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

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:

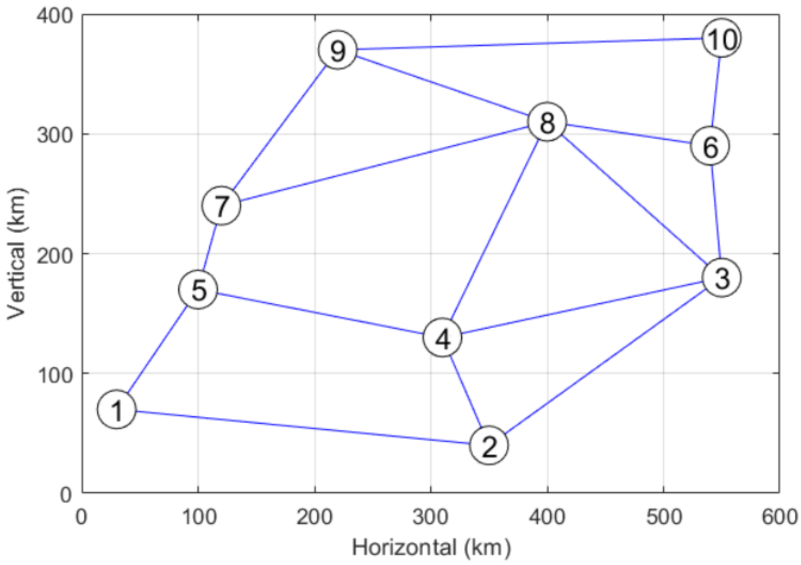

Figura 1: MPSL Network

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 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:

```
fprintf("Task 1 - Alinea A\n");
Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
        540 290
        120 240
        400 310
        220 370
        550 380];
Links= [1 2
         1 5
         2 3
         2 4
         3 4
         3 6
         3 8
```

```
            4  5
            4  8
            5  7
            6  8
            6  10
            7  8
            7  9
            8  9
            9  10];
T= [1   3   1.0  1.0
      1   4   0.7  0.5
      2   7   2.4  1.5
      3   4   2.4  2.1
      4   9   1.0  2.2
      5   6   1.2  1.5
      5   8   2.1  2.5
      5   9   1.6  1.9
      6  10   1.4  1.6];
nNodes= 10;
nLinks= size(Links,1);
nFlows= size(T,1);
B= 625;  %Average packet size in Bytes
co= Nodes(:,1)+j*Nodes(:,2);
L= inf(nNodes);    %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
C= zeros(nNodes);  %Square matrix with arc capacities (in Gbps)
for i=1:nLinks
    C(Links(i,1),Links(i,2))= 10;  %Gbps
    C(Links(i,2),Links(i,1))= 10;  %Gbps
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km
% Compute up to 100 paths for each flow:
n= 100;
[sP nSP]= calculatePaths(L,T,n);
for i = 1:nFlows
    aux = size(sP{i});
    tmp = aux(:,2);
    for j=1:numel(tmp)
        fprintf('Fluxo %d -> Number of different routing paths: %d\n',i,
            tmp(j));
    end
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 algorith during 10 seconds in three cases: (i) using all possbile 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 selecionar um custo random para cada fluxo, calcula-se o tráfego do link com igual custo ao atribuido anteriomente através da função *calculateLinkLoads*. A seguir selecionou-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:

```
clear all;
close all;

fprintf('Task 1 - Alinea B\n');

Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
        540 290
```

```
          120  240
          400  310
          220  370
          550  380];

Links= [1  2
         1  5
         2  3
         2  4
         3  4
         3  6
         3  8
         4  5
         4  8
         5  7
         6  8
         6  10
         7  8
         7  9
         8  9
         9  10];

T= [1   3   1.0  1.0
     1   4   0.7  0.5
     2   7   2.4  1.5
     3   4   2.4  2.1
     4   9   1.0  2.2
     5   6   1.2  1.5
     5   8   2.1  2.5
     5   9   1.6  1.9
     6  10   1.4  1.6];

nNodes= 10;

nLinks= size(Links,1);
nFlows= size(T,1);

B= 625;  %Average packet size in Bytes

co= Nodes(:,1)+j*Nodes(:,2); %calculo para a distancia de cada no

L= inf(nNodes);     %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
C= zeros(nNodes);  %Square matrix with arc capacities (in Gbps)
for i=1:nLinks
    C(Links(i,1),Links(i,2))= 10;   %Gbps
    C(Links(i,2),Links(i,1))= 10;   %Gbps
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km

```

```
% Compute up to 100 paths for each flow:
n= 100;
[sP nSP]= calculatePaths(L,T,n);

%Optimization algorithm resorting to the random strategy:
fprintf('\nSolution random with all possible routing paths\n');
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    for i= 1:nFlows
        sol(i)= randi(nSP(i)); %selecionar um custo random para cada fluxo
    end
    %trafego do link com o custo selecionado
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    %seleciona o que tem maior trafego
    load= max(max(Loads(:,3:4)));
    allValues= [allValues load];
    %ver qual o caminho com o menor trafego
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
figure(1)
grid on
plot(sort(allValues));
title('Random Algorithm - Task 1.B')

fprintf('Worst link load value of the best solution = %.2f Gbps\n',
    bestLoad);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
    allValues));

fprintf('\nSolution random with 10 shortest routing paths\n');
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    for i= 1:nFlows
        n = min(10,nSP(i));
        sol(i)= randi(n);
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
hold on
```

```
grid on
plot(sort(allValues));

fprintf('Worst link load value of the best solution = %.2f Gbps\n', bestLoad);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(allValues));


fprintf('\nSolution random with 5 shortest routing paths\n');
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    for i= 1:nFlows
        n = min(5,nSP(i));
        sol(i)= randi(n);
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
hold on
grid on
plot(sort(allValues));
legend('Random with all possible','Random with 10 shortest','Random with 5 shortest',Location="southeast");
fprintf('Worst link load value of the best solution = %.2f Gbps\n', bestLoad);
fprintf('Number of solutions generated = %d \n', length(allValues));
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 custumam 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 númericos 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.

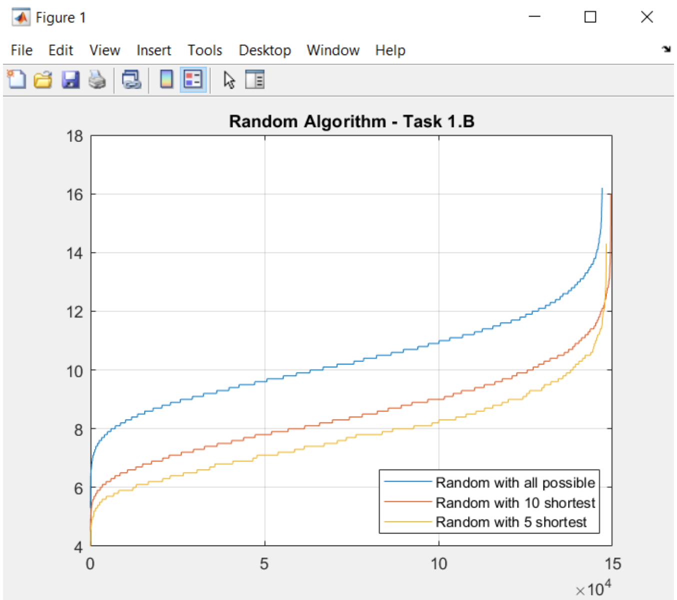

Figura 1.3: Resultados gráficos para a alínea B nos diferentes casos.

A diferença computancionalmente 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 minimo 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 atribuido anteriomente 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:

```
clear all;
close all;

fprintf('Task 1 - Alinea C\n');
Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
```

```
          540  290
          120  240
          400  310
          220  370
          550  380];

Links= [1  2
         1  5
         2  3
         2  4
         3  4
         3  6
         3  8
         4  5
         4  8
         5  7
         6  8
         6  10
         7  8
         7  9
         8  9
         9  10];

T= [1   3   1.0  1.0
     1   4   0.7  0.5
     2   7   2.4  1.5
     3   4   2.4  2.1
     4   9   1.0  2.2
     5   6   1.2  1.5
     5   8   2.1  2.5
     5   9   1.6  1.9
     6  10   1.4  1.6];

nNodes= 10;

nLinks= size(Links,1);
nFlows= size(T,1);

B= 625;  %Average packet size in Bytes

co= Nodes(:,1)+j*Nodes(:,2);

L= inf(nNodes);     %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
C= zeros(nNodes);  %Square matrix with arc capacities (in Gbps)
for i=1:nLinks
    C(Links(i,1),Links(i,2))= 10;  %Gbps
    C(Links(i,2),Links(i,1))= 10;  %Gbps
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km
```

```matlab

% Compute up to 100 paths for each flow:
n= 100;
[sP nSP]= calculatePaths(L,T,n);

fprintf('\nSolution greedy ramdomized using all possible routing paths\n');
%Optimization algorithm resorting to the greedy randomized strategy:
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    ax2 = randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
        k_best = 0;
        best = inf;
        %calculo do custo minimo para os valores aleatorios do fluxo
        for k = 1:nSP(i)
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            if load < best
                k_best = k;
                best = load;
            end
        end
        sol(i) = k_best; %melhor custo para o fluxo i
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end

hold on
grid on
plot(sort(allValues));
title('Greedy Randomized - Task 1.C')
fprintf('Worst link load value of the best solution = %.2f Gbps\n',
    bestLoad);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
    allValues));

fprintf('\nSolution greedy ramdomized using 10 shortest routing paths\n');
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    ax2 = randperm(nFlows);
```

```
    sol= zeros(1,nFlows);
    for i= ax2
        k_best = 0;
        best = inf;
        n = min(10,nSP(i)); % correr so 10 vezes, 10 primeiros
        for k = 1:n
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            if load < best
                k_best = k;
                best = load;
            end
        end
        sol(i) = k_best;
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
hold on
grid on
plot(sort(allValues));
fprintf('Worst link load value of the best solution = %.2f Gbps\n',
    bestLoad);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
    allValues));


fprintf('\nSolution greedy randomized using 5 shortest routing paths\n');
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    ax2 = randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
        k_best = 0;
        best = inf;
        n = min(5,nSP(i)); %correr so 5 vezes, 5 primeiros
        for k = 1:n
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            if load < best
                k_best = k;
                best = load;
            end
        end
```

```
            sol(i) = k_best;
        end
        Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
        load= max(max(Loads(:,3:4)));
        allValues= [allValues load];
        if load<bestLoad
            bestSol= sol;
            bestLoad= load;
        end
end
hold on
grid on
plot(sort(allValues));
legend('Greedy randomized using all possible','Greedy randomized using 10
    shortest','Greedy randomized using 5 shortest',Location="southeast");
fprintf('Worst link load value of the best solution = %.2f Gbps\n',
    bestLoad);
fprintf('Number of solutions generated = %d \n', length(allValues));
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 custumam 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 ramdomized 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 ramdomized 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 númericos 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 proximas 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.

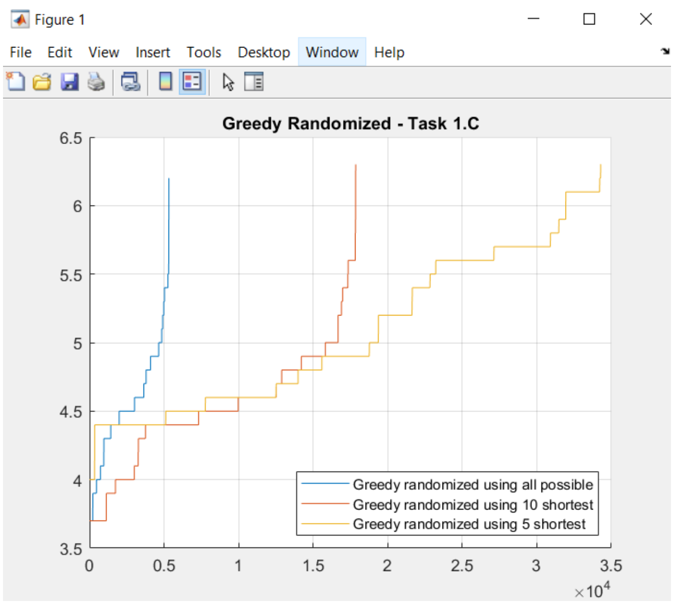

Figura 1.5: Resultados gráficos para a alínea C nos diferentes casos.

A diferença computancionalmente 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.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 tecnica 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:

```
clear all;
close all;

fprintf("Task 1 - Alinea D\n");
Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
        540 290
        120 240
        400 310
        220 370
        550 380];

Links= [1 2
```

```
           1  5
           2  3
           2  4
           3  4
           3  6
           3  8
           4  5
           4  8
           5  7
           6  8
           6  10
           7  8
           7  9
           8  9
           9  10];

T= [1   3   1.0  1.0
    1   4   0.7  0.5
    2   7   2.4  1.5
    3   4   2.4  2.1
    4   9   1.0  2.2
    5   6   1.2  1.5
    5   8   2.1  2.5
    5   9   1.6  1.9
    6  10   1.4  1.6];

nNodes= 10;

nLinks= size(Links,1);
nFlows= size(T,1);

co= Nodes(:,1)+j*Nodes(:,2);
L= inf(nNodes);    %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
for i=1:nLinks
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km

% Compute up to n paths for each flow:
n= inf;
[sP nSP]= calculatePaths(L,T,n);

tempo= 10;

fprintf('\nSolution hill climbing using all possible routing paths\n');
%Optimization algorithm with multi start hill climbing:
t= tic;
bestLoad= inf;
allValues= [];
contadortotal= [];
```

```
while toc(t)<tempo

    %GREEDY RANDOMIZED:
    %construir uma solucao
    ax2= randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
        k_best= 0;
        best= inf;
        for k= 1:nSP(i)
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            if load<best
                k_best= k;
                best= load;
            end
        end
        sol(i)= k_best;
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= best;

    %HILL CLIMBING:
    %pegar na solucao do greedy e escolher a solucao com o hill climbing
    continuar= true;
    while continuar
        i_best= 0;
        k_best= 0;
        best= load;
        for i= 1:nFlows %cada fluxo
            for k= 1:nSP(i) %cada percurso -> vamos a cada fluxo e a cada percurso do fluxo
                if k~=sol(i) %se o percurso for diferente do atualmente escolhido
                    aux= sol(i);
                    sol(i)= k;
                    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol); %calculo cargas
                    load1= max(max(Loads(:,3:4))); %calculo carga maxima
                    if load1<best
                        i_best= i;
                        k_best= k;
                        best= load1;
                    end
                    sol(i)= aux; %repor o fluxo original
                end
            end
        end
        %quando nenhuma das solucoes for melhor, ele para -> i_best=0 (nao
        %houve troca dentro dos fores)
        if i_best>0
            sol(i_best)= k_best;
            load= best;
        else
```

```
                continuar= false;
            end
        end
        allValues= [allValues load];
        if load<bestLoad
            bestSol= sol;
            bestLoad= load;
        end
end
figure(1);
grid on
plot(sort(allValues));
title('Multi Start Hill Climbing - Task 1.D');

fprintf('   Best load = %.2f Gbps\n',bestLoad);
fprintf('   No. of solutions = %d\n',length(allValues));
fprintf('   Av. quality of solutions = %.2f Gbps\n',mean(allValues));


fprintf('\nSolution hill climbing using 10 shortest routing paths\n');
t= tic;
bestLoad= inf;
allValues= [];
contadortotal= [];
while toc(t)<tempo

    %GREEDY RANDOMIZED:
    ax2= randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
        k_best= 0;
        best= inf;
        n = min(10,nSP(i));
        for k= 1:n
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            if load<best
                k_best= k;
                best= load;
            end
        end
        sol(i)= k_best;
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= best;

    %HILL CLIMBING:
    continuar= true;
    while continuar
        i_best= 0;
        k_best= 0;
        best= load;
        for i= 1:nFlows
            for k= 1:nSP(i)
```

```
                if k~=sol(i)
                    aux= sol(i);
                    sol(i)= k;
                    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                    load1= max(max(Loads(:,3:4)));
                    if load1<best
                        i_best= i;
                        k_best= k;
                        best= load1;
                    end
                    sol(i)= aux;
                end
            end
        end
        if i_best>0
            sol(i_best)= k_best;
            load= best;
        else
            continuar= false;
        end
    end
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
hold on
grid on
plot(sort(allValues));

fprintf('   Best load = %.2f Gbps\n',bestLoad);
fprintf('   No. of solutions = %d\n',length(allValues));
fprintf('   Av. quality of solutions = %.2f Gbps\n',mean(allValues));


fprintf('\nSolution hill climbing using 5 shortest routing paths\n');
t= tic;
bestLoad= inf;
allValues= [];
contadortotal= [];
while toc(t)<tempo

    %GREEDY RANDOMIZED:
    ax2= randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
        k_best= 0;
        best= inf;
        n = min(5,nSP(i));
        for k= 1:n
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            if load<best
```

```
                k_best= k;
                best= load;
            end
        end
        sol(i)= k_best;
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= best;

    %HILL CLIMBING:
    continuar= true;
    while continuar
        i_best= 0;
        k_best= 0;
        best= load;
        for i= 1:nFlows
            for k= 1:nSP(i)
                if k~=sol(i)
                    aux= sol(i);
                    sol(i)= k;
                    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                    load1= max(max(Loads(:,3:4)));
                    if load1<best
                        i_best= i;
                        k_best= k;
                        best= load1;
                    end
                    sol(i)= aux;
                end
            end
        end

        if i_best>0
            sol(i_best)= k_best;
            load= best;
        else
            continuar= false;
        end
    end
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
hold on
grid on
plot(sort(allValues));
legend('Hill climbing using all possible','Hill climbing using 10 shortest'
    ,'Hill climbing using 5 shortest',Location="southeast");
fprintf('   Best load = %.2f Gbps\n',bestLoad);
fprintf('   No. of solutions = %d\n',length(allValues));
fprintf('   Av. quality of solutions = %.2f Gbps\n',mean(allValues));
```

1.d.2 Resultados e Conclusões

Com este codigo, 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 custumam 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 númericos 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 analizar, mas esta diferença não é tão significativa como nos algoritmos random e greedy randomized.

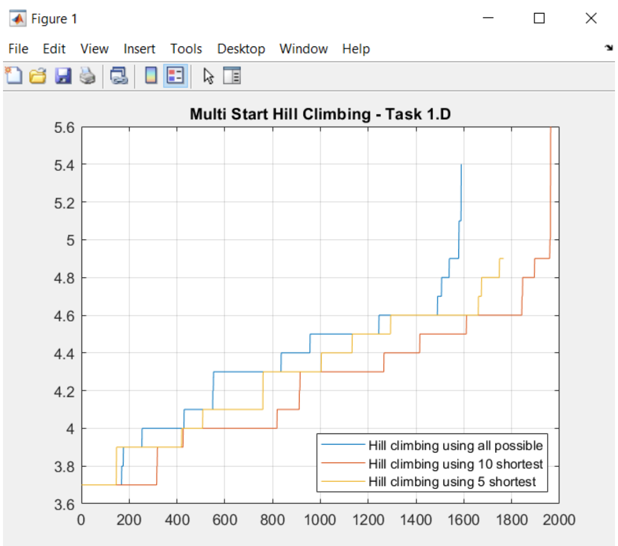

Figura 1.7: Resultados gráficos para a alínea D nos diferentes casos.

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:

```
clear all;
close all;

fprintf('Task 1 - Alinea E\n');

Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
        540 290
        120 240
        400 310
        220 370
        550 380];

Links= [1 2
         1 5
         2 3
         2 4
         3 4
         3 6
         3 8
         4 5
         4 8
         5 7
         6 8
         6 10
         7 8
         7 9
         8 9
         9 10];

T= [1   3   1.0  1.0
    1   4   0.7  0.5
    2   7   2.4  1.5
    3   4   2.4  2.1
    4   9   1.0  2.2
    5   6   1.2  1.5
    5   8   2.1  2.5
    5   9   1.6  1.9
    6  10   1.4  1.6];

nNodes= 10;
```

```
nLinks= size(Links,1);
nFlows= size(T,1);

B= 625;  %Average packet size in Bytes

co= Nodes(:,1)+j*Nodes(:,2); %calculo para a distancia de cada no

L= inf(nNodes);    %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
C= zeros(nNodes);  %Square matrix with arc capacities (in Gbps)
for i=1:nLinks
    C(Links(i,1),Links(i,2))= 10;  %Gbps
    C(Links(i,2),Links(i,1))= 10;  %Gbps
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km

% Compute up to 100 paths for each flow:
n= 100;
[sP nSP]= calculatePaths(L,T,n);

%Optimization algorithm resorting to the random strategy:
fprintf('\nSolution random with all possible routing paths\n');
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    for i= 1:nFlows
        sol(i)= randi(nSP(i));
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
figure(1)
grid on
plot(sort(allValues));
title('Compare the algorithms - Task 1.E')

fprintf('Worst link load value of the best solution = %.2f Gbps\n', bestLoad);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(allValues));

```

```
fprintf('\nSolution greedy ramdomized using all possible routing paths\n');
%Optimization algorithm resorting to the greedy randomized strategy:
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    ax2 = randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
        k_best = 0;
        best = inf;
        for k = 1:nSP(i)
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            if load < best
                k_best = k;
                best = load;
            end
        end
        sol(i) = k_best;
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end

hold on
grid on
plot(sort(allValues));
fprintf('Worst link load value of the best solution = %.2f Gbps\n',
    bestLoad);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
    allValues));

tempo= 10;
fprintf('\nSolution hill climbing using all possible routing paths\n');
%Optimization algorithm with multi start hill climbing:
t= tic;
bestLoad= inf;
allValues= [];
contadortotal= [];
while toc(t)<tempo

    %GREEDY RANDOMIZED:
    ax2= randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
```

```
        k_best= 0;
        best= inf;
        for k= 1:nSP(i)
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            if load<best
                k_best= k;
                best= load;
            end
        end
        sol(i)= k_best;
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= best;

    %HILL CLIMBING:
    continuar= true;
    while continuar
        i_best= 0;
        k_best= 0;
        best= load;
        for i= 1:nFlows
            for k= 1:nSP(i)
                if k~=sol(i)
                    aux= sol(i);
                    sol(i)= k;
                    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                    load1= max(max(Loads(:,3:4)));
                    if load1<best
                        i_best= i;
                        k_best= k;
                        best= load1;
                    end
                    sol(i)= aux;
                end
            end
        end
        if i_best>0
            sol(i_best)= k_best;
            load= best;
        else
            continuar= false;
        end
    end
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
hold on
grid on
plot(sort(allValues));
legend('Random using all possible','Greedy randomized using all possible','
```

```
    Hill climbing using all possible',Location="southeast");

fprintf('   Best load = %.2f Gbps\n',bestLoad);
fprintf('   No. of solutions = %d\n',length(allValues));
fprintf('   Av. quality of solutions = %.2f Gbps\n',mean(allValues));


fprintf('\nSolution random with 10 shortest routing paths\n');
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10

    for i= 1:nFlows
        n = min(10,nSP(i));
        sol(i)= randi(n);
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
figure(2);
grid on

plot(sort(allValues));
title('Compare the algorithms - Task 1.E')
fprintf('Worst link load value of the best solution = %.2f Gbps\n', bestLoad);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(allValues));

fprintf('\nSolution greedy ramdomized using 10 shortest routing paths\n');
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    ax2 = randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
        k_best = 0;
        best = inf;
        n = min(10,nSP(i));
        for k = 1:n
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            if load < best
                k_best = k;
```

```
                    best = load;
                end
            end
            sol(i) = k_best;
        end
        Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
        load= max(max(Loads(:,3:4)));
        allValues= [allValues load];
        if load<bestLoad
            bestSol= sol;
            bestLoad= load;
        end
    end
    hold on
    grid on
    plot(sort(allValues));
    fprintf('Worst link load value of the best solution = %.2f Gbps\n',
        bestLoad);
    fprintf('Number of solutions generated = %d \n', length(allValues));
    fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
        allValues));

    fprintf('\nSolution hill climbing using 10 shortest routing paths\n');
    t= tic;
    bestLoad= inf;
    allValues= [];
    contadortotal= [];
    while toc(t)<tempo

        %GREEDY RANDOMIZED:
        ax2= randperm(nFlows);
        sol= zeros(1,nFlows);
        for i= ax2
            k_best= 0;
            best= inf;
            n = min(10,nSP(i));
            for k= 1:n
                sol(i)= k;
                Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                load= max(max(Loads(:,3:4)));
                if load<best
                    k_best= k;
                    best= load;
                end
            end
            sol(i)= k_best;
        end
        Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
        load= best;

        %HILL CLIMBING:
        continuar= true;
        while continuar
            i_best= 0;
            k_best= 0;
```

```
        best= load;
        for i= 1:nFlows
            for k= 1:nSP(i)
                if k~=sol(i)
                    aux= sol(i);
                    sol(i)= k;
                    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                    load1= max(max(Loads(:,3:4)));
                    if load1<best
                        i_best= i;
                        k_best= k;
                        best= load1;
                    end
                    sol(i)= aux;
                end
            end
        end

        if i_best>0
            sol(i_best)= k_best;
            load= best;
        else
            continuar= false;
        end
    end
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
hold on
grid on
plot(sort(allValues));
legend('Random with 10 shortest','Greedy randomized with 10 shortest','Hill climbing with 10 shortest',Location="southeast");

fprintf('   Best load = %.2f Gbps\n',bestLoad);
fprintf('   No. of solutions = %d\n',length(allValues));
fprintf('   Av. quality of solutions = %.2f Gbps\n',mean(allValues));


fprintf('\nSolution random with 5 shortest routing paths\n');
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    for i= 1:nFlows
        n = min(5,nSP(i));
        sol(i)= randi(n);
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    allValues= [allValues load];
```

```
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
figure(3)
grid on
plot(sort(allValues));
title('Compare the algorithms - Task 1.E')

fprintf('Worst link load value of the best solution = %.2f Gbps\n',
    bestLoad);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
    allValues));

fprintf('\nSolution greedy randomized using 5 shortest routing paths\n');
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    ax2 = randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
        k_best = 0;
        best = inf;
        n = min(5,nSP(i));
        for k = 1:n
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            if load < best
                k_best = k;
                best = load;
            end
        end
        sol(i) = k_best;
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
hold on
grid on
plot(sort(allValues));
fprintf('Worst link load value of the best solution = %.2f Gbps\n',
    bestLoad);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.2f Gbps\n', mean(
    allValues));
```

```
fprintf('\nSolution hill climbing using 5 shortest routing paths\n');
t= tic;
bestLoad= inf;
allValues= [];
contadortotal= [];
while toc(t)<tempo

    %GREEDY RANDOMIZED:
    ax2= randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
        k_best= 0;
        best= inf;
        n = min(5,nSP(i));
        for k= 1:n
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            if load<best
                k_best= k;
                best= load;
            end
        end
        sol(i)= k_best;
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= best;

    %HILL CLIMBING:
    continuar= true;
    while continuar
        i_best= 0;
        k_best= 0;
        best= load;
        for i= 1:nFlows
            for k= 1:nSP(i)
                if k~=sol(i)
                    aux= sol(i);
                    sol(i)= k;
                    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                    load1= max(max(Loads(:,3:4)));
                    if load1<best
                        i_best= i;
                        k_best= k;
                        best= load1;
                    end
                    sol(i)= aux;
                end
            end
        end
        if i_best>0
            sol(i_best)= k_best;
            load= best;
```

```
        else
            continuar= false;
        end
    end
    allValues= [allValues load];
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
hold on
grid on
plot(sort(allValues));
legend('Random with 5 shortest','Greedy randomized with 5 shortest','Hill
    climbing with 5 shortest',Location="southeast");
fprintf('   Best load = %.2f Gbps\n',bestLoad);
fprintf('   No. of solutions = %d\n',length(allValues));
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 rapido e mais eficiente que o algoritmo greedy randomized e random, encontrando mais rápidamente 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 enquando 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 ramdomized 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 all possible routing paths
   Best load = 3.70 Gbps
   No. of solutions = 1623
   Av. quality of solutions = 4.30 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 ramdomized 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 ramdomized 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 10 shortest routing paths
   Best load = 3.70 Gbps
   No. of solutions = 2113
   Av. quality of solutions = 4.23 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 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 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 númericos 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 Gpbs e 3.70 Gpbs para os algorirmos random, greedy e hill climbing, respetivamente, quando se utiliza todas as rotas possíveis. Aqui pode-se observar que a solução random obtem valores piores que os algortimos 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.

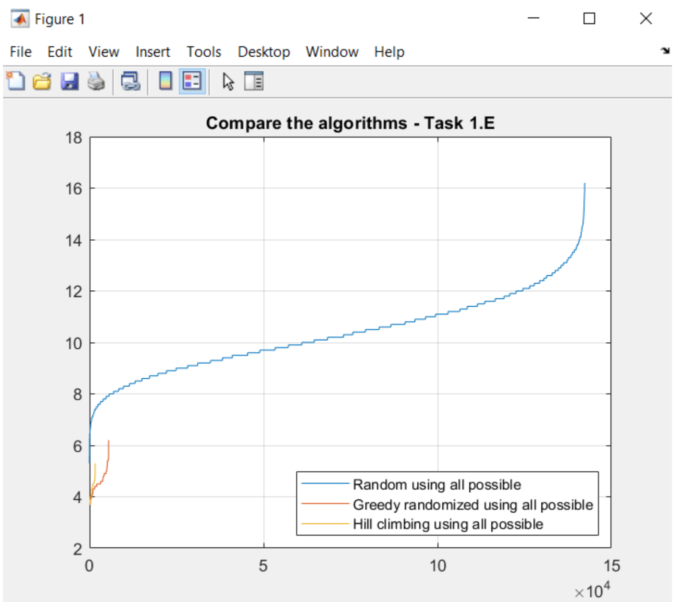

Figura 1.9: Resultados gráficos para a alínea E no caso de ter todas as rotas para os diferentes algoritmos.

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.

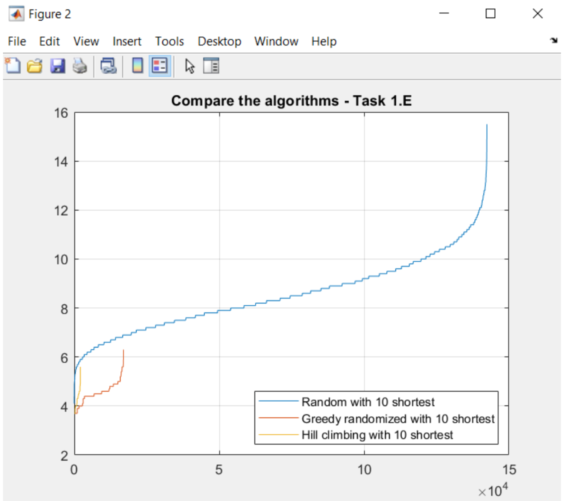

Figura 1.10: Resultados gráficos para a alínea E no caso de ter 10 rotas mais curtas para os diferentes algoritmos.

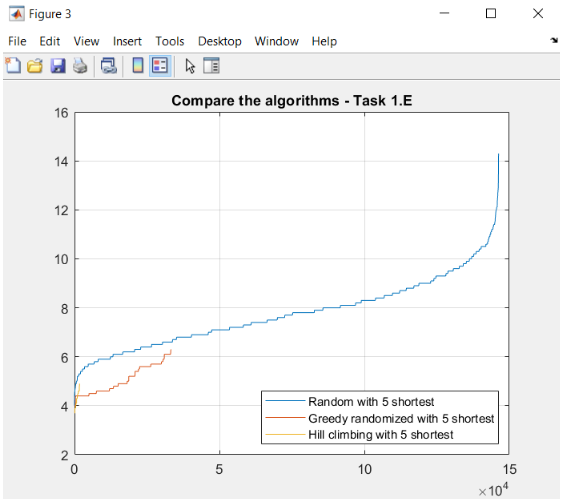

Figura 1.11: Resultados gráficos para a alínea E no caso de ter 5 rotas mais curtas para os diferentes algoritmos.

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 atribuido anteriomente 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:

```
clear all;
close all;

fprintf('Task 2 - Alinea A\n');
Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
        540 290
        120 240
        400 310
        220 370
        550 380];

Links= [1 2
        1 5
        2 3
        2 4
        3 4
        3 6
        3 8
        4 5
        4 8
        5 7
        6 8
        6 10
        7 8
        7 9
        8 9
        9 10];

T= [1  3  1.0  1.0
    1  4  0.7  0.5
    2  7  2.4  1.5
    3  4  2.4  2.1
    4  9  1.0  2.2
    5  6  1.2  1.5
    5  8  2.1  2.5
    5  9  1.6  1.9
```

```
    6 10  1.4  1.6];

nNodes= 10;

nLinks= size(Links,1);
nFlows= size(T,1);

B= 625;  %Average packet size in Bytes

co= Nodes(:,1)+j*Nodes(:,2);

L= inf(nNodes);     %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
C= zeros(nNodes);  %Square matrix with arc capacities (in Gbps)
for i=1:nLinks
    C(Links(i,1),Links(i,2))= 10;  %Gbps
    C(Links(i,2),Links(i,1))= 10;  %Gbps
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km

% Compute up to 100 paths for each flow:
n= 100;
[sP nSP]= calculatePaths(L,T,n);

fprintf('\nSolution random with all possible routing paths\n');
t= tic;
bestEnergy= inf;
sol= zeros(1,nFlows);
allValues= [];
%quanto mais comprimento do link mais energia, logo queremos o que tem
    menos comprimento, menos energia.
while toc(t)<10
    for i= 1:nFlows
        sol(i)= randi(nSP(i));
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    if load<=10
        energy = 0;
        for a=1:nLinks
            if Loads(a,3)+Loads(a,4)>0 %se nao estiver em sleep mode
                energy = energy + L(Loads(a,1),Loads(a,2)); %a energia e
                    igual a energia + o comprimento do link
            end
        end
    else
        energy=inf; %garante que nao escolhe se a capacidade for maior que
            10 gigabits
    end
    %escolhemos o caminho que gasta menos energia
```

```
    allValues= [allValues energy];
    if energy<bestEnergy
        bestSol= sol;
        bestEnergy= energy;
    end
end
figure(1)
grid on
plot(sort(allValues));
title('Random Algorithm - Task 2.A')
fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergy);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.1f \n', mean(
    allValues));


fprintf('\nSolution random with 10 shortest routing paths\n');
t= tic;
bestEnergy= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    for i= 1:nFlows
        n = min(10,nSP(i)); %correr so 10 vezes, 10 primeiros
        sol(i)= randi(n);
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    if load<=10
        energy = 0;
        for a=1:nLinks
            if Loads(a,3)+Loads(a,4)>0
                energy = energy + L(Loads(a,1),Loads(a,2));
            end
        end
    else
        energy=inf;
    end
    allValues= [allValues energy];
    if energy<bestEnergy
        bestSol= sol;
        bestEnergy= energy;
    end
end

hold on
grid on
plot(sort(allValues));
fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergy);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.1f \n', mean(
    allValues));

```

```
fprintf('\nSolution random with 5 shortest routing paths\n');
t= tic;
bestEnergy= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    for i= 1:nFlows
        n = min(5,nSP(i)); %correr so 5 vezes, 5 primeiros
        sol(i)= randi(n);
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    if load<=10
        energy = 0;
        for a=1:nLinks
            if Loads(a,3)+Loads(a,4)>0
                energy = energy + L(Loads(a,1),Loads(a,2));
            end
        end
    else
        energy=inf;
    end
    allValues= [allValues energy];
    if energy<bestEnergy
        bestSol= sol;
        bestEnergy= energy;
    end

end

hold on
grid on
plot(sort(allValues));
legend('Random with all possible','Random with 10 shortest','Random with 5
    shortest',Location="southeast");
fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergy);
fprintf('Number of solutions generated = %d \n', length(allValues));
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 custumam 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 - Alinea 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 númericos 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.

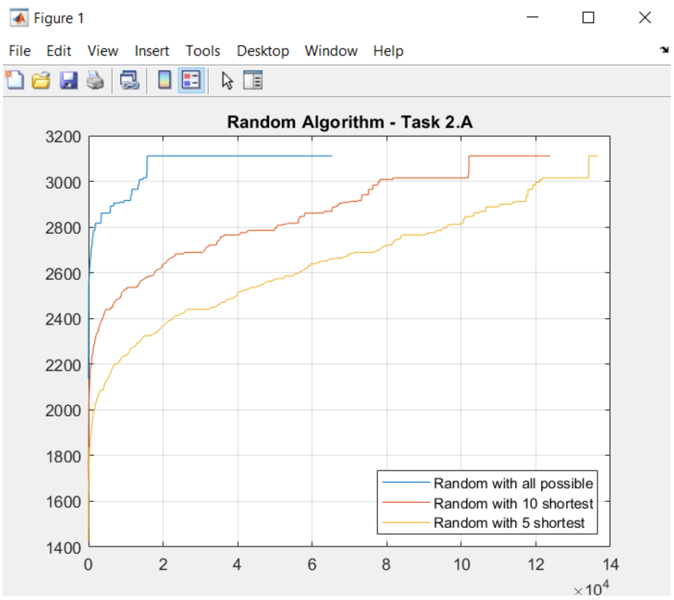

Figura 2.2: Resultados gráficos para a alínea A nos diferentes casos.

A diferença computancionalmente 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 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)*). 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 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 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:

```
clear all;
close all;

fprintf('Task 2 - Alinea B\n');
Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
        540 290
        120 240
        400 310
        220 370
        550 380];

Links= [1 2
        1 5
        2 3
        2 4
        3 4
        3 6
        3 8
        4 5
        4 8
        5 7
        6 8
        6 10
        7 8
        7 9
        8 9
        9 10];

T= [1  3  1.0 1.0
    1  4  0.7 0.5
    2  7  2.4 1.5
    3  4  2.4 2.1
    4  9  1.0 2.2
    5  6  1.2 1.5
    5  8  2.1 2.5
    5  9  1.6 1.9
    6 10  1.4 1.6];

nNodes= 10;

nLinks= size(Links,1);
nFlows= size(T,1);

B= 625;  %Average packet size in Bytes

co= Nodes(:,1)+j*Nodes(:,2);

L= inf(nNodes);    %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
```

```
end
C= zeros(nNodes);  %Square matrix with arc capacities (in Gbps)
for i=1:nLinks
    C(Links(i,1),Links(i,2))= 10;  %Gbps
    C(Links(i,2),Links(i,1))= 10;  %Gbps
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km

% Compute up to 100 paths for each flow:
n= 100;
[sP nSP]= calculatePaths(L,T,n);

fprintf('\nSolution greedy ramdomized using all possible routing paths\n');

t= tic;
bestEnergyGreedy= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    ax2 = randperm(nFlows); %array numa ordem aleatoria
    sol= zeros(1,nFlows);
    for i= ax2
        k_best = 0;
        best = inf;
        %encontrar o melhor vizinho
        for k = 1:nSP(i)
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));

            if load<=10
                energy = 0;
                for a=1:nLinks
                    if Loads(a,3)+Loads(a,4)>0
                        energy = energy + L(Loads(a,1),Loads(a,2));
                    end
                end
            else
                energy=inf;
            end

            if energy < best
                k_best = k;
                best = energy;
            end
        end
        sol(i) = k_best; % percurso com a melhor escolha
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    if load<=10
        energy = 0;
```

```
        for a=1:nLinks
            if Loads(a,3)+Loads(a,4)>0
                energy = energy + L(Loads(a,1),Loads(a,2));
            end
        end
    else
        energy=inf;
    end

    allValues= [allValues energy];
    if energy <bestEnergyGreedy
        bestSol= sol;
        bestEnergyGreedy= energy;
    end
end
hold on
grid on
plot(sort(allValues));
title('Greedy Randomized - Task 2.B')
fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergyGreedy);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.1f \n', mean(
    allValues));


fprintf('\nSolution greedy ramdomized using 10 shortest routing paths\n');
t= tic;
bestEnergyGreedy= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    ax2 = randperm(nFlows); % array numa ordem aleatoria
    sol= zeros(1,nFlows);
    for i= ax2
        k_best = 0;
        best = inf;
        %encontrar o melhor vizinho
        n = min(10,nSP(i)); %correr so 10 vezes, 10 primeiros
        for k = 1:n
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));

            if load<=10
                energy = 0;
                for a=1:nLinks
                    if Loads(a,3)+Loads(a,4)>0
                        energy = energy + L(Loads(a,1),Loads(a,2));
                    end
                end
            else
                energy=inf;
            end

```

```
                if energy < best
                    k_best = k;
                    best = energy;
                end
            end
            sol(i) = k_best; % percurso com a melhor escolha
        end
        Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
        load= max(max(Loads(:,3:4)));
        if load<=10
            energy = 0;
            for a=1:nLinks
                if Loads(a,3)+Loads(a,4)>0
                    energy = energy + L(Loads(a,1),Loads(a,2));
                end
            end
        else
            energy=inf;
        end

        allValues= [allValues energy];
        if energy <bestEnergyGreedy
            bestSol= sol;
            bestEnergyGreedy= energy;
        end
    end

    hold on
    grid on
    plot(sort(allValues));
    fprintf('Energy consumption value of the best solution = %.1f \n',
        bestEnergyGreedy);
    fprintf('Number of solutions generated = %d \n', length(allValues));
    fprintf('Average quality of all solutions generated = %.1f \n', mean(
        allValues));


    fprintf('\nSolution greedy ramdomized using 5 shortest routing paths\n');
    t= tic;
    bestEnergyGreedy= inf;
    sol= zeros(1,nFlows);
    allValues= [];
    while toc(t)<10
        ax2 = randperm(nFlows); %array numa ordem aleatoria
        sol= zeros(1,nFlows);
        for i= ax2
            k_best = 0;
            best = inf;
            %encontrar o melhor vizinho
             n = min(5,nSP(i)); %correr so 5 vezes, 5 primeiros
            for k = 1:n
                sol(i)= k;
                Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                load= max(max(Loads(:,3:4)));

```

```
                if load<=10
                    energy = 0;
                    for a=1:nLinks
                        if Loads(a,3)+Loads(a,4)>0
                            energy = energy + L(Loads(a,1),Loads(a,2));
                        end
                    end
                else
                    energy=inf;
                end

                if energy < best
                    k_best = k;
                    best = energy;
                end
            end
            sol(i) = k_best; % percurso com a melhor escolha
        end
        Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
        load= max(max(Loads(:,3:4)));
        if load<=10
            energy = 0;
            for a=1:nLinks
                if Loads(a,3)+Loads(a,4)>0
                    energy = energy + L(Loads(a,1),Loads(a,2));
                end
            end
        else
            energy=inf; s
        end

        allValues= [allValues energy];
        if energy <bestEnergyGreedy
            bestSol= sol;
            bestEnergyGreedy= energy;
        end
end

hold on
grid on
plot(sort(allValues));
legend('Greedy randomized using all possible','Greedy randomized using 10
    shortest','Greedy randomized using 5 shortest',Location="southeast");
fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergyGreedy);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.1f \n', mean(
    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 custumam 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) obtive-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 ramdomized 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 ramdomized 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 ramdomized 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 númericos 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 proximas 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.

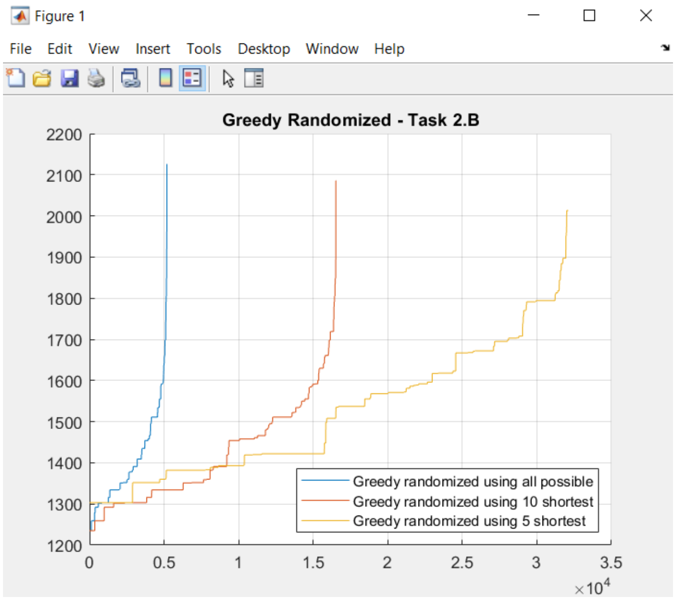

Figura 2.4: Resultados gráficos para a alínea B nos diferentes casos.

A diferença computancionalmente 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. 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 tecnica 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:

```
clear all;
close all;

fprintf('Task 2 - Alinea C\n');
Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
        540 290
        120 240
        400 310
        220 370
        550 380];

```

```
Links= [1  2
        1  5
        2  3
        2  4
        3  4
        3  6
        3  8
        4  5
        4  8
        5  7
        6  8
        6  10
        7  8
        7  9
        8  9
        9  10];

T= [1   3   1.0  1.0
    1   4   0.7  0.5
    2   7   2.4  1.5
    3   4   2.4  2.1
    4   9   1.0  2.2
    5   6   1.2  1.5
    5   8   2.1  2.5
    5   9   1.6  1.9
    6  10   1.4  1.6];

nNodes= 10;
nLinks= size(Links,1);
nFlows= size(T,1);

co= Nodes(:,1)+j*Nodes(:,2);
L= inf(nNodes);    %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
for i=1:nLinks
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km

% Compute up to n paths for each flow:
n= inf;
[sP nSP]= calculatePaths(L,T,n);

tempo= 10;

fprintf('\nSolution hill climbing using all possible routing paths\n');
%Optimization algorithm with multi start hill climbing:
t= tic;
bestEnergy= inf;
allValues= [];
contadortotal= [];
```

```
while toc(t)<tempo
    %GREEDY RANDOMIZED:
    continuar= true;
    while continuar
        continuar= false;
        ax2= randperm(nFlows);
        sol= zeros(1,nFlows);
        for i= ax2
            k_best= 0;
            best= inf;
            for k= 1:nSP(i)
                sol(i)= k;
                Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                load= max(max(Loads(:,3:4)));
                if load <= 10
                    energy= 0;
                    for a= 1:nLinks
                        if Loads(a,3)+Loads(a,4)>0
                            energy= energy + L(Loads(a,1),Loads(a,2));
                        end
                    end
                else
                    energy= inf;
                end
                if energy<best
                    k_best= k;
                    best= energy;
                end
            end
            if k_best>0
                sol(i)= k_best;
            else
                continuar= true;
                break;
            end
        end
    end
    energy= best;

    %HILL CLIMBING:
    continuar= true;
    while continuar
        i_best= 0;
        k_best= 0;
        best= energy;
        for i= 1:nFlows
            for k= 1:nSP(i)
                if k~=sol(i)
                    aux= sol(i);
                    sol(i)= k;
                    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                    load1= max(max(Loads(:,3:4)));
                    if load1 <= 10
                        energy1= 0;
                        for a= 1:nLinks
```

```
                                if Loads(a,3)+Loads(a,4)>0
                                    energy1= energy1 + L(Loads(a,1),Loads(a,2))
                                        ;
                                end
                            end
                        else
                            energy1= inf;
                        end
                        if energy1<best
                            i_best= i;
                            k_best= k;
                            best= energy1;
                        end
                        sol(i)= aux;
                    end
                end
            end
            if i_best>0
                sol(i_best)= k_best;
                energy= best;
            else
                continuar= false;
            end
        end
        allValues= [allValues energy];
        if energy<bestEnergy
            bestSol= sol;
            bestEnergy= energy;
        end
end
figure(1);
grid on
plot(sort(allValues));
title('Multi Start Hill Climbing - Task 2.C');
fprintf('   Best energy = %.1f\n',bestEnergy);
fprintf('   No. of solutions = %d\n',length(allValues));
fprintf('   Av. quality of solutions = %.1f\n',mean(allValues));

% o hill climbing nao e muito vantajoso para calcular a energia,
    normalmente ele nao
% consegue melhorar a solucao inicial so greeedy randomazed

fprintf('\nSolution hill climbing using 10 shortest routing paths\n');
%Optimization algorithm with multi start hill climbing:
t= tic;
bestEnergy= inf;
allValues= [];
contadortotal= [];
while toc(t)<tempo
    %GREEDY RANDOMIZED:
    continuar= true;
    while continuar
        continuar= false;
        ax2= randperm(nFlows);
        sol= zeros(1,nFlows);
```

```
        for i= ax2
            k_best= 0;
            best= inf;
             n = min(10,nSP(i));
            for k= 1:n
                sol(i)= k;
                Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                load= max(max(Loads(:,3:4)));
                if load <= 10
                    energy= 0;
                    for a= 1:nLinks
                        if Loads(a,3)+Loads(a,4)>0
                            energy= energy + L(Loads(a,1),Loads(a,2));
                        end
                    end
                else
                    energy= inf;
                end
                if energy<best
                    k_best= k;
                    best= energy;
                end
            end
            if k_best>0
                sol(i)= k_best;
            else
                continuar= true;
                break;
            end
        end
    end
    energy= best;

    %HILL CLIMBING:
    continuar= true;
    while continuar
        i_best= 0;
        k_best= 0;
        best= energy;
        for i= 1:nFlows
            for k= 1:nSP(i)
                if k~=sol(i)
                    aux= sol(i);
                    sol(i)= k;
                    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                    load1= max(max(Loads(:,3:4)));
                    if load1 <= 10
                        energy1= 0;
                        for a= 1:nLinks
                            if Loads(a,3)+Loads(a,4)>0
                                energy1= energy1 + L(Loads(a,1),Loads(a,2))
                                    ;
                            end
                        end
                    else
```

```
                        energy1= inf;
                    end
                    if energy1<best
                        i_best= i;
                        k_best= k;
                        best= energy1;
                    end
                    sol(i)= aux;
                end
            end
        end
        if i_best>0
            sol(i_best)= k_best;
            energy= best;
        else
            continuar= false;
        end
    end
    allValues= [allValues energy];
    if energy<bestEnergy
        bestSol= sol;
        bestEnergy= energy;
    end
end
hold on
grid on
plot(sort(allValues));
fprintf('   Best energy = %.1f\n',bestEnergy);
fprintf('   No. of solutions = %d\n',length(allValues));
fprintf('   Av. quality of solutions = %.1f\n',mean(allValues));


fprintf('\nSolution hill climbing using 5 shortest routing paths\n');
%Optimization algorithm with multi start hill climbing:
t= tic;
bestEnergy= inf;
allValues= [];
contadortotal= [];
while toc(t)<tempo
    %GREEDY RANDOMIZED:
    continuar= true;
    while continuar
        continuar= false;
        ax2= randperm(nFlows);
        sol= zeros(1,nFlows);
        for i= ax2
            k_best= 0;
            best= inf;
             n = min(5,nSP(i));
            for k= 1:n
                sol(i)= k;
                Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                load= max(max(Loads(:,3:4)));
                if load <= 10
                    energy= 0;
```

```
                    for a= 1:nLinks
                        if Loads(a,3)+Loads(a,4)>0
                            energy= energy + L(Loads(a,1),Loads(a,2));
                        end
                    end
                else
                    energy= inf;
                end
                if energy<best
                    k_best= k;
                    best= energy;
                end
            end
            if k_best>0
                sol(i)= k_best;
            else
                continuar= true;
                break;
            end
        end
    end
    energy= best;

    %HILL CLIMBING:
    continuar= true;
    while continuar
        i_best= 0;
        k_best= 0;
        best= energy;
        for i= 1:nFlows
            for k= 1:nSP(i)
                if k~=sol(i)
                    aux= sol(i);
                    sol(i)= k;
                    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                    load1= max(max(Loads(:,3:4)));
                    if load1 <= 10
                        energy1= 0;
                        for a= 1:nLinks
                            if Loads(a,3)+Loads(a,4)>0
                                energy1= energy1 + L(Loads(a,1),Loads(a,2))
                                    ;
                            end
                        end
                    else
                        energy1= inf;
                    end
                    if energy1<best
                        i_best= i;
                        k_best= k;
                        best= energy1;
                    end
                    sol(i)= aux;
                end
            end
```

```
            end
            if i_best>0
                sol(i_best)= k_best;
                energy= best;
            else
                continuar= false;
            end
        end
        allValues= [allValues energy];
        if energy<bestEnergy
            bestSol= sol;
            bestEnergy= energy;
        end
    end
    hold on
    grid on
    plot(sort(allValues));
    legend('Hill climbing using all possible','Hill climbing using 10 shortest'
        ,'Hill climbing using 5 shortest',Location="southeast");
    fprintf('   Best energy = %.1f\n',bestEnergy);
    fprintf('   No. of solutions = %d\n',length(allValues));
    fprintf('   Av. quality of solutions = %.1f\n',mean(allValues));
```

2.c.2 Resultados e Conclusões

Com este codigo, 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 custumam 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 númericos 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 analizar, mas esta diferença não é tão significativa como nos algoritmos random e greedy randomized.

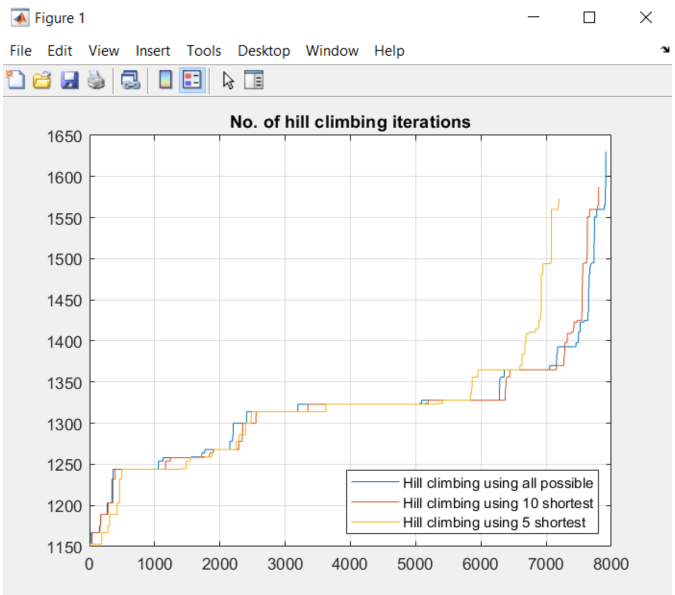

Figura 2.6: Resultados gráficos para a alínea C nos diferentes casos.

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:

```
clear all;
close all;

fprintf('Task 2 - Alinea D\n');
Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
        540 290
        120 240
        400 310
        220 370
        550 380];

Links= [1 2
         1 5
         2 3
         2 4
         3 4
         3 6
         3 8
         4 5
         4 8
         5 7
         6 8
         6 10
         7 8
         7 9
         8 9
         9 10];

T= [1   3   1.0  1.0
    1   4   0.7  0.5
    2   7   2.4  1.5
    3   4   2.4  2.1
    4   9   1.0  2.2
    5   6   1.2  1.5
    5   8   2.1  2.5
    5   9   1.6  1.9
    6  10   1.4  1.6];

nNodes= 10;

```

```
nLinks= size(Links,1);
nFlows= size(T,1);
tempo= 10;

B= 625;  %Average packet size in Bytes

co= Nodes(:,1)+j*Nodes(:,2);

L= inf(nNodes);     %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
C= zeros(nNodes);  %Square matrix with arc capacities (in Gbps)
for i=1:nLinks
    C(Links(i,1),Links(i,2))= 10;  %Gbps
    C(Links(i,2),Links(i,1))= 10;  %Gbps
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km

% Compute up to 100 paths for each flow:
n= 100;
[sP nSP]= calculatePaths(L,T,n);

fprintf('\nSolution random with all possible routing paths\n');
t= tic;
bestEnergy= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    for i= 1:nFlows
        sol(i)= randi(nSP(i));
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    if load<=10
        energy = 0;
        for a=1:nLinks
            if Loads(a,3)+Loads(a,4)>0
                energy = energy + L(Loads(a,1),Loads(a,2));
            end
        end
    else
        energy=inf;
    end

    allValues= [allValues energy];
    if energy<bestEnergy
        bestSol= sol;
        bestEnergy= energy;
    end
end
figure(1);
```

```
grid on
plot(sort(allValues));
title('Compare the algorithms - Task 2.D')
fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergy);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.1f \n', mean(
    allValues));


fprintf('\nSolution greedy ramdomized using all possible routing paths\n');
t= tic;
bestEnergyGreedy= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    ax2 = randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
        k_best = 0;
        best = inf;
        %encontrar o melhor vizinho
        for k = 1:nSP(i)
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));

            if load<=10
                energy = 0;
                for a=1:nLinks
                    if Loads(a,3)+Loads(a,4)>0
                        energy = energy + L(Loads(a,1),Loads(a,2));
                    end
                end
            else
                energy=inf;
            end

            if energy < best
                k_best = k;
                best = energy;
            end
        end
        sol(i) = k_best;
    end

    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    if load<=10
        energy = 0;
        for a=1:nLinks
            if Loads(a,3)+Loads(a,4)>0
                energy = energy + L(Loads(a,1),Loads(a,2));
            end
        end
```

```
    else
        energy=inf;
    end

    allValues= [allValues energy];
    if energy <bestEnergyGreedy
        bestSol= sol;
        bestEnergyGreedy= energy;
    end
end
hold on
grid on
plot(sort(allValues));
fprintf('Energy consumption value of the best solution = %.1f \n',
   bestEnergyGreedy);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.1f \n', mean(
   allValues));

fprintf('\nSolution hill climbing using all possible routing paths\n');
%Optimization algorithm with multi start hill climbing:
t= tic;
bestEnergy= inf;
allValues= [];
contadortotal= [];
while toc(t)<tempo
    %GREEDY RANDOMIZED:
    continuar= true;
    while continuar
        continuar= false;
        ax2= randperm(nFlows);
        sol= zeros(1,nFlows);
        for i= ax2
            k_best= 0;
            best= inf;
            for k= 1:nSP(i)
                sol(i)= k;
                Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                load= max(max(Loads(:,3:4)));
                if load <= 10
                    energy= 0;
                    for a= 1:nLinks
                        if Loads(a,3)+Loads(a,4)>0
                            energy= energy + L(Loads(a,1),Loads(a,2));
                        end
                    end
                else
                    energy= inf;
                end
                if energy<best
                    k_best= k;
                    best= energy;
                end
            end
            if k_best>0
```

```
                    sol(i)= k_best;
                else
                    continuar= true;
                    break;
                end
            end
        end
        energy= best;

        %HILL CLIMBING:
        continuar= true;
        while continuar
            i_best= 0;
            k_best= 0;
            best= energy;
            for i= 1:nFlows
                for k= 1:nSP(i)
                    if k~=sol(i)
                        aux= sol(i);
                        sol(i)= k;
                        Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                        load1= max(max(Loads(:,3:4)));
                        if load1 <= 10
                            energy1= 0;
                            for a= 1:nLinks
                                if Loads(a,3)+Loads(a,4)>0
                                    energy1= energy1 + L(Loads(a,1),Loads(a,2))
                                        ;
                                end
                            end
                        else
                            energy1= inf;
                        end
                        if energy1<best
                            i_best= i;
                            k_best= k;
                            best= energy1;
                        end
                        sol(i)= aux;
                    end
                end
            end
            if i_best>0
                sol(i_best)= k_best;
                energy= best;
            else
                continuar= false;
            end
        end
        allValues= [allValues energy];
        if energy<bestEnergy
            bestSol= sol;
            bestEnergy= energy;
        end
    end
```

```
hold on
grid on
plot(sort(allValues));
legend('Random using all possible','Greedy randomized using all possible','
    Hill climbing using all possible',Location="southeast");
fprintf('   Best energy = %.1f\n',bestEnergy);
fprintf('   No. of solutions = %d\n',length(allValues));
fprintf('   Av. quality of solutions = %.1f\n',mean(allValues));


fprintf('\nSolution random with 10 shortest routing paths\n');
t= tic;
bestEnergy= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    for i= 1:nFlows
        n = min(10,nSP(i));
        sol(i)= randi(n);
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    if load<=10
        energy = 0;
        for a=1:nLinks
            if Loads(a,3)+Loads(a,4)>0
                energy = energy + L(Loads(a,1),Loads(a,2));
            end
        end
    else
        energy=inf;
    end
    allValues= [allValues energy];
    if energy<bestEnergy
        bestSol= sol;
        bestEnergy= energy;
    end
end
figure(2);
grid on
plot(sort(allValues));
title('Compare the algorithms - Task 2.D')
fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergy);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.1f \n', mean(
    allValues));

fprintf('\nSolution greedy ramdomized using 10 shortest routing paths\n');
t= tic;
bestEnergyGreedy= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    ax2 = randperm(nFlows);
```

```
    sol= zeros(1,nFlows);
    for i= ax2
        k_best = 0;
        best = inf;
        %encontrar o melhor vizinho
         n = min(10,nSP(i));
        for k = 1:n
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));

            if load<=10
                energy = 0;
                for a=1:nLinks
                    if Loads(a,3)+Loads(a,4)>0
                        energy = energy + L(Loads(a,1),Loads(a,2));
                    end
                end
            else
                energy=inf;
            end

            if energy < best
                k_best = k;
                best = energy;
            end
        end
        sol(i) = k_best;
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    if load<=10
        energy = 0;
        for a=1:nLinks
            if Loads(a,3)+Loads(a,4)>0
                energy = energy + L(Loads(a,1),Loads(a,2));
            end
        end
    else
        energy=inf;
    end

    allValues= [allValues energy];
    if energy <bestEnergyGreedy
        bestSol= sol;
        bestEnergyGreedy= energy;
    end
end

hold on
grid on
plot(sort(allValues));
fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergyGreedy);
fprintf('Number of solutions generated = %d \n', length(allValues));
```

```
fprintf('Average quality of all solutions generated = %.1f \n', mean(
    allValues));

fprintf('\nSolution hill climbing using 10 shortest routing paths\n');
%Optimization algorithm with multi start hill climbing:
t= tic;
bestEnergy= inf;
allValues= [];
contadortotal= [];
while toc(t)<tempo
    %GREEDY RANDOMIZED:
    continuar= true;
    while continuar
        continuar= false;
        ax2= randperm(nFlows);
        sol= zeros(1,nFlows);
        for i= ax2
            k_best= 0;
            best= inf;
             n = min(10,nSP(i));
            for k= 1:n
                sol(i)= k;
                Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                load= max(max(Loads(:,3:4)));
                if load <= 10
                    energy= 0;
                    for a= 1:nLinks
                        if Loads(a,3)+Loads(a,4)>0
                            energy= energy + L(Loads(a,1),Loads(a,2));
                        end
                    end
                else
                    energy= inf;
                end
                if energy<best
                    k_best= k;
                    best= energy;
                end
            end
            if k_best>0
                sol(i)= k_best;
            else
                continuar= true;
                break;
            end
        end
    end
    energy= best;

    %HILL CLIMBING:
    continuar= true;
    while continuar
        i_best= 0;
        k_best= 0;
        best= energy;
```

```
            for i= 1:nFlows
                for k= 1:nSP(i)
                    if k~=sol(i)
                        aux= sol(i);
                        sol(i)= k;
                        Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                        load1= max(max(Loads(:,3:4)));
                        if load1 <= 10
                            energy1= 0;
                            for a= 1:nLinks
                                if Loads(a,3)+Loads(a,4)>0
                                    energy1= energy1 + L(Loads(a,1),Loads(a,2))
                                        ;
                                end
                            end
                        else
                            energy1= inf;
                        end
                        if energy1<best
                            i_best= i;
                            k_best= k;
                            best= energy1;
                        end
                        sol(i)= aux;
                    end
                end
            end
            if i_best>0
                sol(i_best)= k_best;
                energy= best;
            else
                continuar= false;
            end
        end
        allValues= [allValues energy];
        if energy<bestEnergy
            bestSol= sol;
            bestEnergy= energy;
        end
    end
    hold on
    grid on
    plot(sort(allValues));
    legend('Random with 10 shortest','Greedy randomized with 10 shortest','Hill
        climbing with 10 shortest',Location="southeast");
    fprintf('   Best energy = %.1f\n',bestEnergy);
    fprintf('   No. of solutions = %d\n',length(allValues));
    fprintf('   Av. quality of solutions = %.1f\n',mean(allValues));

    fprintf('\nSolution random with 5 shortest routing paths\n');
    t= tic;
    bestEnergy= inf;
    sol= zeros(1,nFlows);
    allValues= [];
    while toc(t)<10
```

```
    for i= 1:nFlows
        n = min(5,nSP(i));
        sol(i)= randi(n);
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    if load<=10
        energy = 0;
        for a=1:nLinks
            if Loads(a,3)+Loads(a,4)>0
                energy = energy + L(Loads(a,1),Loads(a,2));
            end
        end
    else
        energy=inf;
    end
    allValues= [allValues energy];
    if energy<bestEnergy
        bestSol= sol;
        bestEnergy= energy;
    end

end
figure(3);
grid on
plot(sort(allValues));
title('Compare the algorithms - Task 2.D')
fprintf('Energy consumption value of the best solution = %.1f \n',
    bestEnergy);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.1f \n', mean(
    allValues));

fprintf('\nSolution greedy ramdomized using 5 shortest routing paths\n');
t= tic;
bestEnergyGreedy= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10
    ax2 = randperm(nFlows);
    sol= zeros(1,nFlows);
    for i= ax2
        k_best = 0;
        best = inf;
        %encontrar o melhor vizinho
         n = min(5,nSP(i));
        for k = 1:n
            sol(i)= k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));

            if load<=10
                energy = 0;
                for a=1:nLinks
                    if Loads(a,3)+Loads(a,4)>0
```

```
                        energy = energy + L(Loads(a,1),Loads(a,2));
                    end
                end
            else
                energy=inf;
            end

            if energy < best
                k_best = k;
                best = energy;
            end
        end
        sol(i) = k_best;
    end
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    load= max(max(Loads(:,3:4)));
    if load<=10
        energy = 0;
        for a=1:nLinks
            if Loads(a,3)+Loads(a,4)>0
                energy = energy + L(Loads(a,1),Loads(a,2));
            end
        end
    else
        energy=inf;
    end

    allValues= [allValues energy];
    if energy <bestEnergyGreedy
        bestSol= sol;
        bestEnergyGreedy= energy;
    end
end

hold on
grid on
plot(sort(allValues));
fprintf('Energy consumption value of the best solution = %.1f \n', bestEnergyGreedy);
fprintf('Number of solutions generated = %d \n', length(allValues));
fprintf('Average quality of all solutions generated = %.1f \n', mean(allValues));


fprintf('\nSolution hill climbing using 5 shortest routing paths\n');
%Optimization algorithm with multi start hill climbing:
t= tic;
bestEnergy= inf;
allValues= [];
contadortotal= [];
while toc(t)<tempo
    %GREEDY RANDOMIZED:
    continuar= true;
    while continuar
        continuar= false;
```

```
        ax2= randperm(nFlows);
        sol= zeros(1,nFlows);
        for i= ax2
            k_best= 0;
            best= inf;
             n = min(5,nSP(i));
            for k= 1:n
                sol(i)= k;
                Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                load= max(max(Loads(:,3:4)));
                if load <= 10
                    energy= 0;
                    for a= 1:nLinks
                        if Loads(a,3)+Loads(a,4)>0
                            energy= energy + L(Loads(a,1),Loads(a,2));
                        end
                    end
                else
                    energy= inf;
                end
                if energy<best
                    k_best= k;
                    best= energy;
                end
            end
            if k_best>0
                sol(i)= k_best;
            else
                continuar= true;
                break;
            end
        end
    end
    energy= best;

    %HILL CLIMBING:
    continuar= true;
    while continuar
        i_best= 0;
        k_best= 0;
        best= energy;
        for i= 1:nFlows
            for k= 1:nSP(i)
                if k~=sol(i)
                    aux= sol(i);
                    sol(i)= k;
                    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
                    load1= max(max(Loads(:,3:4)));
                    if load1 <= 10
                        energy1= 0;
                        for a= 1:nLinks
                            if Loads(a,3)+Loads(a,4)>0
                                energy1= energy1 + L(Loads(a,1),Loads(a,2))
                                    ;
                            end
```

```
                        end
                    else
                        energy1= inf;
                    end
                    if energy1<best
                        i_best= i;
                        k_best= k;
                        best= energy1;
                    end
                    sol(i)= aux;
                end
            end
        end
        if i_best>0
            sol(i_best)= k_best;
            energy= best;
        else
            continuar= false;
        end
    end
    allValues= [allValues energy];
    if energy<bestEnergy
        bestSol= sol;
        bestEnergy= energy;
    end
end
hold on
grid on
plot(sort(allValues));
legend('Random with 5 shortest','Greedy randomized with 5 shortest','Hill
    climbing with 5 shortest',Location="southeast");

fprintf('    Best energy = %.1f\n',bestEnergy);
fprintf('    No. of solutions = %d\n',length(allValues));
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 estrategia 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 ramdomized 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 ramdomized 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 ramdomized 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 númericos para a alínea D nos diferentes casos.

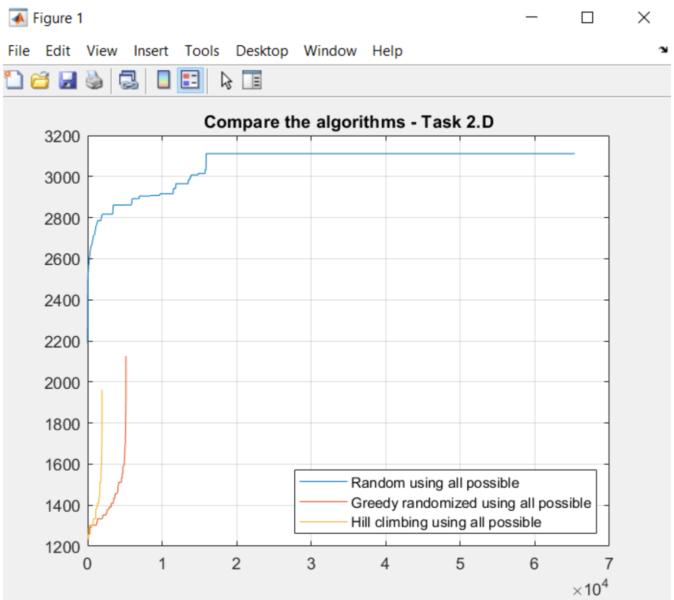

Figura 2.8: Resultados gráficos para a alínea D no caso de ter todas as rotas para os diferentes algoritmos.

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 obtem 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 greeedy 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 vantangens para calcular a energia comparado com o algoritmo greedy randomized.

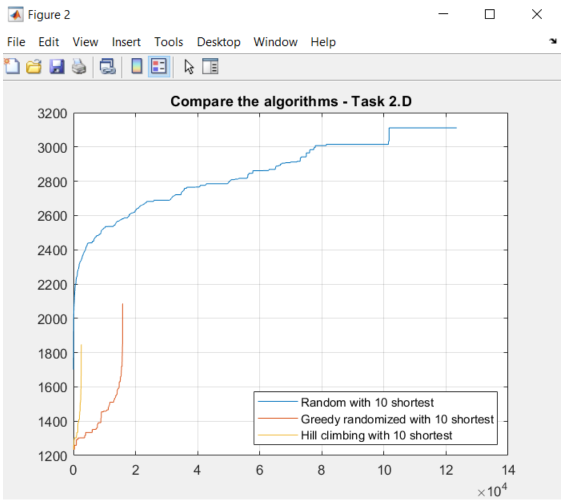

Figura 2.9: Resultados gráficos para a alínea D no caso de ter 10 rotas mais curtas para os diferentes algoritmos.

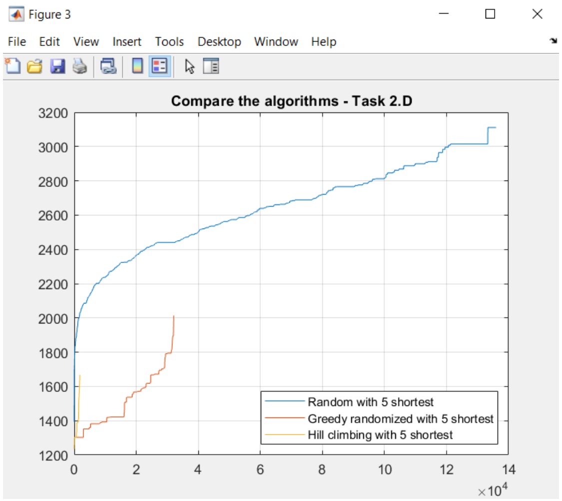

Figura 2.10: Resultados gráficos para a alínea D no caso de ter 5 rotas mais curtas para os diferentes algoritmos.

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 mensionado 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 dijskstra 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 minimo, que é o chamado 1º percurso.

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

```
Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
        540 290
        120 240
        400 310
        220 370
```

```
        550  380];
Links= [1  2
        1  5
        2  3
        2  4
        3  4
        3  6
        3  8
        4  5
        4  8
        5  7
        6  8
        6  10
        7  8
        7  9
        8  9
        9  10];
T= [1   3   1.0  1.0
    1   4   0.7  0.5
    2   7   2.4  1.5
    3   4   2.4  2.1
    4   9   1.0  2.2
    5   6   1.2  1.5
    5   8   2.1  2.5
    5   9   1.6  1.9
    6  10   1.4  1.6];
nNodes= 10;
nLinks= size(Links,1);
nFlows= size(T,1);
co= Nodes(:,1)+j*Nodes(:,2);
L= inf(nNodes);    %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
for i=1:nLinks
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km
fprintf("Task 3 - Alinea A\n");
MTBF = (450*360*24)./L;
A = MTBF./(MTBF+24); % a= availability
A(isnan(A))=0; % quando a matriz a tiver Nan mete essa posicao a 0 em vez do Nan
[sP nSP]= calculatePaths(L,T,1); %retorna o 1   caminho para cada link que e o melhor
%sP sao os caminhos e o nSp sao os custos dos caminhos sP
av = ones(1,nFlows); %avalibility tudo a 1 - inicializacao
for i = 1:nFlows
    aux = cell2mat(sP{i}); %transforma o {...} num array para conseguirmos aceder ao que esta dentro dos {}
    arr = size(aux); % n de linhas n de colunas -> nos queremos os nos = n de colunas
    fprintf('Fluxo %d:\n',i);
```

```
    fprintf('1  caminho: %d',aux(1));
    for j= 2:length(aux)
        fprintf('-%d',aux(j));
    end
    fprintf("\n");
    for j = 1:arr(1,2)-1 %percorre o fluxo i -> o array aux
        av(i) = av(i) * A(aux(j),aux(j+1));
        %estao sempre em serie, pois da sempre o caminho mais curto no
        %calculatePaths
    end
    aux = av(i)*100;
    fprintf("Disponibilidade= %f = %f%%\n",av(i),aux);
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 disponivel, 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:

```
Nodes= [30  70
        350  40
        550  180
        310  130
        100  170
        540  290
        120  240
        400  310
        220  370
        550  380];
Links= [1  2
         1  5
         2  3
         2  4
         3  4
         3  6
         3  8
         4  5
         4  8
         5  7
         6  8
         6  10
         7  8
         7  9
         8  9
         9  10];
T= [1    3    1.0  1.0
     1    4    0.7  0.5
     2    7    2.4  1.5
```

```
    3  4   2.4  2.1
    4  9   1.0  2.2
    5  6   1.2  1.5
    5  8   2.1  2.5
    5  9   1.6  1.9
    6 10   1.4  1.6];
nNodes= 10;
nLinks= size(Links,1);
nFlows= size(T,1);
co= Nodes(:,1)+j*Nodes(:,2);
L= inf(nNodes);    %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
for i=1:nLinks
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km
fprintf("Task 3 - Alinea B\n");
MTBF = (450*360*24)./L;
A = MTBF./(MTBF+24); % a= availability
A(isnan(A))=0; % quando a matriz a tiver Nan mete essa posicao a 0 em vez
    do Nan
AuxL = -log(A)*100;
[sP nSP sP2 nSP2]= calculateDisjointPaths(AuxL,T);
for i=1:nFlows
    aux = cell2mat(sP{1});
    arr = size(aux);
    for j=2:length(aux)
        AuxL(aux(j),aux(j-1))= inf;
        AuxL(aux(j-1),aux(j))= inf;
    end
end
%sP sao os caminhos e o nSp sao os custos dos caminhos sP
av = ones(1,nFlows); %avalibility tudo a 1 - inicializacao
for i = 1:nFlows
    aux = cell2mat(sP2{i}); %transforma o {...} num array para conseguirmos
        aceder ao que esta dentro dos {}
    arr = size(aux); % n de linhas n de colunas -> nos queremos os nos = n
        de colunas
    fprintf('Fluxo %d:',i);
    if ~isempty(sP2{i}{1})
        fprintf('\nSegundo caminho: %d',sP2{i}{1}(1));
        for j= 2:length(sP2{i}{1})
            fprintf('-%d',sP2{i}{1}(j));
        end
        for j = 1:arr(1,2)-1  %percorre o fluxo i -> o array aux
            av(i) = av(i) * A(aux(j),aux(j+1));
            %estao sempre em serie, pois da sempre o caminho mais curto no
            %calculatePaths
        end
        aux = av(i)*100;
        fprintf("\nDisponibilidade= %f = %f%%\n",av(i),aux);
```

```
        end
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:

```
Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
        540 290
        120 240
        400 310
        220 370
        550 380];
Links= [1 2
         1 5
         2 3
         2 4
         3 4
         3 6
         3 8
         4 5
         4 8
         5 7
         6 8
         6 10
         7 8
         7 9
         8 9
         9 10];
T= [1   3   1.0  1.0
     1   4   0.7  0.5
     2   7   2.4  1.5
     3   4   2.4  2.1
     4   9   1.0  2.2
     5   6   1.2  1.5
     5   8   2.1  2.5
     5   9   1.6  1.9
     6  10   1.4  1.6];
nNodes= 10;
nLinks= size(Links,1);
nFlows= size(T,1);
co= Nodes(:,1)+j*Nodes(:,2);
L= inf(nNodes);     %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
for i=1:nLinks
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km
fprintf("Task 3 - Alinea B\n");
MTBF = (450*360*24)./L;
A = MTBF./(MTBF+24); % a= availability
A(isnan(A))=0; % quando a matriz a tiver Nan mete essa posicao a 0 em vez
    do Nan
AuxL = -log(A)*100;
```

```
[sP nSP sP2 nSP2]= calculateDisjointPaths(AuxL,T);
for i=1:nFlows
    aux = cell2mat(sP{1});
    arr = size(aux);
    for j=2:length(aux)
        AuxL(aux(j),aux(j-1))= inf;
        AuxL(aux(j-1),aux(j))= inf;
    end
end
%sP sao os caminhos e o nSp sao os custos dos caminhos sP
av = ones(1,nFlows); %avalibility tudo a 1 - inicializacao
for i = 1:nFlows
    aux = cell2mat(sP2{i}); %transforma o {...} num array para conseguirmos
        aceder ao que esta dentro dos {}
    arr = size(aux); % n de linhas n de colunas -> nos queremos os nos = n
        de colunas
    fprintf('Fluxo %d:',i);
    if ~isempty(sP2{i}{1})
        fprintf('\nSegundo caminho: %d',sP2{i}{1}(1));
        for j= 2:length(sP2{i}{1})
            fprintf('-%d',sP2{i}{1}(j));
        end
        for j = 1:arr(1,2)-1 %percorre o fluxo i -> o array aux
            av(i) = av(i) * A(aux(j),aux(j+1));
            %estao sempre em serie, pois da sempre o caminho mais curto no
            %calculatePaths
        end
        aux = av(i)*100;
        fprintf("\nDisponibilidade= %f = %f%%\n",av(i),aux);
    end
end
bandwith= calculateLinkLoads1plus1(nNodes,Links,T,sP,sP2)
totalbandwith= sum(sum(bandwith(:,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.

```
bandwith =

    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


totalbandwith =

  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 obtem 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:

```
Nodes= [30  70
        350  40
        550  180
```

```
          310  130
          100  170
          540  290
          120  240
          400  310
          220  370
          550  380];
Links= [1  2
          1  5
          2  3
          2  4
          3  4
          3  6
          3  8
          4  5
          4  8
          5  7
          6  8
          6  10
          7  8
          7  9
          8  9
          9  10];
T= [1   3   1.0  1.0
     1   4   0.7  0.5
     2   7   2.4  1.5
     3   4   2.4  2.1
     4   9   1.0  2.2
     5   6   1.2  1.5
     5   8   2.1  2.5
     5   9   1.6  1.9
     6  10   1.4  1.6];
nNodes= 10;
nLinks= size(Links,1);
nFlows= size(T,1);
co= Nodes(:,1)+j*Nodes(:,2);
L= inf(nNodes);    %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
for i=1:nLinks
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km
fprintf("Task 3 - Alinea B\n");
MTBF = (450*360*24)./L;
A = MTBF./(MTBF+24); % a= availability
A(isnan(A))=0; % quando a matriz a tiver Nan mete essa posicao a 0 em vez
    do Nan
AuxL = -log(A)*100;
[sP nSP sP2 nSP2]= calculateDisjointPaths(AuxL,T);
for i=1:nFlows
    aux = cell2mat(sP{1});
```

```
    arr = size(aux);
    for j=2:length(aux)
        AuxL(aux(j),aux(j-1))= inf;
        AuxL(aux(j-1),aux(j))= inf;
    end
end
%sP sao os caminhos e o nSp sao os custos dos caminhos sP
av = ones(1,nFlows); %avalibility tudo a 1 - inicializacao
for i = 1:nFlows
    aux = cell2mat(sP2{i}); %transforma o {...} num array para conseguirmos
        aceder ao que esta dentro dos {}
    arr = size(aux); % n de linhas n de colunas -> nos queremos os nos = n
       de colunas
    fprintf('Fluxo %d:',i);
    if ~isempty(sP2{i}{1})
        fprintf('\nSegundo caminho: %d',sP2{i}{1}(1));
        for j= 2:length(sP2{i}{1})
            fprintf('-%d',sP2{i}{1}(j));
        end
        for j = 1:arr(1,2)-1  %percorre o fluxo i -> o array aux
            av(i) = av(i) * A(aux(j),aux(j+1));
            %estao sempre em serie , pois da sempre o caminho mais curto no
            %calculatePaths
        end
        aux = av(i)*100;
        fprintf("\nDisponibilidade= %f = %f%%\n",av(i),aux);
    end
end
bandwith= calculateLinkLoads1to1(nNodes,Links,T,sP,sP2)
totalbandwith= sum(sum(bandwith(:,3:4)))
[~, index] = max(bandwith(:,3:4));
maxbandwith = bandwith(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.

```
bandwith =

    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


totalbandwith =

  113.4000


maxbandwith =

    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 alinea 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:

```
clear all;
close all;

Nodes= [30 70
        350 40
        550 180
        310 130
        100 170
        540 290
        120 240
        400 310
        220 370
        550 380];
Links= [1 2
         1 5
         2 3
```

```
            2  4
            3  4
            3  6
            3  8
            4  5
            4  8
            5  7
            6  8
            6  10
            7  8
            7  9
            8  9
            9  10];

T= [1   3   1.0  1.0
    1   4   0.7  0.5
    2   7   2.4  1.5
    3   4   2.4  2.1
    4   9   1.0  2.2
    5   6   1.2  1.5
    5   8   2.1  2.5
    5   9   1.6  1.9
    6  10   1.4  1.6];

nNodes= 10;
nLinks= size(Links,1);
nFlows= size(T,1);
co= Nodes(:,1)+j*Nodes(:,2);
L= inf(nNodes);    %Square matrix with arc lengths (in Km)
for i=1:nNodes
    L(i,i)= 0;
end
for i=1:nLinks
    d= abs(co(Links(i,1))-co(Links(i,2)));
    L(Links(i,1),Links(i,2))= d+5; %Km
    L(Links(i,2),Links(i,1))= d+5; %Km
end
L= round(L);  %Km
fprintf("Task 3 - Alinea A\n");
MTBF = (450*360*24)./L;
A = MTBF./(MTBF+24); % a= availability
A(isnan(A))=0; % quando a matriz a tiver Nan mete essa posicao a 0 em vez
   do Nan
AuxL = -log(A);
%[sP nSP sP2 nSP2]= calculateDisjointPaths(AuxL,T);
shortestPathFinal=[];
for i=1:nFlows
    [shortestPath, totalCost] = kShortestPath(AuxL,T(i,1),T(i,2),10); %
       queremos os 10 percursos com custo minimo

    NewAuxL=AuxL;
    for k= 1:10
        tempshortestPath = cell2mat(shortestPath(k));

        for j=2:length(tempshortestPath)
```

```
                NewAuxL(tempshortestPath(j),tempshortestPath(j-1))= inf;
                NewAuxL(tempshortestPath(j-1),tempshortestPath(j))= inf;
            end
            [shortestPath2, totalCost2] = kShortestPath(NewAuxL,T(i,1),T(i,2),1); %queremos os 1 percursos com custo minimo
            shortestPathFinal = [shortestPathFinal shortestPath2];
        end
end


for i = 1:nFlows
    % arr = size(aux); % n de linhas n de colunas -> nos queremos os nos = n de colunas

    fprintf('Fluxo %d:',i);
    for k=1:10
        aux = cell2mat(shortestPathFinal(k)); %transforma o {...} num array para conseguirmos aceder ao que esta dentro dos {}

       fprintf('\nPath %d : %d',k,aux(1));
        for j= 2:length(aux)
            fprintf('-%d',aux(j));
        end
        if ~isempty(aux(2))
             fprintf('\tSecond path: %d',aux(1));
             for j= 2:length(aux)
                 fprintf('-%d',aux(j));
             end
         else
             fprintf('No paths');
         end

    end
    fprintf('\n');
end
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.

```
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
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
```

```
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
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
```

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

```
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
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
```

```
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
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
```

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

Capítulo 5

Contribuições dos autores

Mariana Pinto - 50%
Raquel Pinto - 50%