxo i -> o array aux
76             av(i) = av(i) * A(aux(j),aux(j+1));
77             %estao sempre em serie, pois da sempre o caminho mais curto no
78             %calculatePaths
79         end
80         aux = av(i)*100;
81         fprintf("\nDisponibilidade= %f = %f%%\n",av(i),aux);

```

82       end  
83   end

### 3.b.2 Resultados e Conclusões

Perante este programa obteve-se os resultados da Figura 3.2.

```
Task 3 - Alinea B
Fluxo 1:
Segundo caminho: 1-5-4-3
Disponibilidade= 0.996330 = 99.633015%
Fluxo 2:
Segundo caminho: 1-2-4
Disponibilidade= 0.997358 = 99.735757%
Fluxo 3:
Segundo caminho: 2-3-8-7
Disponibilidade= 0.995409 = 99.540924%
Fluxo 4:
Segundo caminho: 3-2-4
Disponibilidade= 0.997831 = 99.783090%
Fluxo 5:
Segundo caminho: 4-5-7-9
Disponibilidade= 0.997129 = 99.712916%
Fluxo 6:
Segundo caminho: 5-4-3-6
Disponibilidade= 0.996404 = 99.640390%
Fluxo 7:
Segundo caminho: 5-4-8
Disponibilidade= 0.997382 = 99.738170%
Fluxo 8:
Segundo caminho: 5-4-8-9
Disponibilidade= 0.996183 = 99.618259%
Fluxo 9:
Segundo caminho: 6-8-9-10
Disponibilidade= 0.995839 = 99.583911%
```

Figura 3.2: Resultados para a task 3 alínea B nos diferentes casos.

Perante os resultados pode-se afirmar que os resultados são disjuntos nos nós e nas ligações, pois ao comparar os resultados com os resultados da alínea anterior verifica-se que não existem ligações nem existe nós intermédios comuns entre o 1º e o 2º caminho. Isto faz com que se proteja o fluxo para todas as falhas individuais de um elemento (nó ou ligação) exceto a falha do nó origem ou do nó destino do fluxo.

## 3.c Task 3.c - Enunciado

Recall that the capacity of all links is 10 Gbps in each direction. Compute how much bandwidth is required on each direction of each link to support all flows with 1+1 protection using the previous computed pairs of link disjoint paths. Compute also the total bandwidth required on all links. Register which links do not have enough capacity.

### 3.c.1 Código

Neste exercício, era pedido para calcular quanta banda larga era necessária em cada direção de cada link de forma a suportar todos os fluxos com a proteção 1+1.

Usando o código utilizado anteriormente e usando as fórmulas dadas na aula calculou-se a banda larga para cada um dos links e depois calculou-se a soma da banda larga necessária de todos os links.

O código gerado para a resolução do exercício é o seguinte:

```

1  Nodes= [30 70
2          350 40
3          550 180
4          310 130
5          100 170
6          540 290
7          120 240
8          400 310
9          220 370
10         550 380];
11 Links= [1 2
12         1 5
13         2 3
14         2 4
15         3 4
16         3 6
17         3 8
18         4 5
19         4 8
20         5 7
21         6 8
22         6 10
23         7 8
24         7 9
25         8 9
26         9 10];
27 T= [1 3 1.0 1.0
28      1 4 0.7 0.5
29      2 7 2.4 1.5
30      3 4 2.4 2.1
31      4 9 1.0 2.2
32      5 6 1.2 1.5
33      5 8 2.1 2.5
34      5 9 1.6 1.9
35      6 10 1.4 1.6];
36 nNodes= 10;
37 nLinks= size(Links,1);
38 nFlows= size(T,1);
39 co= Nodes(:,1)+j*Nodes(:,2);
40 L= inf(nNodes); %Square matrix with arc lengths (in Km)
41 for i=1:nNodes
42     L(i,i)= 0;
43 end
44 for i=1:nLinks
45     d= abs(co(Links(i,1))-co(Links(i,2)));
46     L(Links(i,1),Links(i,2))= d+5; %Km
47     L(Links(i,2),Links(i,1))= d+5; %Km
48 end
49 L= round(L); %Km
50 fprintf("Task 3 - Alinea B\n");
51 MTBF = (450*360*24)./L;
52 A = MTBF./(MTBF+24); % a= availability
53 A(isnan(A))=0; % quando a matriz a tiver Nan mete essa posicao a 0 em vez
54     do Nan
55 AuxL = -log(A)*100;

```

```

55 [sP nSP sP2 nSP2]= calculateDisjointPaths(AuxL,T);
56 for i=1:nFlows
57     aux = cell2mat(sP{1});
58     arr = size(aux);
59     for j=2:length(aux)
60         AuxL(aux(j),aux(j-1))= inf;
61         AuxL(aux(j-1),aux(j))= inf;
62     end
63 end
64 %sP sao os caminhos e o nSP sao os custos dos caminhos sP
65 av = ones(1,nFlows); %avalibility tudo a 1 - inicializacao
66 for i = 1:nFlows
67     aux = cell2mat(sP2{i}); %transforma o {...} num array para conseguirmos
        aceder ao que esta dentro dos {}
68     arr = size(aux); % n de linhas n de colunas -> nos queremos os nos = n
        de colunas
69     fprintf('Fluxo %d:',i);
70     if ~isempty(sP2{i}{1})
71         fprintf('\nSegundo caminho: %d',sP2{i}{1}(1));
72         for j= 2:length(sP2{i}{1})
73             fprintf('-%d',sP2{i}{1}(j));
74         end
75         for j = 1:arr(1,2)-1 %percorre o fluxo i -> o array aux
76             av(i) = av(i) * A(aux(j),aux(j+1));
77             %estao sempre em serie , pois da sempre o caminho mais curto no
78             %calculatePaths
79         end
80         aux = av(i)*100;
81         fprintf("\nDisponibilidade= %f = %f%%\n",av(i),aux);
82     end
83 end
84 bandwidth= calculateLinkLoads1plus1(nNodes,Links,T,sP,sP2)
85 totalbandwidth= sum(sum(bandwidth(:,3:4)))

```

### 3.c.2 Resultados e Conclusões

A proteção 1+1 dita que se houver 2 fluxos que passam pelos mesmos routers soma-se a banda larga (gbps) que passam por lá pois, neste caso, o fluxo é duplicado pelos 2 percursos e o tempo de recuperação de falhas é muito curto, o que exige muitos recursos da rede.

Perante o programa anterior obteve-se os resultados da Figura 3.3.

```
bandwidth =  
  
1.0000 2.0000 1.7000 1.5000  
1.0000 5.0000 1.7000 1.5000  
2.0000 3.0000 5.5000 4.9000  
2.0000 4.0000 5.5000 4.1000  
3.0000 4.0000 4.9000 4.3000  
3.0000 6.0000 1.2000 1.5000  
3.0000 8.0000 2.4000 1.5000  
4.0000 5.0000 10.8000 10.3000  
4.0000 8.0000 4.7000 6.6000  
5.0000 7.0000 8.3000 9.6000  
6.0000 8.0000 2.9000 2.8000  
6.0000 10.0000 1.4000 1.6000  
7.0000 8.0000 4.8000 6.4000  
7.0000 9.0000 2.6000 4.1000  
8.0000 9.0000 4.0000 5.7000  
9.0000 10.0000 1.4000 1.6000  
  
totalbandwidth =  
  
131.8000
```

Figura 3.3: Resultados para a task 3 alínea C nos diferentes casos.

Como cada link tem a capacidade de 10 Gbps em cada direção, os links que não têm capacidade necessária são todos os que forem maior que 10 Gbps. Nesse caso, e analisando os resultados obtidos, a linha que obtém maior banda larga do que 10 Gbps é a entre o link dos nós 4 e 5, que necessita 10.800 Gbps e 10.300 Gbps, respetivamente.

### 3.d Task 3.d - Enunciado

Compute how much bandwidth is required on each link to support all flows with 1:1 protection using the previous computed pairs of link disjoint paths. Compute also the total bandwidth required on all links. Register which links do not have enough capacity and the highest bandwidth value required among all links.

#### 3.d.1 Código

Neste exercício, era pedido para calcular quanta banda larga era necessária em cada direção de cada link de forma a suportar todos os fluxos com a proteção 1:1.

Usando o código utilizado anteriormente e usando as fórmulas dadas calculou-se a banda larga para cada um dos links, calculou-se a soma da banda larga necessária de todos os links e depois o link que precisava de mais banda larga dos calculados.

O código gerado para a resolução do exercício é o seguinte:

```
1 Nodes= [30 70  
2         350 40  
3         550 180
```

```

4         310 130
5         100 170
6         540 290
7         120 240
8         400 310
9         220 370
10        550 380];
11 Links= [1 2
12         1 5
13         2 3
14         2 4
15         3 4
16         3 6
17         3 8
18         4 5
19         4 8
20         5 7
21         6 8
22         6 10
23         7 8
24         7 9
25         8 9
26         9 10];
27 T= [1 3 1.0 1.0
28     1 4 0.7 0.5
29     2 7 2.4 1.5
30     3 4 2.4 2.1
31     4 9 1.0 2.2
32     5 6 1.2 1.5
33     5 8 2.1 2.5
34     5 9 1.6 1.9
35     6 10 1.4 1.6];
36 nNodes= 10;
37 nLinks= size(Links,1);
38 nFlows= size(T,1);
39 co= Nodes(:,1)+j*Nodes(:,2);
40 L= inf(nNodes); %Square matrix with arc lengths (in Km)
41 for i=1:nNodes
42     L(i,i)= 0;
43 end
44 for i=1:nLinks
45     d= abs(co(Links(i,1))-co(Links(i,2)));
46     L(Links(i,1),Links(i,2))= d+5; %Km
47     L(Links(i,2),Links(i,1))= d+5; %Km
48 end
49 L= round(L); %Km
50 fprintf("Task 3 - Alinea B\n");
51 MITBF = (450*360*24)./L;
52 A = MITBF./(MITBF+24); % a= availability
53 A(isnan(A))=0; % quando a matriz a tiver Nan mete essa posicao a 0 em vez
    do Nan
54 AuxL = -log(A)*100;
55 [sP nSP sP2 nSP2]= calculateDisjointPaths(AuxL,T);
56 for i=1:nFlows
57     aux = cell2mat(sP{1});

```

```

58     arr = size(aux);
59     for j=2:length(aux)
60         AuxL(aux(j),aux(j-1))= inf;
61         AuxL(aux(j-1),aux(j))= inf;
62     end
63 end
64 %sP sao os caminhos e o nSp sao os custos dos caminhos sP
65 av = ones(1,nFlows); %avalibility tudo a 1 – inicializacao
66 for i = 1:nFlows
67     aux = cell2mat(sP2{i}); %transforma o {...} num array para conseguirmos
        aceder ao que esta dentro dos {}
68     arr = size(aux); % n de linhas n de colunas -> nos queremos os nos = n
        de colunas
69     fprintf('Fluxo %d:',i);
70     if ~isempty(sP2{i}{1})
71         fprintf('\nSegundo caminho: %d',sP2{i}{1}(1));
72         for j= 2:length(sP2{i}{1})
73             fprintf('-%d',sP2{i}{1}(j));
74         end
75         for j = 1:arr(1,2)-1 %percorre o fluxo i -> o array aux
76             av(i) = av(i) * A(aux(j),aux(j+1));
77             %estao sempre em serie , pois da sempre o caminho mais curto no
78             %calculatePaths
79         end
80         aux = av(i)*100;
81         fprintf("\nDisponibilidade= %f = %f%%\n",av(i),aux);
82     end
83 end
84 bandwidth= calculateLinkLoads1to1(nNodes,Links,T,sP,sP2)
85 totalbandwidth= sum(sum(bandwidth(:,3:4)))
86 [~, index] = max(bandwidth(:,3:4));
87 maxbandwidth = bandwidth(index,:);

```

### 3.d.2 Resultados e Conclusões

Usando a proteção 1:1, o fluxo é enviado por um dos percursos (percurso de serviço) e o outro (percurso de proteção) só é usado em caso de falha do primeiro. Contudo, o tempo de recuperação é grande e os recursos de proteção podem ser partilhados entre diferentes fluxos. Se houver 2 fluxos que passam pelos mesmos routers, escolheu-se os que têm valores de banda larga máximo que passa por lá.

Perante este programa obteve-se os resultados da Figura 3.4.

bandwidth =			
1.0000	2.0000	1.7000	1.5000
1.0000	5.0000	1.7000	1.5000
2.0000	3.0000	3.4000	3.4000
2.0000	4.0000	4.8000	3.6000
3.0000	4.0000	3.9000	3.3000
3.0000	6.0000	1.2000	1.5000
3.0000	8.0000	2.4000	1.5000
4.0000	5.0000	6.9000	5.6000
4.0000	8.0000	4.7000	6.6000
5.0000	7.0000	8.3000	9.6000
6.0000	8.0000	2.9000	2.8000
6.0000	10.0000	1.4000	1.6000
7.0000	8.0000	4.8000	6.4000
7.0000	9.0000	2.6000	4.1000
8.0000	9.0000	2.6000	4.1000
9.0000	10.0000	1.4000	1.6000
totalbandwidth =			
113.4000			
maxbandwidth =			
5.0000	7.0000	8.3000	9.6000
5.0000	7.0000	8.3000	9.6000

Figura 3.4: Resultados para a task 3 alínea D nos diferentes casos.

Como cada link tem a capacidade de 10 Gbps em cada direção, os links que não têm capacidade necessária são todos os que forem maior que 10 Gbps. Nesse caso, e analisando os resultados obtidos, todos os links têm capacidade e o link que necessita maior banda larga é entre os nós 5 e 7.

## 3.e Task 3.e - Enunciado

Compare the results of 3.c and 3.d and justify the differences.

### 3.e.1 Resultados e Conclusões

Comparando os resultados da alínea c e d, consegue-se entender que o desempenho de falhas do 1+1 é mais rápido que o 1:1, sendo uma das razões das diferenças nos valores entre as duas alíneas. Enquanto que 1+1 simplesmente duplica os dados pelo dois caminhos de cada fluxo e portanto, a largura de banda necessária é a soma de cada fluxo, o 1:1 faz partilha de recursos fazendo com que em alguns links, a largura de banda necessária seja menor.



Outra situação que poderá acontecer é um caminho já estar a ser utilizado por outro fluxo e o fluxo seguinte nunca chegar ao seu destino, daí os mecanismos de proteção serem tão importantes.

# Capítulo 4

## Task 4

Consider the same availability values as in Task 3. In this task, the aim is to compute a pair of symmetrical routing paths to support each flow of the unicast service with 1:1 protection which minimizes the highest required bandwidth value among all links.

### 4.a Task 4.a - Enunciado

For each flow, compute 10 pairs of link disjoint paths in the following way. With a k-shortest path algorithm, first compute the  $k = 10$  most available routing paths provided by the network to each traffic flow. Then, compute the most available path which is link disjoint with each of the  $k$  previous paths.

#### 4.a.1 Código

Para este exercício, foi pedido para calcular 10 caminhos de pares disjuntos. Para isto calculou-se os 10 primeiros shortest paths com a função `kShortestPath`, a seguir iterou-se sobre o array calculado e criou-se uma matriz temporaria para cada path com nome de *NewAuxL* onde é igual à original (*AuxL*) onde se alterou os links usados no `shortestPath` para “inf”. Depois voltou-se a calcular o `kShortestPath`, mas com a matriz temporaria criada anteriormente e com tamanho 1 e guarda-se os resultados outro array (*shortestPath2*). A partir daqui, fez-se como no exercício 3.d.

O código gerado para a resolução do exercício é o seguinte:

```
1 clear all;
2 close all;
3
4 Nodes= [30 70
5         350 40
6         550 180
7         310 130
8         100 170
9         540 290
10        120 240
11        400 310
12        220 370
13        550 380];
14 Links= [1 2
15         1 5
16         2 3
```

```

17         2 4
18         3 4
19         3 6
20         3 8
21         4 5
22         4 8
23         5 7
24         6 8
25         6 10
26         7 8
27         7 9
28         8 9
29         9 10];
30
31 T= [1  3  1.0  1.0
32      1  4  0.7  0.5
33      2  7  2.4  1.5
34      3  4  2.4  2.1
35      4  9  1.0  2.2
36      5  6  1.2  1.5
37      5  8  2.1  2.5
38      5  9  1.6  1.9
39      6 10  1.4  1.6];
40
41 nNodes= 10;
42 nLinks= size(Links,1);
43 nFlows= size(T,1);
44 co= Nodes(:,1)+j*Nodes(:,2);
45 L= inf(nNodes); %Square matrix with arc lengths (in Km)
46 for i=1:nNodes
47     L(i,i)= 0;
48 end
49 for i=1:nLinks
50     d= abs(co(Links(i,1))-co(Links(i,2)));
51     L(Links(i,1),Links(i,2))= d+5; %Km
52     L(Links(i,2),Links(i,1))= d+5; %Km
53 end
54 L= round(L); %Km
55 fprintf("Task 3 - Alinea A\n");
56 MTBF = (450*360*24)./L;
57 A = MTBF./(MTBF+24); % a= availability
58 A(isnan(A))=0; % quando a matriz a tiver Nan mete essa posicao a 0 em vez
    do Nan
59 AuxL = -log(A);
60 %[sP nSP sP2 nSP2]= calculateDisjointPaths(AuxL,T);
61 shortestPathFinal=[];
62 for i=1:nFlows
63     [shortestPath, totalCost] = kShortestPath(AuxL,T(i,1),T(i,2),10); %
        queremos os 10 percursos com custo minimo
64
65     NewAuxL=AuxL;
66     for k= 1:10
67         tempshortestPath = cell2mat(shortestPath(k));
68
69         for j=2:length(tempshortestPath)

```

```

70         NewAuxL(tempshortestPath(j),tempshortestPath(j-1))= inf;
71         NewAuxL(tempshortestPath(j-1),tempshortestPath(j))= inf;
72     end
73     [shortestPath2, totalCost2] = kShortestPath(NewAuxL,T(i,1),T(i,2)
74         ,1); %queremos os 1 percursos com custo minimo
75     shortestPathFinal = [shortestPathFinal shortestPath2];
76 end
77
78
79 for i = 1:nFlows
80     % arr = size(aux); % n de linhas n de colunas -> nos queremos os nos = n
81     % de colunas
82     fprintf('Fluxo %d:',i);
83     for k=1:10
84         aux = cell2mat(shortestPathFinal(k)); %transforma o {...} num array
85         % para conseguirmos aceder ao que esta dentro dos {}
86         fprintf('\nPath %d : %d',k,aux(1));
87         for j= 2:length(aux)
88             fprintf('-%d',aux(j));
89         end
90         if ~isempty(aux(2))
91             fprintf('\tSecond path: %d',aux(1));
92             for j= 2:length(aux)
93                 fprintf('-%d',aux(j));
94             end
95         else
96             fprintf('No paths');
97         end
98     end
99     fprintf('\n');
100 end
101 fprintf('\n');

```

## 4.a.2 Resultados e Conclusões

Infelizmente como se pode ver nas Figuras 4.1, 4.2 e 4.3 os resultados obtidos não foram os esperados, pois obteve-se os mesmo caminhos para todos os fluxos.

<p>Fluxo 1: Path 1 : 1-5-4-3      Second path: 1-5-4-3 Path 2 : 1-2-4      Second path: 1-2-4 Path 3 : 2-3-8-7      Second path: 2-3-8-7 Path 4 : 2-3-8-7      Second path: 2-3-8-7 Path 5 : 2-3-8-9-7      Second path: 2-3-8-9-7 Path 6 : 3-2-4      Second path: 3-2-4 Path 7 : 3-8-4      Second path: 3-8-4 Path 8 : 3-6-8-7-5-4      Second path: 3-6-8-7-5-4 Path 9 : 4-5-7-9      Second path: 4-5-7-9 Path 10 : 4-3-6-10-9      Second path: 4-3-6-10-9</p> <p>Fluxo 2: Path 1 : 1-5-4-3      Second path: 1-5-4-3 Path 2 : 1-2-4      Second path: 1-2-4 Path 3 : 2-3-8-7      Second path: 2-3-8-7 Path 4 : 2-3-8-7      Second path: 2-3-8-7 Path 5 : 2-3-8-9-7      Second path: 2-3-8-9-7 Path 6 : 3-2-4      Second path: 3-2-4 Path 7 : 3-8-4      Second path: 3-8-4 Path 8 : 3-6-8-7-5-4      Second path: 3-6-8-7-5-4 Path 9 : 4-5-7-9      Second path: 4-5-7-9 Path 10 : 4-3-6-10-9      Second path: 4-3-6-10-9</p>	<p>Fluxo 3: Path 1 : 1-5-4-3      Second path: 1-5-4-3 Path 2 : 1-2-4      Second path: 1-2-4 Path 3 : 2-3-8-7      Second path: 2-3-8-7 Path 4 : 2-3-8-7      Second path: 2-3-8-7 Path 5 : 2-3-8-9-7      Second path: 2-3-8-9-7 Path 6 : 3-2-4      Second path: 3-2-4 Path 7 : 3-8-4      Second path: 3-8-4 Path 8 : 3-6-8-7-5-4      Second path: 3-6-8-7-5-4 Path 9 : 4-5-7-9      Second path: 4-5-7-9 Path 10 : 4-3-6-10-9      Second path: 4-3-6-10-9</p> <p>Fluxo 4: Path 1 : 1-5-4-3      Second path: 1-5-4-3 Path 2 : 1-2-4      Second path: 1-2-4 Path 3 : 2-3-8-7      Second path: 2-3-8-7 Path 4 : 2-3-8-7      Second path: 2-3-8-7 Path 5 : 2-3-8-9-7      Second path: 2-3-8-9-7 Path 6 : 3-2-4      Second path: 3-2-4 Path 7 : 3-8-4      Second path: 3-8-4 Path 8 : 3-6-8-7-5-4      Second path: 3-6-8-7-5-4 Path 9 : 4-5-7-9      Second path: 4-5-7-9 Path 10 : 4-3-6-10-9      Second path: 4-3-6-10-9</p>
---	---

Figura 4.1: Resultados numéricos para a alínea A nos diferentes casos.

<p>Fluxo 5: Path 1 : 1-5-4-3      Second path: 1-5-4-3 Path 2 : 1-2-4      Second path: 1-2-4 Path 3 : 2-3-8-7      Second path: 2-3-8-7 Path 4 : 2-3-8-7      Second path: 2-3-8-7 Path 5 : 2-3-8-9-7      Second path: 2-3-8-9-7 Path 6 : 3-2-4      Second path: 3-2-4 Path 7 : 3-8-4      Second path: 3-8-4 Path 8 : 3-6-8-7-5-4      Second path: 3-6-8-7-5-4 Path 9 : 4-5-7-9      Second path: 4-5-7-9 Path 10 : 4-3-6-10-9      Second path: 4-3-6-10-9</p> <p>Fluxo 6: Path 1 : 1-5-4-3      Second path: 1-5-4-3 Path 2 : 1-2-4      Second path: 1-2-4 Path 3 : 2-3-8-7      Second path: 2-3-8-7 Path 4 : 2-3-8-7      Second path: 2-3-8-7 Path 5 : 2-3-8-9-7      Second path: 2-3-8-9-7 Path 6 : 3-2-4      Second path: 3-2-4 Path 7 : 3-8-4      Second path: 3-8-4 Path 8 : 3-6-8-7-5-4      Second path: 3-6-8-7-5-4 Path 9 : 4-5-7-9      Second path: 4-5-7-9 Path 10 : 4-3-6-10-9      Second path: 4-3-6-10-9</p>	<p>Fluxo 7: Path 1 : 1-5-4-3      Second path: 1-5-4-3 Path 2 : 1-2-4      Second path: 1-2-4 Path 3 : 2-3-8-7      Second path: 2-3-8-7 Path 4 : 2-3-8-7      Second path: 2-3-8-7 Path 5 : 2-3-8-9-7      Second path: 2-3-8-9-7 Path 6 : 3-2-4      Second path: 3-2-4 Path 7 : 3-8-4      Second path: 3-8-4 Path 8 : 3-6-8-7-5-4      Second path: 3-6-8-7-5-4 Path 9 : 4-5-7-9      Second path: 4-5-7-9 Path 10 : 4-3-6-10-9      Second path: 4-3-6-10-9</p> <p>Fluxo 8: Path 1 : 1-5-4-3      Second path: 1-5-4-3 Path 2 : 1-2-4      Second path: 1-2-4 Path 3 : 2-3-8-7      Second path: 2-3-8-7 Path 4 : 2-3-8-7      Second path: 2-3-8-7 Path 5 : 2-3-8-9-7      Second path: 2-3-8-9-7 Path 6 : 3-2-4      Second path: 3-2-4 Path 7 : 3-8-4      Second path: 3-8-4 Path 8 : 3-6-8-7-5-4      Second path: 3-6-8-7-5-4 Path 9 : 4-5-7-9      Second path: 4-5-7-9 Path 10 : 4-3-6-10-9      Second path: 4-3-6-10-9</p>
---	---

Figura 4.2: Resultados numéricos para a alínea A nos diferentes casos.

Fluxo 9:  
Path 1 : 1-5-4-3      Second path: 1-5-4-3  
Path 2 : 1-2-4      Second path: 1-2-4  
Path 3 : 2-3-8-7      Second path: 2-3-8-7  
Path 4 : 2-3-8-7      Second path: 2-3-8-7  
Path 5 : 2-3-8-9-7      Second path: 2-3-8-9-7  
Path 6 : 3-2-4      Second path: 3-2-4  
Path 7 : 3-8-4      Second path: 3-8-4  
Path 8 : 3-6-8-7-5-4      Second path: 3-6-8-7-5-4  
Path 9 : 4-5-7-9      Second path: 4-5-7-9  
Path 10 : 4-3-6-10-9      Second path: 4-3-6-10-9

Figura 4.3: Resultados numéricos para a alínea A nos diferentes casos.

## Capítulo 5

# Contribuições dos autores

Mariana Pinto - 50%

Raquel Pinto - 50%