universidade
de aveiro

DEPARTAMENTO DE ELETRÓNICA, TELECOMUNICAÇÕES E INFORMÁTICA

MESTRADO EM ENGENHARIA DE COMPUTADORES E TELEMÁTICA
ANO 2021/2022

MODELAÇÃO E DESEMPENHO DE REDES E SERVIÇOS

MINI-PROJECT 2
TRAFFIC ENGINEERING OF TELECOMMUNCATION NETWORKS

Tiago Dias (88896)

Rita Amante (89264)

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. 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?

Matlab code

```
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);

for i = 1:nNodes
    L(i,i) = 0;
end

for i = 1:nLinks
    d = abs(co(Links(i,1)) - co(Links(i,2)));
```

```
55      L(Links(i,1),Links(i,2)) = d+5;
56      L(Links(i,2),Links(i,1)) = d+5;
57  end
58
59  L = round(L);
60  n = inf;
61  [sP nSP] = calculatePaths(L,T,n);
62
63  fprintf('With a k-shortest path algorithm (using the lengths of the links):\n');
64
65  for i = 1:nFlows
66      fprintf('   Flow %d has %d different routing paths provided by the network.\n', i, nSP(i));
67  end
```

Code analysis

Primeiramente, definiram-se três matrizes: a matriz com a localização de cada nó para depois calcular o comprimento das ligações, onde a primeira coluna corresponde à coordenada x e a segunda à coordenada y (linhas 4 a 13); a matriz que contém todas as ligações da rede (linhas 15 a 30) e a matriz para cada fluxo, onde a primeira coluna corresponde ao nó origem, a segunda ao nó destino, a terceira ao débito binário origem-destino e a quarta coluna ao débito binário destino-origem (linhas 32 a 40).

De seguida, inicializaram-se algumas variáveis como o número de nós na rede (linha 42), o número de ligações (linha 43), o número de fluxos (linha 44) e os números complexos, onde a parte real corresponde à coordenada x e a parte imaginária à coordenada y (linha 45).

Posteriormente, definiu-se uma matriz L que contém os comprimentos, em km, de cada ligação ij, ou infinito se a ligação não existir, com a diagonal preenchida a zeros (linhas 47 a 59).

Depois, definiram-se quantos caminhos se pretende usar, onde n = inf corresponde a todos os caminhos possíveis na rede (linha 60). Calcularam-se todos os caminhos da rede para cada fluxo, do mais curto para o mais longo, com a ajuda da função auxiliar *calculatePaths(L,T,n)* que devolve, para cada fluxo, em sP os caminhos possíveis e em nSP o número total de caminhos.

Por fim, imprimiu-se o número total de caminhos para cada fluxo (linhas 63 a 67).

O código das linhas 1 a 61, será utilizado em todas as restantes tarefas.

Result

```
With a k-shortest path algorithm (using the lengths of the links):
   Flow 1 has 32 different routing paths provided by the network.
   Flow 2 has 32 different routing paths provided by the network.
   Flow 3 has 38 different routing paths provided by the network.
   Flow 4 has 24 different routing paths provided by the network.
   Flow 5 has 36 different routing paths provided by the network.
   Flow 6 has 37 different routing paths provided by the network.
   Flow 7 has 25 different routing paths provided by the network.
   Flow 8 has 41 different routing paths provided by the network.
   Flow 9 has 28 different routing paths provided by the network.
```

Conclusions

Por forma a calcular o número de diferentes caminhos de roteamento fornecidos pela rede para cada fluxo de tráfego, foi utilizado o algoritmo *Dijkstra*, onde são exploradas todos os possíveis caminhos do nó inicial do fluxo até ao nó final do mesmo, sem repetir nós.

Os possíveis caminhos identificados estão em formato *cell*, por ordem crescente do caminho mais curto para o mais longo, na variável sP e a soma dos diferentes caminhos gerados é obtida na variável nSP.

Conclui-se, a partir de uma observação mais detalhada de algumas *cells* de cada fluxo, que o caminho mais curto não é necessariamente o caminho com menos nós, quanto maior a distância entre os dois nós, maior é o número de caminhos de encaminhamento.

1.b. 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 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.

Matlab code

```
1  fprintf('RANDOM STRATEGY\n');
2  fprintf('   Using all possible routing paths:\n');
3
4  n = inf;
5  [sP nSP] = calculatePaths(L,T,n);
6
7  sol = ones(1,nFlows);
8  Loads = calculateLinkLoads(nNodes,Links,T,sP,sol);
9  maxLoad = max(max(Loads(:,3:4)));
10
11 time = 10;
12
13 t = tic;
14 bestLoad = inf;
15 sol = zeros(1,nFlows);
16 allValues = [];
17
18 while toc(t) < time
19     for i = 1:nFlows
20         sol(i) = randi(nSP(i));
21     end
22     Loads = calculateLinkLoads(nNodes,Links,T,sP,sol);
23     load = max(max(Loads(:,3:4)));
24     allValues = [allValues load];
25     if load < bestLoad
26         bestSol = sol;
27         bestLoad = load;
28     end
29 end
30
31 fprintf('      Worst load = %.2f Gbps\n', bestLoad);
32 fprintf('      No. of solutions = %d\n', length(allValues));
33 fprintf('      Av. quality of solutions = %.2f Gbps\n\n', mean(allValues));
34
35 figure(1);
36 hold on
37 plot(sort(allValues));
38 ...
39 title({'Random algorithm'}, {'to minimize the worst link load'});
40 xlabel('No. of solutions');
41 ylabel('Worst Load (Gbps)');
42 legend('All possible routing paths','10 shortest routing paths','5 shortest routing
43 paths','Location','northwest');
```

Code analysis

Primeiramente, definiu-se o número de caminhos a utilizar no algoritmo *Random* (linha 4), que irá variar entre inf (usando todos os caminhos possíveis na rede), 10 (usando 10 caminhos de roteamento mais curtos) e 5 (usando 5 caminhos de roteamento mais curtos).

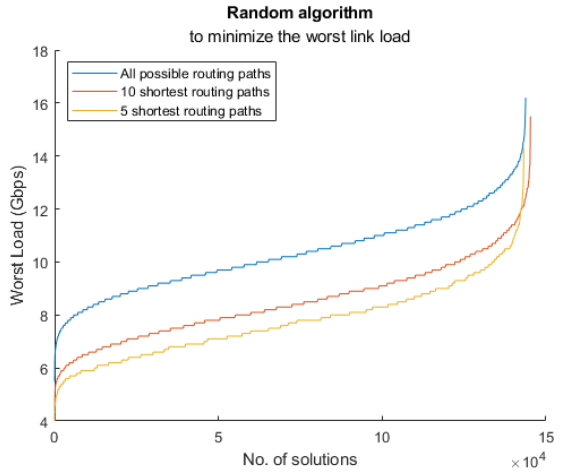
De seguida, calcularam-se os n caminhos da rede para cada fluxo (linha 5) e as cargas das ligações usando o primeiro caminho mais curto de cada fluxo (linhas 7 a 9), definiu-se o critério de paragem (linha 11) e inicializaram-se algumas variáveis auxiliares (linhas 13 a 16).

Enquanto o tempo não ultrapassa o estipulado (linhas 18 a 29), selecionou-se um caminho aleatório para cada fluxo, calcularam-se as cargas da solução gerada, verificou-se o maior valor das cargas entre a terceira e quarta coluna, guardaram-se todos os valores de carga máxima de todas as soluções e ficou-se com a melhor solução de todas (linhas 19 a 28).

Este processo repete-se para n = 10 e n = 5 caminhos mais curtos. Para cada valor de n, imprimiu-se o pior valor de carga da melhor ligação, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções (linhas 31 a 33) e, por fim, desenhou-se o gráfico com as melhores cargas de todas as soluções geradas para cada simulação de n (linhas 35 a 43).

É importante salientar que, quando a função objetivo é minimizar a carga máxima, não é preciso se preocupar se a carga máxima ultrapassa os 10 Gbps. Escolhe-se o melhor percurso e, mesmo que a carga máxima seja superior a 10 Gbps, não há problema porque como se pretende minimizar a carga máxima, pode-se começar com uma carga superior a 10Gbps e depois, ou o algoritmo *Hill Climbing* consegue baixar a carga ou, se no fim tiver a solução superior a 10 Gbps, desde que haja uma solução abaixo dos 10 Gbps, ignora-se todas as soluções acima do 10 Gbps.

Result




```
RANDOM STRATEGY
   Using all possible routing paths:
      Worst load = 5.10 Gbps
      No. of solutions = 145524
      Av. quality of solutions = 10.32 Gbps

   Using 10 shortest routing paths:
      Worst load = 4.30 Gbps
      No. of solutions = 145942
      Av. quality of solutions = 8.51 Gbps

   Using 5 shortest routing paths:
      Worst load = 4.00 Gbps
      No. of solutions = 145847
      Av. quality of solutions = 7.75 Gbps
```

Conclusions

Relativamente à qualidade das soluções e à melhor carga, quando n = inf, ou seja, quando são utilizados todos os caminhos possíveis na rede para cada fluxo, obtêm-se soluções com baixa qualidade, pois como como se calcula a carga para todos os caminhos possíveis, considerando tanto os mais curtos como os mais longos, a qualidade média das soluções é elevada.

Quando n = 10, ou seja, quando são utilizados os 10 caminhos mais curtos, obtêm-se soluções com melhor qualidade, o que aumenta a probabilidade de encontrar uma carga melhor, visto que se restringe o intervalo de procura para 10 caminhos mais curtos para cada fluxo em vez de se analisarem todos os caminhos possíveis.

Quando n = 5, ou seja, quando são utilizados os 5 caminhos mais curtos, obtêm-se soluções com melhor qualidade, o que aumenta ainda mais a probabilidade de encontrar uma carga melhor.

Relativamente ao número de soluções geradas, para cada um dos valores de n, variam ligeiramente, mas, de uma vista geral, o número de caminhos de roteamento não influencia o número de soluções.

Concluindo, o número de caminhos de roteamento influencia a eficiência do algoritmo *Random*, pois quanto menor for o número de caminhos, menor será a melhor carga da ligação e melhor será a qualidade das soluções. No entanto, o número de caminhos não influencia o número de soluções geradas.

1.c. 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.

Matlab code

```
fprintf('GREEDY RANDOMIZED STRATEGY\n');
fprintf('   Using all possible routing paths:\n');

n = inf;
[sP nSP] = calculatePaths(L,T,n);

sol = ones(1,nFlows);
Loads = calculateLinkLoads(nNodes,Links,T,sP,sol);
maxLoad = max(max(Loads(:,3:4)));

time = 10;

t = tic;
bestLoad = inf;
allValues = [];

while toc(t) < time
    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
    load = best;
    allValues = [allValues load];
    if load < bestLoad
        bestSol = sol;
        bestLoad = load;
    end
end

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

figure(2);
hold on
plot(sort(allValues));
...
title({'Greedy Randomized algorithm'}, {'to minimize the worst link load'});
xlabel('No. of solutions');
ylabel('Worst Load (Gbps)');
legend('All possible routing paths','10 shortest routing paths','5 shortest routing
paths','Location','southeast');
```

Code analysis

Primeiramente, definiu-se o número de caminhos a utilizar no algoritmo *Greedy Randomized* (linha 4), que irá variar entre inf (usando todos os caminhos possíveis na rede), 10 (usando 10 caminhos de roteamento mais curtos) e 5 (usando 5 caminhos de roteamento mais curtos).

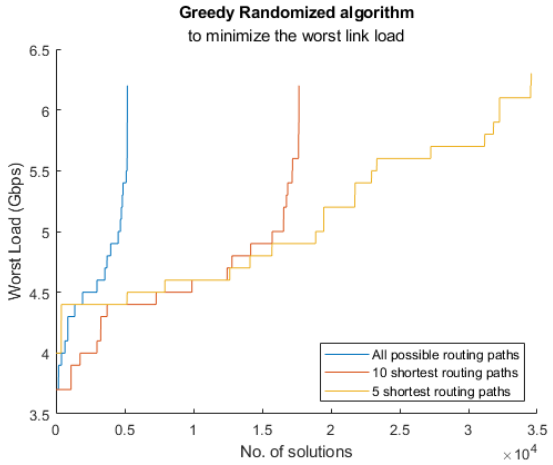
De seguida, calcularam-se os n caminhos da rede para cada fluxo (linha 5) e as cargas das ligações usando o primeiro caminho mais curto de cada fluxo (linhas 7 a 9), definiu-se o critério de paragem (linha 11) e inicializaram-se algumas variáveis auxiliares (linhas 13 a 15).

Enquanto o tempo não ultrapassa o estipulado (linhas 17 a 40), escolheu-se uma ordem aleatória para os fluxos e depois, para cada fluxo, por essa ordem, escolheu-se o percurso que dá a melhor função objetivo. Ou seja, o i vai rodar por todos os fluxos pela ordem ax2, vão calcular-se as cargas e a carga máxima entre a terceira e quarta coluna e vai escolher-se o melhor percurso para o fluxo i da função objetivo (linhas 18 a 39).

Este processo repete-se para n = 10 e n = 5 caminhos mais curtos. Para cada valor de n, imprimiu-se o pior valor de carga da melhor ligação, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções (linhas 42 a 44) e, por fim, desenhou-se o gráfico com as melhores cargas de todas as soluções geradas para cada simulação de n (linhas 46 a 54).

O algoritmo implementado é *greedy* porque sempre que se escolhe um percurso, escolhe-se o melhor e é *randomized* pois escolhe-se uma ordem para os fluxos e ao se escolher diferentes ordens, o resultado são diferentes soluções.

Result




```
GREEDY RANDOMIZED STRATEGY
   Using all possible routing paths:
      Worst load = 3.70 Gbps
      No. of solutions = 5594
      Av. quality of solutions = 4.55 Gbps

   Using 10 shortest routing paths:
      Worst load = 3.70 Gbps
      No. of solutions = 18600
      Av. quality of solutions = 4.52 Gbps

   Using 5 shortest routing paths:
      Worst load = 4.00 Gbps
      No. of solutions = 31989
      Av. quality of solutions = 5.07 Gbps
```

Conclusions

Relativamente à qualidade das soluções e à melhor carga, quando n = inf, ou seja, quando são utilizados todos os caminhos possíveis na rede para cada fluxo, obtém-se uma eficiência boa, com poucas soluções geradas e com boa qualidade média das soluções.

Quando n = 10, ou seja, quando são utilizados os 10 caminhos mais curtos, obtém-se uma eficiência semelhante à anterior quando n = inf, mas com um aumento bastante significativo no número de soluções geradas pelo algoritmo.

Quando n = 5, ou seja, quando são utilizados os 5 caminhos mais curtos, a eficiência do algoritmo piora, geram-se mais soluções com pior qualidade e o valor da melhor carga aumenta.

Concluindo, o número de caminhos de roteamento influencia a eficiência do algoritmo *Greedy Randomized*, pois quanto menor for o número de caminhos, maior será a melhor carga da ligação e pior será a qualidade média das soluções, aumentando o número de soluções geradas.

1.d. 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.

Matlab code

```
fprintf('MULTI START HILL CLIMBING STRATEGY\n');
fprintf('   Using all possible routing paths:\n');

n = inf;
[sP nSP] = calculatePaths(L,T,n);

sol = ones(1,nFlows);
Loads = calculateLinkLoads(nNodes,Links,T,sP,sol);
maxLoad = max(max(Loads(:,3:4)));

time = 10;

t = tic;
bestLoad = inf;
allValues = [];
contadortotal = [];

while toc(t) < time
    % 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
    load = best;
    % Multi start 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;
```

```
61              load = best;
62          else
63              continuar = false;
64          end
65      end
66      allValues = [allValues load];
67      if load < bestLoad
68          bestSol = sol;
69          bestLoad = load;
70      end
71  end
72
73  fprintf('      Best load = %.2f Gbps\n', bestLoad);
74  fprintf('      No. of solutions = %d\n', length(allValues));
75  fprintf('      Av. quality of solutions = %.2f Gbps\n\n', mean(allValues));
76
77  figure(3);
78  hold on
79  plot(sort(allValues));
80  ...
81  title({'Multi start Hill CLimbing algorithm'}, {'to minimize the worst link load'});
82  xlabel('No. of solutions');
83  ylabel('Worst Load (Gbps)');
84  legend('All possible routing paths','10 shortest routing paths','5 shortest routing
85  paths','Location','southeast');
```

Code analysis

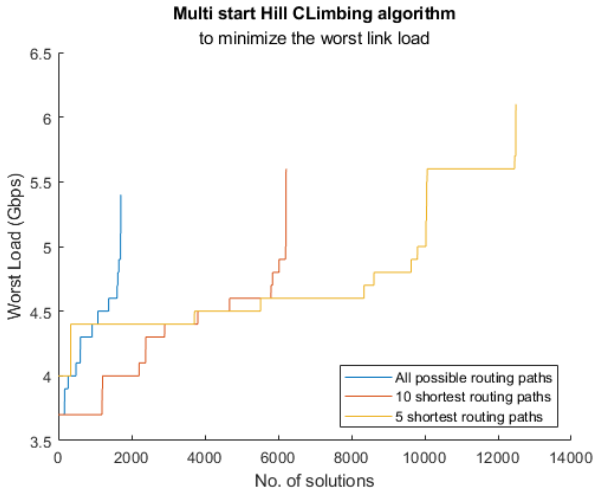
Primeiramente, definiu-se o número de caminhos a utilizar no algoritmo *Multi Start Hill Climbing* (linha 4), que irá variar entre inf (usando todos os caminhos possíveis na rede), 10 (usando 10 caminhos de roteamento mais curtos) e 5 (usando 5 caminhos de roteamento mais curtos).

De seguida, calcularam-se os n caminhos da rede para cada fluxo (linha 5) e as cargas das ligações usando o primeiro caminho mais curto de cada fluxo (linhas 7 a 9), definiu-se o critério de paragem (linha 11) e inicializaram-se algumas variáveis auxiliares (linhas 13 a 16).

Enquanto o tempo não ultrapassa o estipulado (linhas 18 a 71), em primeiro lugar, construiu-se uma solução, usando o algoritmo *Greedy Randomized* (linhas 19 a 36) e, com essa solução, calculou-se uma nova solução aplicando o algoritmo *Hill Climbing* (linhas 37 a 70). Já tendo o percurso escolhido para cada fluxo testaram-se todas as soluções dadas pela troca de um percurso por um outro para cada fluxo, ou seja, testando todas essas soluções, verifica-se se a solução é melhor que a atual e, caso seja, troca-se e volta-se a repetir até que nenhuma das trocas individuais seja melhor (mínimo local).

Este processo repete-se para n = 10 e n = 5 caminhos mais curtos. Para cada valor de n, imprimiu-se o pior valor de carga da melhor ligação, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções (linhas 73 a 75) e, por fim, desenhou-se o gráfico com as melhores cargas de todas as soluções geradas para cada simulação de n (linhas 77 a 85).

Result




```
MULTI START HILL CLIMBING STRATEGY
   Using all possible routing paths:
      Worst load = 3.70 Gbps
      No. of solutions = 1717
      Av. quality of solutions = 4.29 Gbps

   Using 10 shortest routing paths:
      Worst load = 3.70 Gbps
      No. of solutions = 6076
      Av. quality of solutions = 4.28 Gbps

   Using 5 shortest routing paths:
      Worst load = 4.00 Gbps
      No. of solutions = 12449
      Av. quality of solutions = 4.75 Gbps
```

Conclusions

Relativamente à qualidade das soluções e à melhor carga, quando n = inf, ou seja, quando são utilizados todos os caminhos possíveis na rede para cada fluxo, obtém-se uma eficiência bastante boa, com poucas soluções geradas e com boa qualidade média das soluções.

Quando n = 10, ou seja, quando são utilizados os 10 caminhos mais curtos, obtém-se uma eficiência igual à anterior quando n = inf, mas com um aumento significativo no número de soluções geradas pelo algoritmo.

Quando n = 5, ou seja, quando são utilizados os 5 caminhos mais curtos, a eficiência do algoritmo piora, geram-se mais soluções com pior qualidade e o valor da melhor carga aumenta.

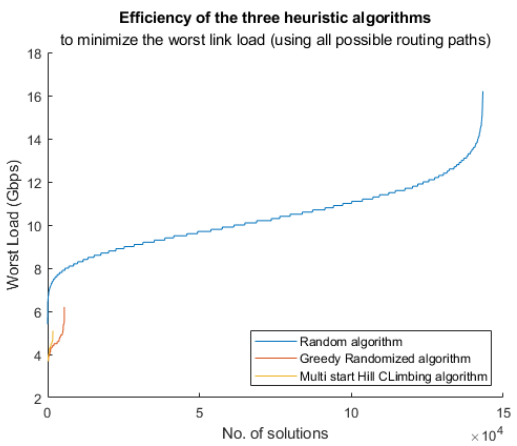
Concluindo, o número de caminhos de roteamento influencia a eficiência do algoritmo *Multi Start Hill Climbing*, pois quanto menor for o número de caminhos, maior será a melhor carga da ligação e pior será a qualidade média das soluções, aumentando o número de soluções geradas.

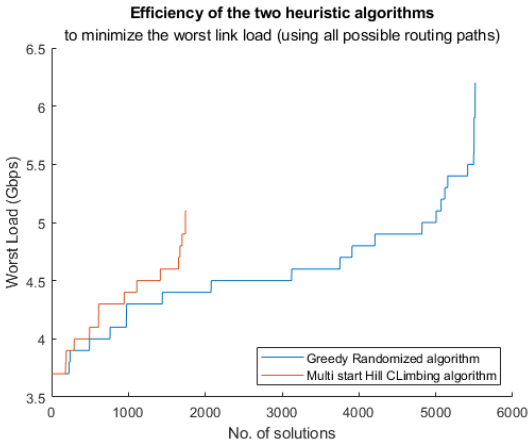
1.e. Compare the efficiency of the three heuristic algorithms based on the results obtained in **1.b**, **1.c** and **1.d**.

Code analysis

Para esta alínea reaproveitaram-se os códigos das alíneas 1.b, 1.c e 1.d, considerando apenas um caso: n = inf (usando todos os caminhos possíveis na rede).

Result






```
RANDOM STRATEGY
   Worst load = 5.10 Gbps
   No. of solutions = 145208
   Av. quality of solutions = 10.32 Gbps

GREEDY RANDOMIZED STRATEGY
   Worst load = 3.70 Gbps
   No. of solutions = 5376
   Av. quality of solutions = 4.55 Gbps

MULTI START HILL CLIMBING STRATEGY
   Worst load = 3.70 Gbps
   No. of solutions = 1843
   Av. quality of solutions = 4.28 Gbps
```

Conclusions

Relativamente ao algoritmo *Random*, é notório que é um algoritmo pouco eficiente quando se pretende minimizar a carga máxima, uma vez que apresenta um elevado número de soluções geradas, com má qualidade e a carga máxima ainda continua a estar elevada.

O algoritmo *Greedy Randomized* e o *Multi Start Hill Climbing* foram igualmente eficientes, obtendo uma melhor solução com o mesmo valor. No entanto, o algoritmo *Multi Start Hill Climbing* gera muito menos soluções que o algoritmo *Greedy Randomized* e com melhor qualidade média de soluções.

Concluindo, o *Multi Start Hill Climbing* traz ganhos para a função objetivo pois consegue minimizar a carga máxima.

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. 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.

Matlab code

```
fprintf('RANDOM STRATEGY\n');
fprintf('   Using all possible routing paths:\n');

n = inf;
[sP nSP] = calculatePaths(L,T,n);

time = 10;

t = tic;
bestEnergy = inf;
sol = zeros(1,nFlows);
allValues = [];

while toc(t) < time
    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
fprintf('      Best energy = %.1f Km\n', bestEnergy);
fprintf('      No. of solutions = %d\n', length(allValues));
fprintf('      Av. quality of solutions = %.1f Km\n\n', mean(allValues));

figure(1);
hold on
plot(sort(allValues));
...
title({'Random algorithm'}, {'to minimize the energy consumption of the network'});
xlabel('No. of solutions');
```

```
46  ylabel('Best energy (Km)');
47  legend('All possible routing paths','10 shortest routing paths','5 shortest routing
48  paths','Location','southeast');
```

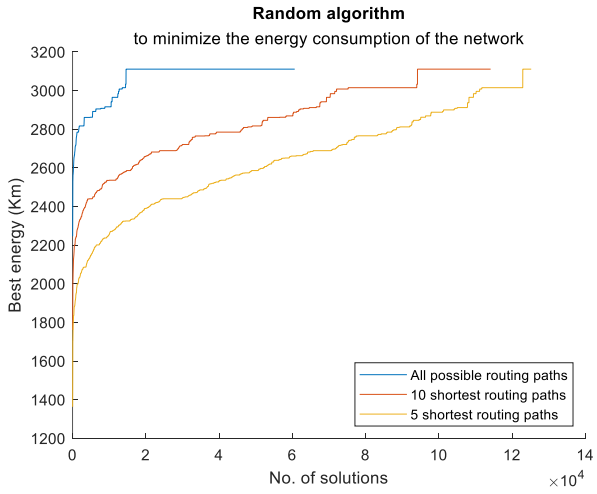
Code analysis

Na tarefa 2, a função objetivo é diferente que a tarefa 1, em vez de ser para minimizar a carga máxima é para minimizar o consumo de energia. Quando a função objetivo é minimizar o consumo de energia, há tendência a concentrar os fluxos no menor número de ligações para que se obtenha o máximo de ligações possíveis sem suportar fluxos para se poderem colocar em *sleeping mode* (minimizar a energia). Sendo, então, preciso forçar, sempre que se constrói uma solução, a que essa solução não ultrapasse os 10 Gbps.

Reaproveitou-se o código da experiência 1.b, fazendo algumas alterações. Enquanto o tempo não ultrapassa o estipulado (linhas 14 a 35), selecionou-se um caminho de roteamento aleatório para cada fluxo, calcularam-se as cargas da solução gerada, verificou-se o maior valor das cargas entre a terceira e quarta coluna e verificou-se se a carga máxima da solução não ultrapasse os 10 Gbps pois, caso ultrapasse, é ignorada e volta-se a gerar uma nova solução até que a carga máxima não ultrapasse os 10 Gbps, caso contrário, a solução é aceite. Calculou-se a energia como a soma dos comprimentos de todas as ligações que não estão em *spleeping mode*, uma vez que a energia é proporcional ao comprimento das ligações que não estão em *spleeping mode*. Verificou-se se a energia é menor que o *best* e, caso seja, guarda-se esse valor (linhas 15 a 34).

Este processo repete-se para n = 10 e n = 5 caminhos mais curtos. Para cada valor de n, imprimiu-se o valor de consumo de energia da melhor solução, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções (linhas 36 a 38) e, por fim, desenhou-se o gráfico com as melhores energias de todas as soluções geradas para cada simulação de n (linhas 40 a 48).

Result




```
RANDOM STRATEGY
   Using all possible routing paths:
      Best energy = 2246.0 Km
      No. of solutions = 133396
      Av. quality of solutions = Inf Km

   Using 10 shortest routing paths:
      Best energy = 1761.0 Km
      No. of solutions = 134017
      Av. quality of solutions = Inf Km

   Using 5 shortest routing paths:
      Best energy = 1363.0 Km
      No. of solutions = 133627
      Av. quality of solutions = Inf Km
```

Conclusions

Por observação direta do gráfico, o algoritmo *Random* atinge um limite máximo aos 3111 que é quando a solução passa por todos os nós. Logo, a qualidade média das soluções tende sempre para inf.

Quanto à melhor energia, quando n = inf, ou seja, quando são utilizados todos os caminhos possíveis na rede para cada fluxo, obtém-se pior eficiência, com um valor elevado da melhor energia. Quando n = 10, ou seja, quando são utilizados os 10 caminhos mais curtos, obtém-se uma melhor energia, visto que se restringe o intervalo de procura para 10 caminhos mais curtos para cada fluxo. Quando n = 5, ou seja, quando são utilizados os 5 caminhos mais curtos, obtém-se uma energia ainda melhor.

Concluindo, o número de caminhos de roteamento influencia a eficiência do algoritmo *Random*, pois quanto menor for o número de caminhos, menor será o consumo de energia, pois os caminhos possíveis para a sua escolha são também menores.

2.b. 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.

Matlab code

```
fprintf('GREEDY RANDOMIZED STRATEGY\n');
fprintf('   Using all possible routing paths:\n');

n = inf;
[sP nSP] = calculatePaths(L,T,n);

time = 10;

t = tic;
bestEnergy = inf;
allValues = [];

while toc(t) < time
    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;
    allValues = [allValues energy];
    if energy < bestEnergy
        bestSol = sol;
        bestEnergy = energy;
    end
end
fprintf('      Best energy = %.1f Km\n', bestEnergy);
fprintf('      No. of solutions = %d\n', length(allValues));
fprintf('      Av. quality of solutions = %.1f Km\n\n', mean(allValues));

figure(2);
```

```
61  hold on
62  plot(sort(allValues));
63  ...
64  title({'Greedy Randomized algorithm'}, {'to minimize the energy consumption of the network'});
65  xlabel('No. of solutions');
66  ylabel('Best energy (Km)');
67  legend('All possible routing paths','10 shortest routing paths','5 shortest routing
68  paths','Location','southeast');
```

Code analysis

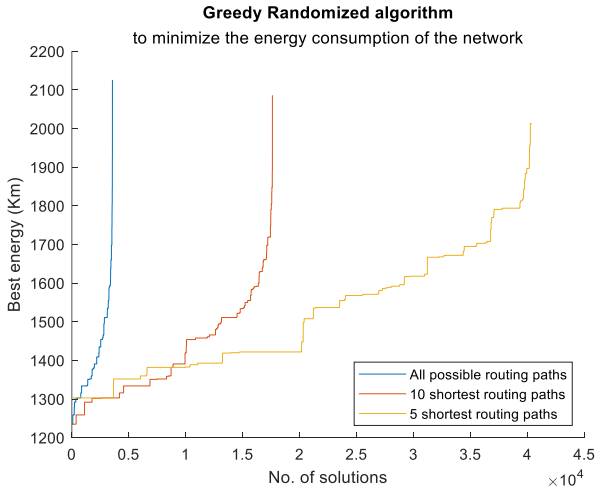
Primeiramente, definiu-se o número de caminhos a utilizar no algoritmo *Greedy Randomized* (linha 4), que irá variar entre inf (usando todos os caminhos possíveis na rede), 10 (usando 10 caminhos de roteamento mais curtos) e 5 (usando 5 caminhos de roteamento mais curtos).

De seguida, calcularam-se os n caminhos da rede para cada fluxo (linha 5), definiu-se o critério de paragem (linha 7) e inicializaram-se algumas variáveis auxiliares (linhas 9 a 11).

Enquanto o tempo não ultrapassa o estipulado (linhas 13 a 55), criou-se um ciclo *while* para construir a solução (linhas 15 a 48). Dentro deste ciclo, escolheu-se uma ordem aleatória para os fluxos e depois, para cada fluxo, por essa ordem, escolheu-se o percurso que dá a melhor função objetivo. Ou seja, o i vai rodar por todos os fluxos pela ordem ax2, vai calcular as cargas e a carga máxima entre a terceira e quarta coluna, vai verificar se a solução criada não ultrapasse os 10 Gbps pois, caso ultrapasse, é ignorada e volta-se a gerar uma nova solução até que a carga máxima não ultrapasse os 10 Gbps. Caso contrário, a solução vai ser aceite e calculada a energia como a soma dos comprimentos de todas as ligações que não estão em *spleeping mode*, vai se verificar se a energia é menor que o *best* e, caso seja, guarda-se esse valor (linhas 17 a 47). Caso k_best = 0, quer dizer que no fluxo atual, nenhum dos percursos foi possível ser guardado sem que a carga máxima ultrapasse os 10 Gbps. Caso k_best > 0, quer dizer que, pelo menos, um percurso foi possível guardar que não ultrapasse os 10 Gbps (linhas 41 a 46).

Este processo repete-se para n = 10 e n = 5 caminhos mais curtos. Para cada valor de n, imprimiu-se o valor de consumo de energia da melhor solução, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções (linhas 56 a 58) e, por fim, desenhou-se o gráfico com as melhores energias de todas as soluções geradas para cada simulação de n (linhas 60 a 68).

Result




```
GREEDY RANDOMIZED STRATEGY
   Using all possible routing paths:
      Best energy = 1235.0 Km
      No. of solutions = 3560
      Av. quality of solutions = 1400.3 Km

   Using 10 shortest routing paths:
      Best energy = 1235.0 Km
      No. of solutions = 17608
      Av. quality of solutions = 1413.9 Km

   Using 5 shortest routing paths:
      Best energy = 1303.0 Km
      No. of solutions = 40371
      Av. quality of solutions = 1511.5 Km
```

Conclusions

Relativamente à qualidade das soluções e à melhor energia, quando n = inf, ou seja, quando são utilizados todos os caminhos possíveis na rede para cada fluxo, obtém-se uma eficiência boa, com poucas soluções geradas e com boa qualidade média das soluções.

Quando n = 10, ou seja, quando são utilizados os 10 caminhos mais curtos, obtém-se uma eficiência semelhante à anterior quando n = inf, mas com um aumento bastante significativo no número de soluções geradas e, consequentemente, um aumento na qualidade média das soluções geradas pelo algoritmo.

Quando n = 5, ou seja, quando são utilizados os 5 caminhos mais curtos, a eficiência do algoritmo piora, geram-se mais soluções com pior qualidade e o valor da melhor energia aumenta.

Concluindo, o número de caminhos de roteamento influencia a eficiência do algoritmo *Greedy Randomized*, pois quanto menor for o número de caminhos, maior será a melhor carga da ligação e pior será a qualidade média das soluções, aumentando o número de soluções geradas.

2.c. 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.

Matlab code

```
fprintf('MULTI START HILL CLIMBING STRATEGY\n');
fprintf('   Using all possible routing paths:\n');

n = inf;
[sP nSP] = calculatePaths(L,T,n);

time = 10;

t = tic;
bestEnergy = inf;
allValues = [];
contadortotal = [];

while toc(t) < time
    % 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;

    % Multi start Hill CLimbing:
    continuar = true;
    while continuar
        i_best = 0;
        k_best = 0;
        best = energy;
        for i = 1:nFlows
            for k = 1:nSP(i)
```

```
61                  if k ~= sol(i)
62                      aux = sol(i);
63                      sol(i) = k;
64                      Loads = calculateLinkLoads(nNodes,Links,T,sP,sol);
65                      load1 = max(max(Loads(:,3:4)));
66                      if load1 <= 10
67                          energy1 = 0;
68                          for a = 1:nLinks
69                              if Loads(a,3)+Loads(a,4)>0
70                                  energy1 = energy1 + L(Loads(a,1),Loads(a,2));
71                              end
72                          end
73                      else
74                          energy1 = inf;
75                      end
76                      if energy1 < best
77                          i_best = i;
78                          k_best = k;
79                          best = energy1;
80                      end
81                      sol(i) = aux;
82                  end
83              end
84          end
85          if i_best > 0
86              sol(i_best) = k_best;
87              energy = best;
88          else
89              continuar = false;
90          end
91      end
92      allValues = [allValues energy];
93      if energy < bestEnergy
94          bestSol = sol;
95          bestEnergy = energy;
96      end
97  end
98  fprintf('      Best energy = %.1f Km\n', bestEnergy);
99  fprintf('      No. of solutions = %d\n', length(allValues));
100 fprintf('      Av. quality of solutions = %.1f Km\n\n', mean(allValues));
101
102 figure(3);
103 hold on
104 plot(sort(allValues));
105 ...
106 title({'Multi start Hill Climbing algorithm'}, {'to minimize the energy consumption of the
107 network'});
108 xlabel('No. of solutions');
109 ylabel('Best energy (Km)');
110 legend('All possible routing paths','10 shortest routing paths','5 shortest routing
111 paths','Location','southeast');
```

Code analysis

Primeiramente, definiu-se o número de caminhos a utilizar no algoritmo *Multi Start Hill Climbing* (linha 4), que irá variar entre inf (usando todos os caminhos possíveis na rede), 10 (usando 10 caminhos de roteamento mais curtos) e 5 (usando 5 caminhos de roteamento mais curtos).

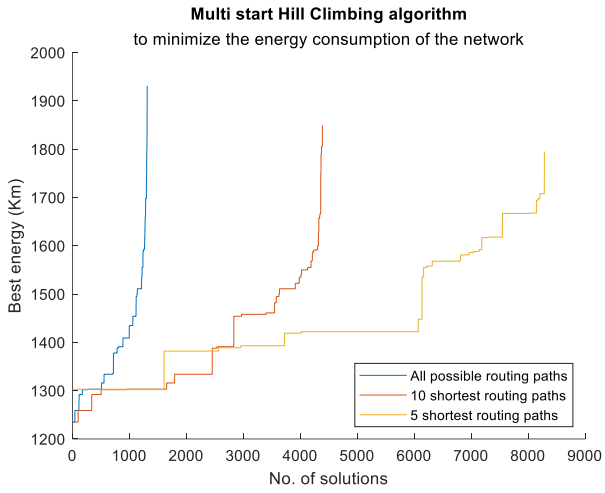
De seguida, calcularam-se os n caminhos da rede para cada fluxo (linha 5), definiu-se o critério de paragem (linha 7) e inicializaram-se algumas variáveis auxiliares (linhas 9 a 12).

Enquanto o tempo não ultrapassa o estipulado (linhas 14 a 97), em primeiro lugar, construiu-se uma solução, usando o algoritmo *Greedy Randomized* (linhas 15 a 51), como implementado na alínea 2.b e, com essa solução, calculou-se uma nova solução aplicando o algoritmo *Hill Climbing*. Já tendo o percurso escolhido para cada fluxo, testaram-se todas as soluções dadas pela troca de um percurso por um outro

para cada fluxo, ou seja, testando todas essas soluções, vê-se qual a melhor e, se for melhor que a solução atual, troca-se e volta-se a repetir até que nenhuma das trocas individuais seja melhor (mínimo local), sempre considerando se a carga é inferior aos 10 Gbps tal como é implementado no algoritmo *Greedy Randomized* (linhas 53 a 91).

Este processo repete-se para n = 10 e n = 5 caminhos mais curtos. Para cada valor de n, imprimiu-se o valor de consumo de energia da melhor solução, o número de soluções geradas pelo algoritmo e a qualidade média de todas as soluções (linhas 98 a 100) e, por fim, desenhou-se o gráfico com as melhores energias de todas as soluções geradas para cada simulação de n (linhas 102 a 111).

Result




```
MULTI START HILL CLIMBING STRATEGY
   Using all possible routing paths:
      Best energy = 1235.0 Km
      No. of solutions = 1312
      Av. quality of solutions = 1375.1 Km

   Using 10 shortest routing paths:
      Best energy = 1235.0 Km
      No. of solutions = 4399
      Av. quality of solutions = 1384.6 Km

   Using 5 shortest routing paths:
      Best energy = 1303.0 Km
      No. of solutions = 8299
      Av. quality of solutions = 1440.7 Km
```

Conclusions

Relativamente à qualidade das soluções e à melhor energia, quando n = inf, ou seja, quando são utilizados todos os caminhos possíveis na rede para cada fluxo, obtém-se uma eficiência bastante boa, com poucas soluções geradas e com boa qualidade média das soluções.

Quando n = 10, ou seja, quando são utilizados os 10 caminhos mais curtos, obtém-se uma eficiência igual à anterior quando n = inf, mas com um aumento significativo no número e na qualidade média das soluções geradas pelo algoritmo.

Quando n = 5, ou seja, quando são utilizados os 5 caminhos mais curtos, a eficiência do algoritmo piora, geram-se mais soluções com pior qualidade e o valor da melhor energia aumenta.

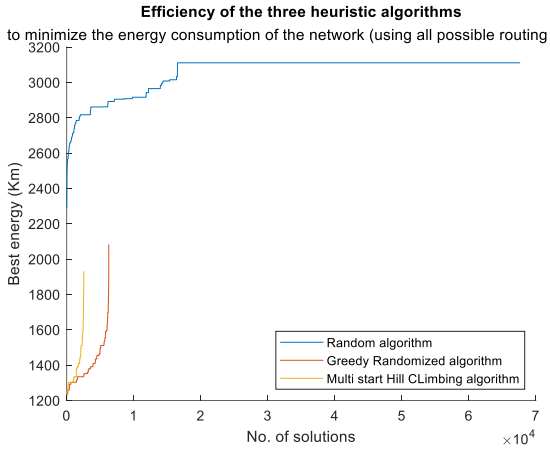
Concluindo, o número de caminhos de roteamento influencia a eficiência do algoritmo *Multi Start Hill Climbing*, pois quanto menor for o número de caminhos, maior será a melhor carga da ligação e pior será a qualidade média das soluções, aumentando o número de soluções geradas.

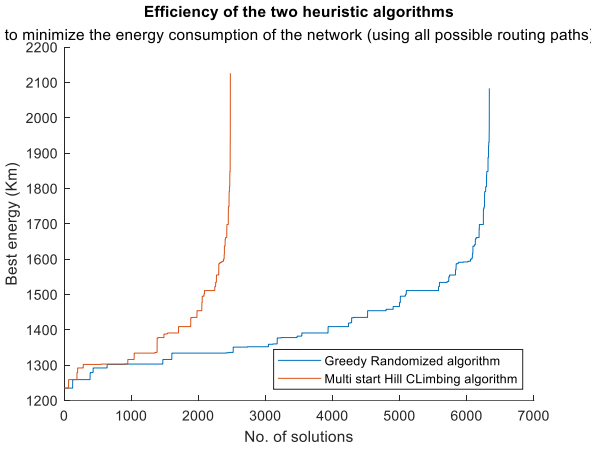
2.d. Compare the efficiency of the three heuristic algorithms based on the results obtained in **2.a**, **2.b** and **2.c**.

Code analysis

Para esta alínea reaproveitaram-se os códigos das alíneas 2.a, 2.b e 2.c, considerando apenas um caso: n = inf (usando todos os caminhos possíveis na rede).

Result






```
RANDOM STRATEGY
   Best energy = 2288.0 Km
   No. of solutions = 148732
   Av. quality of solutions = Inf Km

GREEDY RANDOMIZED STRATEGY
   Best energy = 1235.0 Km
   No. of solutions = 6268
   Av. quality of solutions = 1396.4 Km

MULTI START HILL CLIMBING STRATEGY
   Best energy = 1235.0 Km
   No. of solutions = 2547
   Av. quality of solutions = 1380.2 Km
```

Conclusions

Relativamente ao algoritmo *Random*, é notório que é um algoritmo pouco eficiente quando se pretende minimizar o consumo de energia, uma vez que apresenta um elevado número de soluções geradas, com qualidade média inf e o consumo de energia mínimo ainda continua a estar elevado.

O algoritmo *Greedy Randomized* e o *Multi Start Hill Climbing* variam entre um mínimo e um máximo praticamente igual, a qualidade das soluções geradas é praticamente igual e o valor da melhor energia é igual. Logo, o *Multi Start Hill Climbing* não traz ganhos para a função objetivo pois não consegue minimizar o consumo de energia.

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

Matlab code

```
1  MTBF = (450*365*24)./L;
2  A = MTBF./(MTBF + 24);
3  A(isnan(A))= 0;
4  logA = -log(A);
5
6  [sP nSP] = calculatePaths(logA,T,1);
7
8  Count = 1;
9  Ava = ones(1,length(sP));
10
11 for I = 1:length(sP)
12     fprintf('Flow %d: ',i);
13
14     path = sP{i}{1};
15     aux = 1;
16
17     for j = 1:(length(path)-1)
18         initialNode = path(j);
19         nextNode = path(j+1);
20         ava(i) = ava(i)*A(initialNode,nextNode);
21     end
22     fprintf('availability of Path ');
23     fprintf('%d ', path);
24     fprintf('= %.5f%%\n', ava(i))
25 end
```

Code analysis

Supondo que todos os *routers* tenham disponibilidade muito alta (ou seja, sua disponibilidade é 1,0), calculou-se a disponibilidade de cada *link* como uma matriz quadrada A assumindo o modelo considerado em J.-P. Vasseur, M. Pickavet e P. Demeester, "*Network Recovery: Protection and Restoration of Optical, SONET-DH, IP, and MPLS*", Elsevier (2004) (linhas 1 a 4).

Calcularam-se os caminhos da rede para cada fluxo, usando o primeiro caminho de roteamento mais curto e imprimiu-se, para cada fluxo, um dos seus caminhos de roteamento dados pelo caminho mais disponível, com a respetiva disponibilidade (linhas 11 a 25).

Result

```
Flow 1: availability of Path 1 2 3 = 0.99651%
Flow 2: availability of Path 1 5 4 = 0.99790%
Flow 3: availability of Path 2 4 5 7 = 0.99757%
Flow 4: availability of Path 3 4 = 0.99848%
Flow 5: availability of Path 4 8 9 = 0.99756%
Flow 6: availability of Path 5 7 8 6 = 0.99685%
Flow 7: availability of Path 5 7 8 = 0.99774%
Flow 8: availability of Path 5 7 9 = 0.99850%
Flow 9: availability of Path 6 10 = 0.99942%
```

3.b. 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).

Matlab code

```
1  [sP1 a1 sP2 a2] = calculateDisjointPaths(logA,T);
2  clc
3  for I = 1:nFlows
4      fprintf('Flow %d:\n',i);
5      fprintf('   First path: %d',sP1{i}{1}(1));
6      for j = 2:length(sP1{i}{1})
7          fprintf('-%d',sP1{i}{1}(j));
8      end
9      if ~isempty(sP2{i}{1})
10         fprintf('\n   Second path: %d',sP2{i}{1}(1));
11         for j = 2:length(sP2{i}{1})
12             fprintf('-%d',sP2{i}{1}(j));
13         end
14     end
15     fprintf('\n   Availability of First Path= %.5f%%\n',100*a1(i));
16     if ~isempty(sP2{i}{1})
17         fprintf('   Availability of Second Path= %.5f%%\n',100*a2(i));
18     end
19 end
20 fprintf('\nAverage Service availability of First Path= %.5f%%\n',mean(a1));
21 fprintf('Average Service availability of Second Path= %.5f%%\n',mean(a2));
```

Code analysis

Para cada fluxo, calculou-se outro caminho de roteamento fornecido pelo caminho mais disponível, que é o *link* disjunto com o caminho de roteamento calculado anteriormente, através da função auxiliar *calculateDisjointPaths(logA,T)*, onde a variável sP1 contém o percurso mais disponível, a variável a1 contém a disponibilidade do primeiro percurso, a variável sP2 contém o segundo percurso mais disponível disjunto do primeiro e a variável a2 contém a disponibilidade do segundo percurso, para cada fluxo (linha 1).

De seguida, para cada fluxo, imprimiu-se todos os pares de caminhos de roteamento de cada fluxo e as respetivas disponibilidades (linhas 3 a 19) e, por fim, imprimiu-se a disponibilidade média do serviço, isto é, o valor médio da disponibilidade entre todos os fluxos do serviço (linhas 20 a 21).

Result

```
Flow 1:
   First path: 1-2-3
   Second path: 1-5-4-3
   Availability of First Path= 99.65085%
   Availability of Second Path= 99.63803%
Flow 2:
   First path: 1-5-4
   Second path: 1-2-4
   Availability of First Path= 99.78969%
   Availability of Second Path= 99.73937%
Flow 3:
   First path: 2-4-5-7
   Second path: 2-3-8-7
   Availability of First Path= 99.75688%
   Availability of Second Path= 99.54719%
Flow 4:
   First path: 3-4
   Second path: 3-2-4
   Availability of First Path= 99.84802%
   Availability of Second Path= 99.78606%
Flow 5:
   First path: 4-8-9
   Second path: 4-5-7-9
   Availability of First Path= 99.75631%
   Availability of Second Path= 99.71684%
Flow 6:
   First path: 5-7-8-6
   Second path: 5-4-3-6
   Availability of First Path= 99.68533%
   Availability of Second Path= 99.64530%
Flow 7:
   First path: 5-7-8
   Second path: 5-4-8
   Availability of First Path= 99.77394%
   Availability of Second Path= 99.74175%
Flow 8:
   First path: 5-7-9
   Second path: 5-4-8-9
   Availability of First Path= 99.84980%
   Availability of Second Path= 99.62348%
Flow 9:
   First path: 6-10
   Second path: 6-8-9-10
   Availability of First Path= 99.94159%
   Availability of Second Path= 99.58959%

Average Service availability of First Path= 0.99784%
Average Service availability of Second Path= 0.99670%
```

3.c. 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.

Matlab code

```
1  [sP1 a1 sP2 a2] = calculateDisjointPaths(logA,T);
2  Loads = calculateLinkLoads1plus1(nNodes,Links,T,sP1,sP2);
3  for I = 1:nLinks
4      fprintf('Link [%d %d]:\n',Loads(i,1), Loads(i,2));
5      fprintf('    Bandwidth required in direction %d-%d: %.4f Gbps\n',Loads(i,1), Loads(i,2),
6  Loads(i,3));
7      fprintf('    Bandwidth required in direction %d-%d: %.4f Gbps\n',Loads(i,2), Loads(i,1),
8  Loads(i,4));
9      fprintf('   Total bandwidth required on link [%d %d]: %.4f Gbps\n',Loads(i,2), Loads(i,1),
10 sum(Loads(i,3:4)));
11     if Loads(i,3) > 10
12         fprintf('  The link %d-%d does not have sufficient capacity\n', Loads(i,1), Loads(i,2));
13     end
14     if Loads(i,4) > 10
15         fprintf('  The link %d-%d does not have sufficient capacity\n', Loads(i,2), Loads(i,1));
16     end
17 end
18 fprintf('\nTotal bandwidth required on all links: %.4f Gbps\n', sum(sum(Loads(:,3:4))));
```

Code analysis

Para cada fluxo, calculou-se outro caminho de roteamento fornecido pelo caminho mais disponível, que é o *link* disjunto com o caminho de roteamento calculado anteriormente, através da função auxiliar *calculateDisjointPaths(logA,T)* (linha 1).

De seguida, calculou-se a largura de banda necessária em cada direção de cada *link* para suportar todos os fluxos com proteção 1+1 usando os pares calculados anteriormente de caminhos desconectados de *link*, através da função auxiliar *calculateLinkLoads1plus1(nNodes,Links,T,sP1,sP2)* (linha 2).

Para cada ligação, imprimiu-se a largura de banda em cada direção, a sua largura de banda total (soma das larguras de banda origem-destino e destino-origem), registou-se se a ligação não tem capacidade suficiente, verificando se em cada direção do *link* a largura é superior a 10 (capacidade máxima em cada direção) e, por fim, imprimiu-se a largura de banda total necessária para todos os *links* (linhas 3 a 18).

Result

```
Link [1 2]:
   Bandwidth required on link 1-2: 1.7000 Gbps
   Bandwidth required on link 2-1: 1.5000 Gbps
   Total bandwidth required on link [2 1]: 3.2000 Gbps
Link [1 5]:
   Bandwidth required on link 1-5: 1.7000 Gbps
   Bandwidth required on link 5-1: 1.5000 Gbps
   Total bandwidth required on link [5 1]: 3.2000 Gbps
Link [2 3]:
   Bandwidth required on link 2-3: 5.5000 Gbps
   Bandwidth required on link 3-2: 4.9000 Gbps
   Total bandwidth required on link [3 2]: 10.4000 Gbps
Link [2 4]:
   Bandwidth required on link 2-4: 5.5000 Gbps
   Bandwidth required on link 4-2: 4.1000 Gbps
   Total bandwidth required on link [4 2]: 9.6000 Gbps
Link [3 4]:
   Bandwidth required on link 3-4: 4.9000 Gbps
   Bandwidth required on link 4-3: 4.3000 Gbps
   Total bandwidth required on link [4 3]: 9.2000 Gbps
Link [3 6]:
   Bandwidth required on link 3-6: 1.2000 Gbps
   Bandwidth required on link 6-3: 1.5000 Gbps
   Total bandwidth required on link [6 3]: 2.7000 Gbps
Link [3 8]:
   Bandwidth required on link 3-8: 2.4000 Gbps
   Bandwidth required on link 8-3: 1.5000 Gbps
   Total bandwidth required on link [8 3]: 3.9000 Gbps
Link [7 9]:
   Bandwidth required on link 7-9: 2.6000 Gbps
   Bandwidth required on link 9-7: 4.1000 Gbps
   Total bandwidth required on link [9 7]: 6.7000 Gbps
Link [8 9]:
   Bandwidth required on link 8-9: 4.0000 Gbps
   Bandwidth required on link 9-8: 5.7000 Gbps
   Total bandwidth required on link [9 8]: 9.7000 Gbps
Link [9 10]:
   Bandwidth required on link 9-10: 1.4000 Gbps
   Bandwidth required on link 10-9: 1.6000 Gbps
   Total bandwidth required on link [10 9]: 3.0000 Gbps

Total bandwidth required on all links: 131.8000 Gbps
```

```
Link [4 5]:
   Bandwidth required on link 4-5: 10.8000 Gbps
   Bandwidth required on link 5-4: 10.3000 Gbps
   Total bandwidth required on link [5 4]: 21.1000 Gbps
   The link 4-5 does not have sufficient capacity
   The link 5-4 does not have sufficient capacity
Link [4 8]:
   Bandwidth required on link 4-8: 4.7000 Gbps
   Bandwidth required on link 8-4: 6.6000 Gbps
   Total bandwidth required on link [8 4]: 11.3000 Gbps
Link [5 7]:
   Bandwidth required on link 5-7: 8.3000 Gbps
   Bandwidth required on link 7-5: 9.6000 Gbps
   Total bandwidth required on link [7 5]: 17.9000 Gbps
Link [6 8]:
   Bandwidth required on link 6-8: 2.9000 Gbps
   Bandwidth required on link 8-6: 2.8000 Gbps
   Total bandwidth required on link [8 6]: 5.7000 Gbps
Link [6 10]:
   Bandwidth required on link 6-10: 1.4000 Gbps
   Bandwidth required on link 10-6: 1.6000 Gbps
   Total bandwidth required on link [10 6]: 3.0000 Gbps
Link [7 8]:
   Bandwidth required on link 7-8: 4.8000 Gbps
   Bandwidth required on link 8-7: 6.4000 Gbps
   Total bandwidth required on link [8 7]: 11.2000 Gbps
```

3.d. 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.

Matlab code

```
1  [sP1 a1 sP2 a2] = calculateDisjointPaths(logA,T);
2  Loads = calculateLinkLoads1to1(nNodes,Links,T,sP1,sP2);
3  bestLoad = 0;
4  for I = 1:nLinks
5      fprintf('Link [%d %d]:\n',Loads(i,1), Loads(i,2));
6      fprintf('    Bandwidth required in direction %d-%d: %.4f Gbps\n',Loads(i,1), Loads(i,2),
7  Loads(i,3));
8      fprintf('    Bandwidth required in direction %d-%d: %.4f Gbps\n',Loads(i,2), Loads(i,1),
9  Loads(i,4));
10     fprintf('   Total bandwidth required on link [%d %d]: %.4f Gbps\n',Loads(i,2), Loads(i,1),
11 sum(Loads(i,3:4)));
12     if Loads(i,3) > 10
13         fprintf('   The link %d-%d does not have sufficient capacity\n', Loads(i,1), Loads(i,2));
14     end
15     if Loads(i,4) > 10
16         fprintf('   The link %d-%d does not have sufficient capacity\n', Loads(i,2), Loads(i,1));
17     end
18     if Loads(i,3) > bestLoad
19         bestLoad = Loads(i,3);
20     end
21     if Loads(i,4) > bestLoad
22         bestLoad = Loads(i,4);
23     end
24 end
25 fprintf('\nTotal bandwidth required on all links: %.4f Gbps\n', sum(sum(Loads(:,3:4))));
26 fprintf('The highest bandwidth value required among all links: %.4f Gbps\n', bestLoad);
```

Code analysis

Para cada fluxo, calculou-se outro caminho de roteamento fornecido pelo caminho mais disponível, que é o *link* disjunto com o caminho de roteamento calculado anteriormente, através da função auxiliar *calculateDisjointPaths(logA,T)* (linha 1).

De seguida, calculou-se a largura de banda necessária em cada *link* para dar suporte a todos os fluxos com proteção 1:1 usando os pares calculados anteriormente de caminhos desconectados de *link*, através da função auxiliar *calculateLinkLoads1to1(nNodes,Links,T,sP1,sP2)* (linha 2).

Para cada ligação, imprimiu-se a largura de banda em cada direção, a sua largura de banda total (soma das larguras de banda origem-destino e destino-origem), registou-se se a ligação não tem capacidade suficiente, verificando se em cada direção do *link* a largura é superior a 10 (capacidade máxima em cada direção) e, por fim, imprimiu-se a largura de banda total necessária para todos os *links* e o maior valor de largura de banda necessário entre todos os *links*. (linhas 3 a 26).

Result

```
Link [1 2]:
   Bandwidth required on link 1-2: 1.7000 Gbps
   Bandwidth required on link 2-1: 1.5000 Gbps
   Total bandwidth required on link [2 1]: 3.2000 Gbps
Link [1 5]:
   Bandwidth required on link 1-5: 1.7000 Gbps
   Bandwidth required on link 5-1: 1.5000 Gbps
   Total bandwidth required on link [5 1]: 3.2000 Gbps
Link [2 3]:
   Bandwidth required on link 2-3: 3.4000 Gbps
   Bandwidth required on link 3-2: 3.4000 Gbps
   Total bandwidth required on link [3 2]: 6.8000 Gbps
Link [2 4]:
   Bandwidth required on link 2-4: 4.8000 Gbps
   Bandwidth required on link 4-2: 3.6000 Gbps
   Total bandwidth required on link [4 2]: 8.4000 Gbps
Link [3 4]:
   Bandwidth required on link 3-4: 3.9000 Gbps
   Bandwidth required on link 4-3: 3.3000 Gbps
   Total bandwidth required on link [4 3]: 7.2000 Gbps
Link [3 6]:
   Bandwidth required on link 3-6: 1.2000 Gbps
   Bandwidth required on link 6-3: 1.5000 Gbps
   Total bandwidth required on link [6 3]: 2.7000 Gbps
Link [3 8]:
   Bandwidth required on link 3-8: 2.4000 Gbps
   Bandwidth required on link 8-3: 1.5000 Gbps
   Total bandwidth required on link [8 3]: 3.9000 Gbps
Link [8 9]:
   Bandwidth required on link 8-9: 2.6000 Gbps
   Bandwidth required on link 9-8: 4.1000 Gbps
   Total bandwidth required on link [9 8]: 6.7000 Gbps
Link [9 10]:
   Bandwidth required on link 9-10: 1.4000 Gbps
   Bandwidth required on link 10-9: 1.6000 Gbps
   Total bandwidth required on link [10 9]: 3.0000 Gbps

Total bandwidth required on all links: 113.4000 Gbps

The highest bandwidth value required among all links: 9.6000 Gbps
```

```
Link [4 5]:
   Bandwidth required on link 4-5: 6.9000 Gbps
   Bandwidth required on link 5-4: 5.6000 Gbps
   Total bandwidth required on link [5 4]: 12.5000 Gbps
Link [4 8]:
   Bandwidth required on link 4-8: 4.7000 Gbps
   Bandwidth required on link 8-4: 6.6000 Gbps
   Total bandwidth required on link [8 4]: 11.3000 Gbps
Link [5 7]:
   Bandwidth required on link 5-7: 8.3000 Gbps
   Bandwidth required on link 7-5: 9.6000 Gbps
   Total bandwidth required on link [7 5]: 17.9000 Gbps
Link [6 8]:
   Bandwidth required on link 6-8: 2.9000 Gbps
   Bandwidth required on link 8-6: 2.8000 Gbps
   Total bandwidth required on link [8 6]: 5.7000 Gbps
Link [6 10]:
   Bandwidth required on link 6-10: 1.4000 Gbps
   Bandwidth required on link 10-6: 1.6000 Gbps
   Total bandwidth required on link [10 6]: 3.0000 Gbps
Link [7 8]:
   Bandwidth required on link 7-8: 4.8000 Gbps
   Bandwidth required on link 8-7: 6.4000 Gbps
   Total bandwidth required on link [8 7]: 11.2000 Gbps
Link [7 9]:
   Bandwidth required on link 7-9: 2.6000 Gbps
   Bandwidth required on link 9-7: 4.1000 Gbps
   Total bandwidth required on link [9 7]: 6.7000 Gbps
```

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

Conclusions

É possível verificar que na tarefa 3.c, usando uma proteção de 1+1, é necessário mais 18.4000 Gbps que na tarefa 3.d que usa uma proteção de 1:1. Isto é facilmente explicado tendo em conta que numa proteção de 1:1 os recursos do percurso podem ser partilhados entre diferentes fluxos.

Todos os links com proteção 1:1 têm capacidade suficiente enquanto que com uma proteção de 1+1, o link [4 5] não cumpre com os requisitos da sua capacidade visto o uso exagerado de recursos na rede com esta proteção. Os links [2 3] [2 4] [3 4] [4 5] [8 9] diferem na sua *bandwidth* necessária entre os dois tipos de proteção, necessitando sempre mais recursos numa proteção 1+1 devido ao tráfego neste método ser duplicado pelos dois percursos.

Concluímos então que para se obter um maior aproveitamento e equilíbrio de recursos é mais eficiente utilizar uma proteção de 1:1, onde a partilha de recursos entre os fluxos ajuda no alívio da mesma.

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. 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.

Matlab code

```
1  cost = zeros(nNodes,nFlows);
2
3  for i = 1:nFlows
4      [shortestPath, totalCost] = kShortestPath(L,T(i,1),T(i,2),nNodes);
5      sP{i} = shortestPath;
6      cost(:,i) = totalCost;
7  end
8
9  for i = 1:nFlows
10     for j = 1:nNodes
11         sP1{i}{j} = sP{i}{j};
12         a1(i,j) = exp(-cost(j,i));
13         path1 = sP{i}{j};
14         Laux = L;
15
16         for k = 2:length(path1)
17             Laux(path1(k),path1(k-1)) = inf;
18             Laux(path1(k-1),path1(k)) = inf;
19         end
20
21         [shortestPath, totalCost] = kShortestPath(Laux,T(i,1),T(i,2),1);
22
23         if length(shortestPath) > 0
24             sP2{i}{j} = shortestPath{1};
25             a2(i,j) = exp(-totalCost);
26         else
27             sP2{i}{j} = [];
28             s2(i,j) = 0;
29         end
30     end
31 end
```

Code analysis

Inicialmente são calculados os n percursos mais disponíveis (linhas 3 a 6). Posteriormente para cada fluxo é calculado os custos das ligações para uma matriz de custos disjuntos seguido da obtenção do percurso disjunto ao anterior (linhas 8 a 18). Finalmente são adicionados os segundos caminhos mais perto juntamente com os seus custos (linhas 20 a 25).

AUXILIARY FUNCTIONS

Matlab code

```
function Loads= calculateLinkLoads(nNodes,Links,T,sP,Solution)
    nFlows= size(T,1);
    nLinks= size(Links,1);
    aux= zeros(nNodes);
    for i= 1:nFlows
        if Solution(i)>0
            path= sP{i}{Solution(i)};
            for j=2:length(path)
                aux(path(j-1),path(j))= aux(path(j-1),path(j)) + T(i,3);
                aux(path(j),path(j-1))= aux(path(j),path(j-1)) + T(i,4);
            end
        end
    end
    Loads= [Links zeros(nLinks,2)];
    for i= 1:nLinks
        Loads(i,3)= aux(Loads(i,1),Loads(i,2));
        Loads(i,4)= aux(Loads(i,2),Loads(i,1));
    end
end

function [sP nSP]= calculatePaths(L,T,n)
    nFlows= size(T,1);
    nSP= zeros(1,nFlows);
    for i=1:nFlows
        [shortestPath, totalCost] = kShortestPath(L,T(i,1),T(i,2),n);
        sP{i}= shortestPath;
        nSP(i)= length(totalCost);
    end
end

function Loads= calculateLinkLoads1plus1(nNodes,Links,T,sP1,sP2)
    nFlows= size(T,1);
    nLinks= size(Links,1);
    aux= zeros(nNodes);
    for i= 1:nFlows
        if ~isempty(sP1{i}{1})
            path= sP1{i}{1};
            for j=2:length(path)
                aux(path(j-1),path(j))= aux(path(j-1),path(j)) + T(i,3);
                aux(path(j),path(j-1))= aux(path(j),path(j-1)) + T(i,4);
            end
        end
        if ~isempty(sP2{i}{1})
            path= sP2{i}{1};
            for j=2:length(path)
                aux(path(j-1),path(j))= aux(path(j-1),path(j)) + T(i,3);
                aux(path(j),path(j-1))= aux(path(j),path(j-1)) + T(i,4);
            end
        end
    end
    Loads= [Links zeros(nLinks,2)];
    for i= 1:nLinks
        Loads(i,3)= aux(Loads(i,1),Loads(i,2));
        Loads(i,4)= aux(Loads(i,2),Loads(i,1));
    end
end

function Loads= calculateLinkLoads1to1(nNodes,Links,T,sP1,sP2)
    nFlows= size(T,1);
    nLinks= size(Links,1);
    aux= zeros(nNodes);
    for i= 1:nFlows
        path= sP1{i}{1};
```

```
64          for j=2:length(path)
65              aux(path(j-1),path(j))= aux(path(j-1),path(j)) + T(i,3);
66              aux(path(j),path(j-1))= aux(path(j),path(j-1)) + T(i,4);
67          end
68      end
69      for link= 1:nLinks
70          aux2= zeros(nNodes);
71          t1= Links(link,1);
72          t2= Links(link,2);
73          for i= 1:nFlows
74              path= sP1{i}{1};
75              pathdif= find(path==t1 | path==t2);
76              if length(pathdif)<2 || pathdif(2)-pathdif(1)>1
77                  for j=2:length(path)
78                      aux2(path(j-1),path(j))= aux2(path(j-1),path(j)) + T(i,3);
79                      aux2(path(j),path(j-1))= aux2(path(j),path(j-1)) + T(i,4);
80                  end
81              elseif ~isempty(sP2{i}{1})
82                  path= sP2{i}{1};
83                  for j=2:length(path)
84                      aux2(path(j-1),path(j))= aux2(path(j-1),path(j)) + T(i,3);
85                      aux2(path(j),path(j-1))= aux2(path(j),path(j-1)) + T(i,4);
86                  end
87              end
88          end
89          aux=max(aux,aux2);
90      end
91      Loads= [Links zeros(nLinks,2)];
92      for i= 1:nLinks
93          Loads(i,3)= aux(Loads(i,1),Loads(i,2));
94          Loads(i,4)= aux(Loads(i,2),Loads(i,1));
95      end
96  end
```