

TRAFFIC ENGINEERING OF TELECOMMUNICATION NETWORKS

Universidade de Aveiro

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

Relatório



António Domingues 89007 & Henrique Silva 88857

Professor Amaro de Sousa

20 de Janeiro de 2022

Índice

Tarefa 1	4
1.a	4
Código e Resultados Obtidos	4
Conclusões	4
1.b	5
Código e Resultados Obtidos	5
Conclusões	7
1.c	8
Código e Resultados Obtidos	8
Conclusões	10
1.d	11
Código e Resultados Obtidos	11
Conclusões	13
1.e	14
Comparações	14
Tarefa 2	16
2.a	16
Código e Resultados Obtidos	16
Conclusões	18
2.b	18
Código e Resultados Obtidos	18
Conclusões	20
2.c	21
Código e Resultados Obtidos	21
Conclusões	22
2.d	22
Tarefa 3	23
3.a	23
Código e Resultados Obtidos	23
3.b	24
Código e Resultados Obtidos	24
3.c	26
Código e Resultados Obtidos	26
3.d	27
Código e Resultados Obtidos	27
3.e	28
Conclusões	28
Tarefa 4	29
4.a	29
Código e Resultados Obtidos	29
4.b	31
Código e Resultados Obtidos	31

Tarefa 1



1.a

Código e Resultados Obtidos

Para se obter os diversos *routing paths*, fornecidos pela rede, utilizou-se a função **kShortestPath**, de modo a descobrir os diversos caminhos. Para tal, utilizou-se um valor de n (n primeiros *shortest paths*) bastante elevado, de modo a garantir que se obtém todos os diferentes resultados. Assim, depois de configurar a rede fornecida pelo professor, executou-se o seguinte código:

```
n= inf; %infinitos k/shortest paths (ou seja todos os possíveis)
[sP nSP]= calculatePaths(L,T,n); %calcular os shortestPaths para cada flow
assim como o número total de shortestpaths para cada flow
```

A função **calculatePaths** elabora um conjunto de caminhos mais curtos para cada fluxo, devolvendo as várias possibilidades de caminhos para cada fluxo ou seja o **sP** (em forma de cell arrays). Retorna ainda o número de caminhos encontrados para os diferentes fluxos da rede **nSP**. Depois de executar o código obteve-se os seguintes resultados

 nSP	[32,32,38,24,36,37,25,41,28]
 sP	1x9 cell

Conclusões

Neste exercício são determinados todos os caminhos possíveis(possibilidades) para os vários fluxos usando os vários nós da rede (sem repetição). Para determinar tais valores, é usado o conhecido algoritmo do caixeiro viajante, também conhecido pelo algoritmo de dijkstra.

São exploradas todas as combinações possíveis de caminhos desde o nó inicial até ao nó final. Estas são expostas na forma de cell's por ordem crescente de caminho mais curto. Conclui-se também que o caminho mais curto, não necessita implicitamente de ter menos nós na solução, podendo observar tal acontecimento abrindo uma das cells da variável **sP**.

1.b

Código e Resultados Obtidos

Antes de executar qualquer código MatLab, criaram-se todos os valores e parâmetros para a rede e serviço proposta pelo professor. De seguida calcularam-se todos os *shortestpaths* possíveis(**sP**) para os valores pedidos na descrição das tarefas(**n**):

```
n= inf; %(todos os possíveis)
[sP nSP]= calculatePaths(L,T,n); %calcular os shortestPaths para cada flow
assim como o número total de shortestpaths para cada flow
```

De seguida utilizou-se o seguinte código para simular a estratégia random, fazendo variar o n para os vários valores solicitados:

```
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10      % nº de segundos a percorrer desde o instante de tempo
até à comparação
    % gerar múltiplas soluções random
    for i= 1:nFlows
        sol(i)= randi(nSP(i)); % nSP(i) = seleciona aleat um dos percursos
possiveis do fluxo
    end

    % calcula as cargas destas soluç ao gerada
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    % verificar com o maior valor das cargas entre a 3 e 4 colunas(visto que
são as que possuem os valores)
    load= max(max(Loads(:,3:4)));
    % guardar todos os valores de carga máxima de todas as soluções
    allValues= [allValues load];
    % ficar com a melhor solução de todas
    if load<bestLoad
```

```

        bestSol= sol;
        bestLoad= load;
    end
end

```

Foram adicionados comentários para dar a entender qual o propósito de cada linha de código.

Resultados obtidos relativamente a todas as possibilidades ($n = \text{inf}$, para garantir que se encontram todos os caminhos possíveis) de *routing paths*:

```

RANDOM:
    Best load = 5.40 Gbps
    No. of solutions = 147413
    Av. quality of solutions = 10.32 Gbps

```

Resultados referentes a **10 shortest routing paths**(neste caso, é necessário alterar o n , de inf para 10, de modo a garantir apenas as n primeiras *shortest routing paths* desde o nó início até ao nó destino).

```

n= 10; %10 k/shortest paths
[sP nSP]= calculatePaths(L,T,n); %calcular os shortestPaths para cada flow
assim como o número total de shortestpaths para cada flow

```

```

RANDOM:
    Best load = 4.40 Gbps
    No. of solutions = 146940
    Av. quality of solutions = 8.51 Gbps

```

E por último, os resultados para **5 shortest routing paths**($n = 5$):

```

n= 5; %5 k/shortest paths
[sP nSP]= calculatePaths(L,T,n); %calcular os shortestPaths para cada flow
assim como o número total de shortestpaths para cada flow

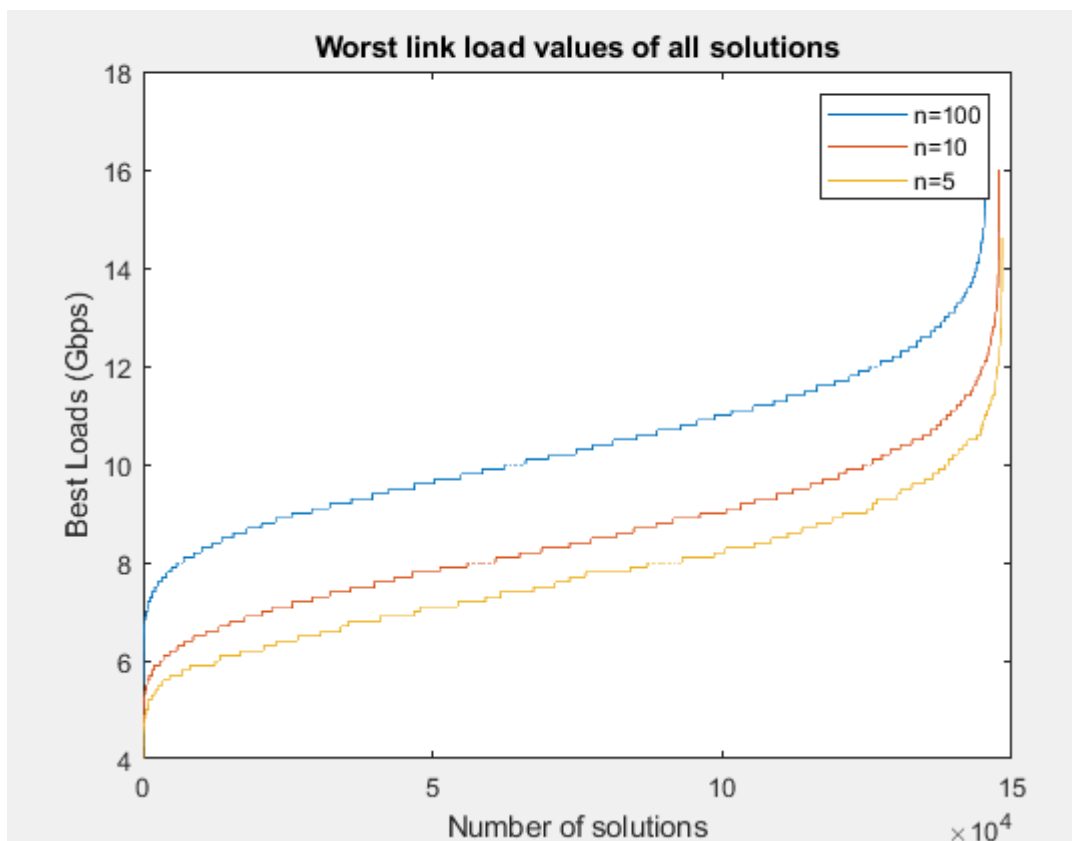
```

```

RANDOM:
    Best load = 4.00 Gbps
    No. of solutions = 148195
    Av. quality of solutions = 7.74 Gbps

```

De seguida é apresentado o gráfico resultante da execução das 3 alíneas:



Conclusões

Ao observar o gráfico acima apresentado, é possível retirar de imediato algumas conclusões. Quanto menor for o número de *routing paths*, menor vai ser a melhor carga, ou seja, quanto menor for o valor de *k_shortest_paths* melhor vai ser a eficiência do algoritmo usado. Em termos de média de qualidade de soluções encontradas, basta olhar para a área do gráfico entre o eixo do x e a linha do algoritmo em questão. Esta mesma área, vai de encontro à alcançada através do MatLab.

Quando se trabalha com total possível de caminhos, obtém-se uma maior média de soluções. Como o algoritmo *random*, seleciona aleatoriamente um dos caminhos possíveis do fluxo, calculando a carga para esse mesmo caminho, pode nem sempre encontrar uma solução melhor, visto que analisa todas as soluções possíveis, daí a média de soluções encontradas também ser maior do que para outros n's.

Quando se reduz o número para 10 *shortest routing paths*, consegue-se encontrar uma solução ligeiramente melhor, pois já não se analisam todas as possíveis, mas sim os 10 caminhos mais curtos, reduzindo também o valor da qualidade média de soluções. Assim surge também a possibilidade de encontrar uma melhor carga, visto que se restringe o intervalo de procura para 10 caminhos mais curtos para cada fluxo.

O mesmo vai acontecer para n= 5, mas agora iterando sobre os 5 *shortest routing paths*. Aqui é reduzido ainda mais o intervalo de procura, conseguindo alcançar uma melhor solução, reduzindo também a qualidade média de soluções.

1.c

Código e Resultados Obtidos

Neste exercício, apenas temos de reformular o algoritmo usado anteriormente, usando o **greedy randomized**. O mesmo é apresentado de seguida, apenas fazendo variar o parâmetro n para cada caso, conforme cada alínea.

```
[sP nSP]= calculatePaths(L,T,n); %calcular os shortestPaths para cada flow
assim como o número total de shortestpaths para cada flow
t= tic;
bestLoad= inf;
sol= zeros(1,nFlows);
allValues= [];
while toc(t)<10 % nº de segundos a percorrer desde o instante de tempo
até à comparacao
    ax2= randperm(nFlows); %vetor aleatório sem repetição de todos os nºs de
flows
    sol= zeros(1,nFlows);
    for i= ax2 % i = ao fluxo selecionado em ax2
        k_best= 0;
        best= inf;
        for k =1:nSP(i) %de 1 até ao nº total de shortest paths do fluxo i
            sol(i) = k; %vamos determinar a solução do fluxo i, para este k
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
            %para cada fluxo ver qual é a carga mais baixa de entre
            %todas as shortest paths daquele fluxo
            if load<best
                k_best= k;
                best= load;
            end
        end
        sol(i)= k_best;
    end
    % calcula as cargas desta solucao gerada
    Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
    % verificar com o maior valor das cargas entre a 3 e 4 colunas
    load= max(max(Loads(:,3:4)));
    % guardar todos os valores de carga maxima de todas as solucoes
    allValues= [allValues load];
    % ficar com a melhor solucao de todas
    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
    end
end
```


end

Enquanto que no algoritmo anterior apenas seleciona aleatoriamente um dos **SP**, para um determinado fluxo calculando a carga dessa solução neste, percorre-se cada fluxo de modo aleatório (através do **randperm**) e calcula-se as cargas para cada percurso desse fluxo. No fim é escolhido, para cada fluxo, o melhor valor.

GREEDY RANDOMIZED:

n= inf

Best load = 3.70 Gbps

No. of solutions = 4311

Av. quality of solutions = 4.53 Gbps

GREEDY RANDOMIZED:

n=10

Best load = 3.70 Gbps

No. of solutions = 15907

Av. quality of solutions = 4.52 Gbps

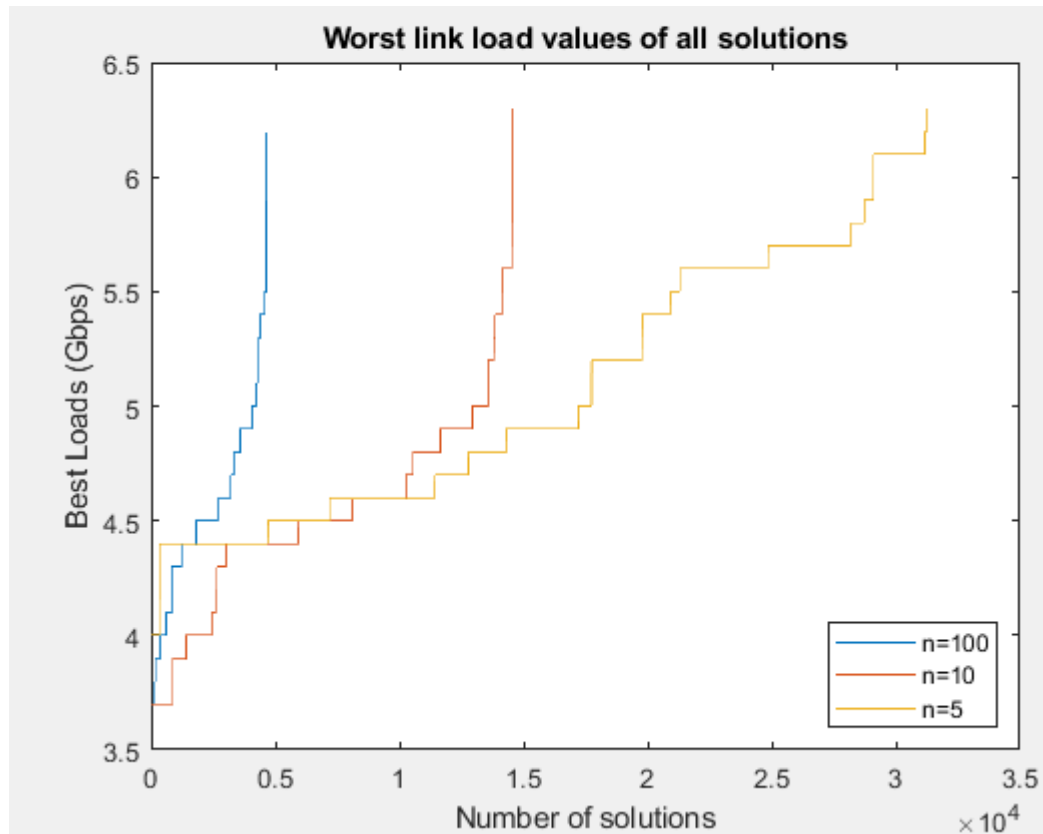
GREEDY RANDOMIZED:

n=5

Best load = 4.00 Gbps

No. of solutions = 30710

Av. quality of solutions = 5.07 Gbps



Conclusões

Nesta alínea os resultados são bastante diferentes. O que sobressai de imediato é o número de soluções para os vários valores de **n**. Para o número máximo de caminhos, o algoritmo percorre cada fluxo. Para cada fluxo vai determinar a melhor carga para todas as possibilidades de **SP** desse fluxo. Como o código é executado apenas durante 10 segundos iterando sobre todas as soluções possíveis de cada fluxo, o número total de soluções para um **n** superior vai ser menor do que um **n** inferior, pois tendo um **n** inferior, pode-se iterar (em 10 segundos) sobre um maior número de soluções.

É de notar ainda, que quanto maior é o valor de possibilidades de caminhos (**n**) maior é o declive. Acontece pelo facto do algoritmo ter de verificar todos os *shortest paths* gerados, tendo menos tempo para gerar mais soluções.

Falta ainda explicar o porquê da melhor carga para os 5 caminhos mais curtos ser pior do que a dos outros valores. Ter o caminho mais curto, não implica ter o caminho com a melhor/menor carga, daí neste caso o valor ser maior do que nos outros valores de **n**.

1.d

Código e Resultados Obtidos

Nesta alínea, efetuou-se o mesmo que nas duas anteriores, apenas alterando o algoritmo a usar(***multi start hill climbing*** neste caso)

Neste algoritmo, é elaborada uma solução inicial (*greedy randomized*) e vai-se de encontro a uma melhor solução, comparando com uma já conhecida. No que diz respeito ao algoritmo, inicialmente é efetuado um *greedy randomized*, e de seguida aplica-se o algoritmo de forma a encontrar uma melhor solução vizinha.

```
%HILL CLIMBING:
continuar= true;
while continuar
    i_best= 0;
    k_best= 0;
    best= load;
    for i= 1:nFlows %Para cada fluxo
        for k= 1:nSP(i) %para todos os caminhos gerados possíveis desse
fluxo
            if k~=sol(i) %not equal à solução do greedy

                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

```

MULTI START HILL CLIMBING:

n = inf

Best load = 3.70 Gbps

No. of solutions = 1663

Av. quality of solutions = 4.29 Gbps

MULTI START HILL CLIMBING:

n = 10

Best load = 3.70 Gbps

No. of solutions = 5731

Av. quality of solutions = 4.27 Gbps

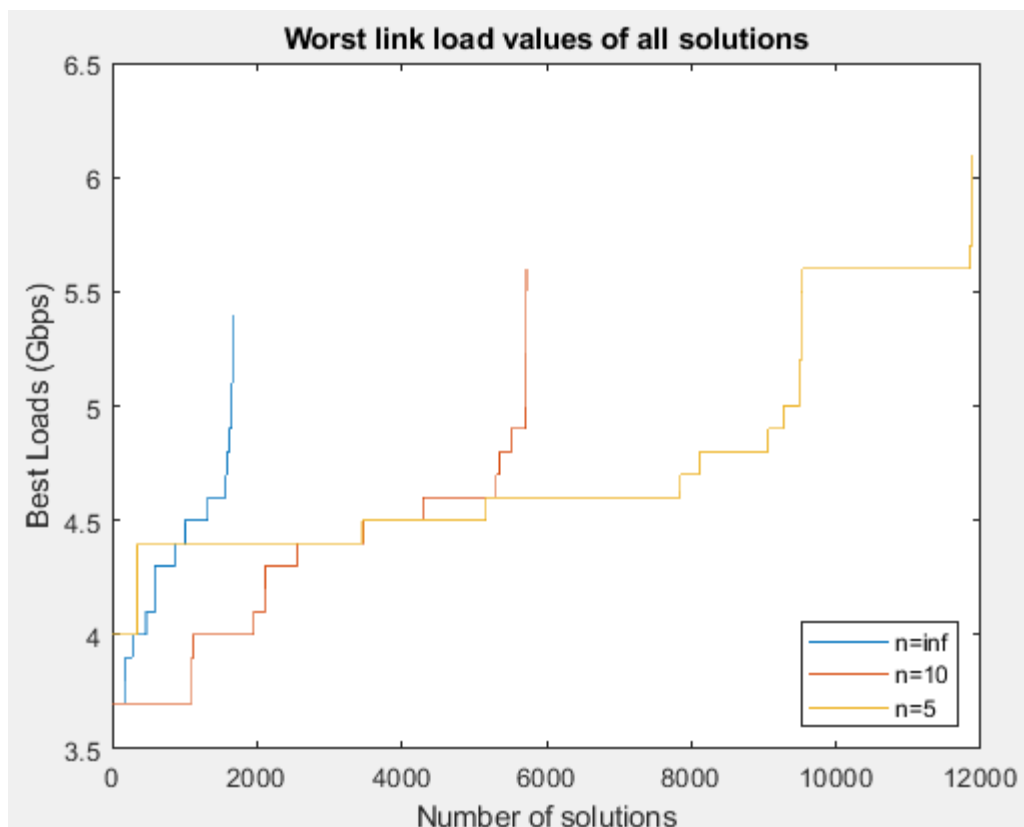
MULTI START HILL CLIMBING:

n = 5

Best load = 4.00 Gbps

No. of solutions = 11882

Av. quality of solutions = 4.75 Gbps



Conclusões

Neste algoritmo aumentar o número de soluções possíveis, de facto, faz com que se encontre a melhor carga diminuindo também o número de soluções encontradas. Ou seja, determinam-se menos soluções obtendo um valor de carga menor. É de notar ainda que ao diminuir no número de soluções possíveis para metade, implica aumentar para o dobro o número total de soluções exploradas, pois dispõe-se de menos caminhos possíveis a iterar no mesmo intervalo de tempo.

<i>Number of routing paths</i>	<i>Worst link load value of best solution</i>	<i>Number of solutions</i>	<i>Average quality of solution</i>
optimization algorithm	random greedy multistart	random greedy multistart	random greedy multistart
inf	5.4 3.7 3.7	147413 4311 1663	10.32 4.53 4.29
10	4.4 3.7 3.7	146940 15907 5731	8.51 4.52 4.27
5	4 4 4	148195 30710 11882	7.74 5.07 4.75

Comparações

Melhor carga: relativamente à melhor solução encontrada, pode-se concluir através dos resultados obtidos que tanto a greedy randomized como a multi start hill climbing encontram os mesmos valores(para o caso de todos os caminhos). O algoritmo random, não chega a encontrar uma solução melhor, pois como descrito anteriormente, este pode nunca iterar sobre uma iteração ótima.

Quando reduzimos o número de caminhos, a melhor solução já é igual em todas. A estratégia greedy e multistart não encontram uma tão boa, pois a melhor solução não tem de estar necessariamente nos 5 caminhos mais curtos. O random encontra valores iguais às outras duas, pelo facto de o intervalo de resultados ser menor e assim conseguir iterar aleatoriamente sobre todas as possibilidades.

Número de soluções: Analisando em detalhe, verifica-se que o algoritmo *multi start hill climbing* utiliza um número menor de soluções, visto que aplica um *greedy randomized*, e pesquisa melhores soluções vizinhas(como explicitado anteriormente). O *greedy randomized* mesmo assim ainda itera sobre um número menor de soluções que o algoritmo *random*. Ainda assim, este analisa mais soluções que o algoritmo *multi start hill climbing*. Como o *random*, tem um carácter aleatório, este pesquisa num maior número de soluções no intervalo de tempo definido.

Qualidade média de soluções: A qualidade média de soluções é melhor para o caso do algoritmo usado na alínea d, pois este encontra uma solução, e apenas procura outra melhor do que a que já tem, descartando inúmeras possibilidades menos boas. O *random* em comparação com o *greedy randomized*, possui qualidade média de soluções pior que o greedy randomized, pelo facto de ser completamente aleatório, repetindo tanto soluções boas como menos boas. O

greedy randomized, já percorre cada fluxo de modo aleatório calculando as cargas para cada percurso desse fluxo. Assim obtêm-se melhores soluções que no algoritmo anterior.

Conclusões:

Como descrito, o *multi start hill climbing* obtém melhores médias de soluções do que todos os outros algoritmos usados. Encontra a mesma melhor solução que o *greedy randomized*, com a vantagem explorar ainda menos soluções.

Assim sendo a melhor maneira de procurar uma solução para suportar o serviço unicast que minimiza a pior carga de link resultante é utilizando o algoritmo *multi start hill climbing*, de seguida o *greedy randomized*, e por último a estratégia *random*.

Tarefa 2

2.a

Código e Resultados Obtidos

O código usado para a realização da task2 é semelhante ao código usado para a realização da task1, existindo apenas algumas diferenças no código dos algoritmos usados.

Na alínea a) é usado o algoritmo *random*. A diferença deste algoritmo para o usado na alínea 1b) é a necessidade de verificar se um link está ativo, isto é, se a soma das suas cargas (em cada direção) é superior a 0, e se a sua carga total não é superior a 10Gbps, visto que essa é a maior carga que um link pode ter. Se ambas as condições se verificarem, então o valor da energia desse link é calculado como a distância entre as duas extremidades. Isto pode ser feito pois é dito no enunciado que o consumo de energia de cada link é proporcional ao seu tamanho. Antes desta operação o valor da energia é sempre reiniciado a 0 para que não seja uma operação cumulativa de solução para solução. Seguidamente o valor calculado para a energia de cada solução é guardado num array de energias (allValues) que é posteriormente organizado por ordem crescente e mostrado num gráfico. Por fim, se a energia calculada for a mais baixa, é guardada na variável bestEnergy.

```
n=inf; %all possible shortest paths

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

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

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
    end
end
```



```

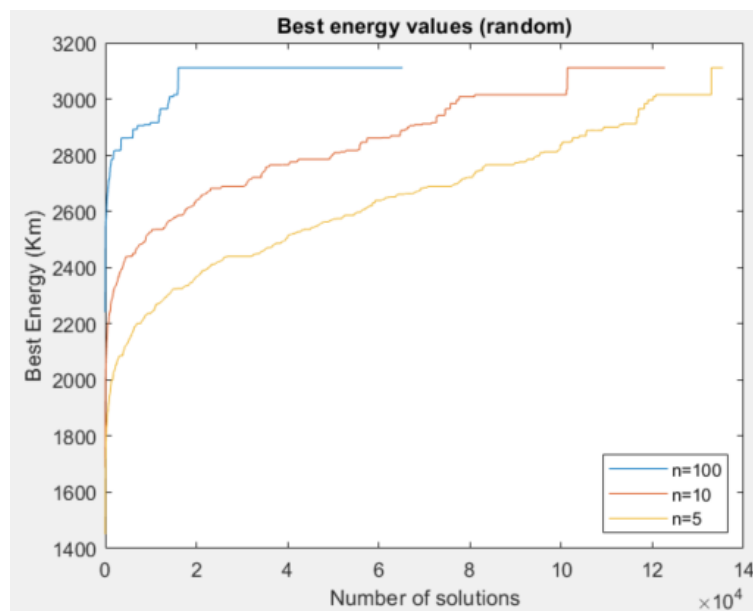
else
    energy= inf;
end

allValues= [allValues energy];

if energy<bestEnergy
    bestSol= sol;
    bestEnergy= energy;
end
end
end

```

Para o cálculo dos valores de carga de cada solução existem 3 variantes. Uso de todos caminhos existentes (n=inf), uso apenas dos 10 mais curtos e o uso dos 5 mais curtos. O gráfico correspondente a cada caso é apresentado de seguida.



Os valores pedidos no enunciado - best energy, número de soluções e o valor médio das soluções - foram os seguintes para cada n:

```

RANDOM(n=inf):
    Best energy = 2241.00 Km
    No. of solutions = 144144
    Av. quality of solutions = Inf Km
RANDOM(n=10):
    Best energy = 1688.00 Km
    No. of solutions = 143687
    Av. quality of solutions = Inf Km
RANDOM(n=5):
    Best energy = 1451.00 Km
    No. of solutions = 144659
    Av. quality of solutions = Inf Km

```

Conclusões

Quanto ao gráfico gerado, é visível que independentemente do valor de n existe sempre um ponto em que o valor das soluções tende para um limite. Isto acontece pois o valor máximo de energia que um percurso pode ter é 3111, correspondente a um percurso que passa por todos os nós. Esse valor corresponde à soma de todas as distâncias existentes. Isto justifica o porquê de a pior solução ser sempre 3111 sendo que a partir daí o valor da solução tende para esse limite.

No entanto, o número de soluções encontradas antes dessa estagnação varia conforme o valor de n , sendo tanto maior quanto menor for o valor de n . Quando é escolhido um $n=\text{inf}$ em ordem a poder usar todos os caminhos gerados, ao escolher um caminho aleatório para cada fluxo é muito provável que seja escolhido um caminho muito grande. Para o caso de $n=5$ o caminho escolhido aleatoriamente é quase sempre mais pequeno pois só pode ser escolhido de entre os 5 caminhos gerados mais curtos. Os caminhos gerados, sendo mais pequenos têm também uma carga mais pequena o que reduz o número de soluções onde a capacidade máxima de carga seja ultrapassada.

O número de soluções geradas é muito semelhante para todos os valores de n usados. No entanto, devido aos fatores explicados acima, quando usamos todos os caminhos gerados para procurar soluções ($n=\text{inf}$), o número de soluções válidas é muito reduzido (cerca de 15mil) quando comparado com $n=5$ (cerca de 130mil). Já para $n=10$ (cerca de 100mil soluções válidas) a tendência é a mesma, sendo muito superior ao número de soluções válidas para $n=\text{inf}$, mas consideravelmente mais pequeno quando comparado com $n=5$.

A solução mais otimizada é encontrada quanto menor for o n pois a margem de gasto de energia é menor, visto que os caminhos disponíveis para escolha são também eles menores. Quanto aos valores médios das soluções tendem sempre para inf pois há soluções que tendem para inf e portanto a sua média é inevitavelmente inf .

2.b

Código e Resultados Obtidos

O código da alínea b) da task2 varia em relação à alínea a) no algoritmo usado. Neste caso é usado o algoritmo greedy randomized. É visitado cada fluxo existente de modo aleatório (através do **randperm**) e são calculadas as cargas para cada percurso desse fluxo. No fim é escolhido, para cada fluxo, o melhor percurso (**k_best**).

```
while toc(t)<10
    ax2= randperm(nFlows); %valores de 1 a 9 organizados aleatoriamente
    sol= zeros(1,nFlows);
    for i= ax2
        k_best= 0;
        best= inf;
        for k =1:nSP(i) % percorre todos os percursos
            sol(i) = k;
            Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
            load= max(max(Loads(:,3:4)));
```

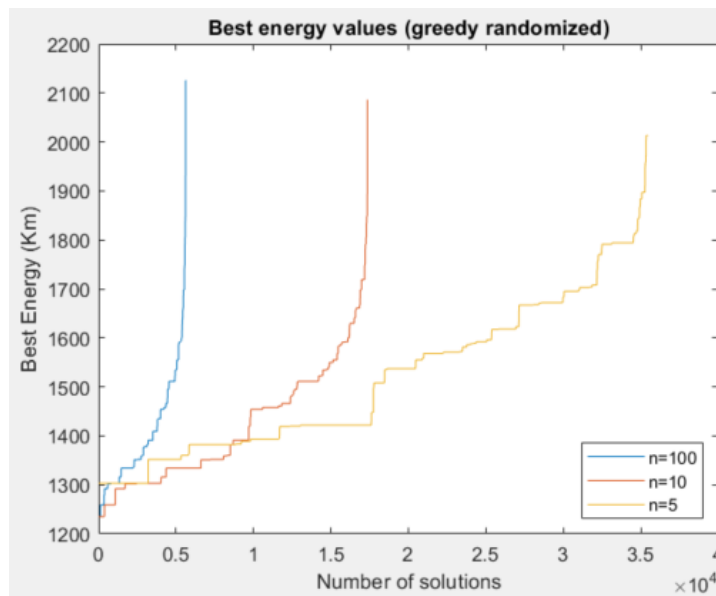
```

        if load <= 10
            energy= 0;
            for a= 1:nLinks
                if Loads(a,3)+Loads(a,4) > 0 %esta a suportar trafego
                    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
% calcula as cargas desta solucao gerada
Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
% verificar o maior valor das cargas entre a 3 e 4 colunas
load= max(max(Loads(:,3:4)));
if load <= 10
    energy= 0;
    for a= 1:nLinks
        if Loads(a,3)+Loads(a,4) > 0 %% está a suportar tráfego
            energy= energy + L(Loads(a,1),Loads(a,2));
        end
    end
else
    energy= inf;
end
allValues= [allValues energy];
% fica com a melhor solucao de todas
if energy<bestEnergy
    bestSol= sol;
    bestEnergy= energy;
end
end
end

```

Mais uma vez foram usados 3 valores para n, resultando no seguinte gráfico.



Os valores pedidos no enunciado - best energy, número de soluções e o valor médio das soluções - foram os seguintes para cada n:

```
Greedy Randomized(n=inf):
    Best energy = 1235.00 Km
    No. of solutions = 5618
    Av. quality of solutions = 1397.39 Km
Greedy Randomized(n=10):
    Best energy = 1235.00 Km
    No. of solutions = 17359
    Av. quality of solutions = 1415.55 Km
Greedy Randomized(n=5):
    Best energy = 1303.00 Km
    No. of solutions = 35460
    Av. quality of solutions = 1513.05 Km
```

Conclusões

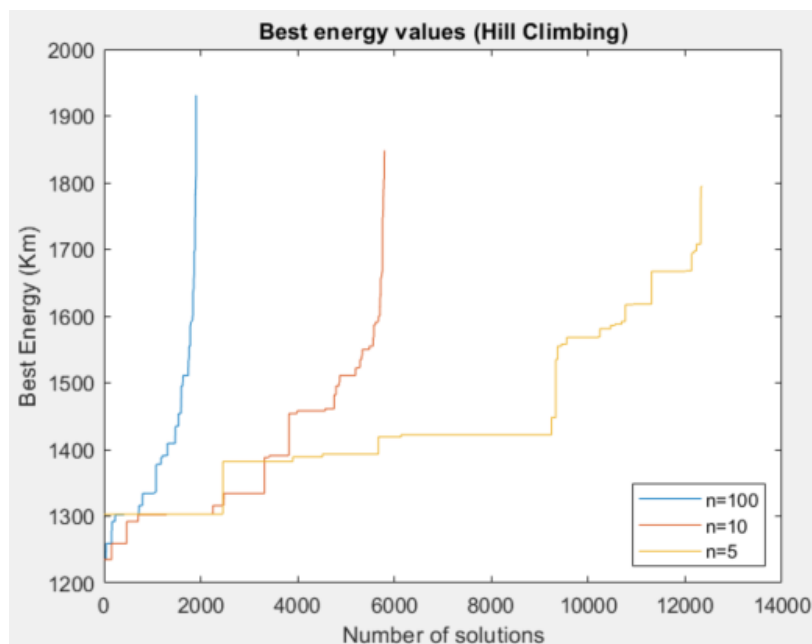
Ao usar o algoritmo greedy randomized os gráficos de cada n têm declives distintos, sendo que quanto maior o n mais acentuado é o declive. Isto acontece pois ao ter que verificar todos os shortest paths gerados, o algoritmo tem menos tempo para gerar mais soluções, resultando num número de soluções mais reduzido. Quando usamos n=5, o algoritmo apenas tem que verificar os 5 caminhos mais curtos para cada fluxo, o que resulta num maior número de soluções visto que o tempo para as gerar é muito inferior do que para um n maior. Contudo, o facto de gerar mais soluções não é sinónimo de gerar as melhores soluções. Isto é visível nos resultados obtidos para o valor de best energy, que foi mais otimizado quando n=inf e n=10. O valor mais reduzido é igual em ambos pois na solução gerada todos os shortest path escolhidos têm um índice inferior a 10, o que significa que independentemente de se ter n=10 ou n=inf obtém-se sempre a melhor solução. No entanto o valor médio das soluções é mais reduzido para n=inf pois ao ter acesso a todos os shortest paths são sempre descobertas as soluções mais otimizadas.

2.c

Código e Resultados Obtidos

O código da alínea c) da task2 é em tudo semelhante aos anteriores, com a diferença de que neste caso é usado o algoritmo hill climbing. Tal como explicado na task anterior, é elaborada uma solução inicial (*greedy randomized*) procurando-se uma melhor solução, ao comparar com uma já conhecida. Em termos de código, o algoritmo é semelhante ao mostrado na task1 sendo a única diferença o facto de ser necessário verificar se a ligação está ativa e se a sua carga não ultrapassa os 10Gbps.

O gráfico relativo aos 3 valores de n é o seguinte.



Os valores pedidos no enunciado - best energy, número de soluções e o valor médio das soluções - foram os seguintes para cada n:

```
Hill Climbing(n=inf):
  Best energy = 1235.00 Km
  No. of solutions = 1908
  Av. quality of solutions = 1375.77 Km
Hill Climbing(n=10):
  Best energy = 1235.00 Km
  No. of solutions = 5804
  Av. quality of solutions = 1380.68 Km
Hill Climbing(n=5):
  Best energy = 1303.00 Km
  No. of solutions = 12358
  Av. quality of solutions = 1437.13 Km
```

Conclusões

Os resultados obtidos com o algoritmo hill climbing são algo semelhantes aos do greedy randomized. Isto acontece porque o hill climbing utiliza greedy randomized para encontrar uma solução, comparando-a com uma solução já existente e utilizando-a apenas se for mais otimizada. Isto significa que são geradas menos soluções, pois é necessário mais tempo para serem geradas e o critério para uma solução ser aceite é mais restrito. No entanto as soluções geradas são mais otimizadas visto que uma solução só é usada se for mais otimizada que as existentes, causando uma qualidade média das soluções melhor do que nos algoritmos anteriores. Quanto maior o n menor o número de soluções geradas pois o acesso a mais shortest paths implica mais tempo para gerar uma solução. Tal como no greedy randomized o $n=\text{inf}$ e $n=10$ obtêm a mesma solução pois conseguem ter acesso a todas as melhores soluções. O mesmo não acontece para o $n=5$ pois como não consegue aceder a todos os caminhos mais curtos existentes, não gera sempre as melhores soluções.

2.d

Conclusões

São visíveis várias diferenças nos resultados obtidos consoante o algoritmo usado. Quanto ao número de soluções geradas é muito superior quando é usado o algoritmo random. Ao usar o algoritmo greedy randomized o número de soluções encontradas diminui significativamente e essa diferença é ainda mais acentuada quando é usado o algoritmo hill climbing. Estas diferenças no número de soluções encontradas acontecem porque o esforço computacional é menor no algoritmo random quando comparado com o greedy randomized e com o hill climbing e porque, neste último, uma solução só é escolhida se for melhor que as existentes.

Relativamente à melhor solução, é encontrada pelo greedy randomized e pelo hill climbing, quando o $n=\text{inf}$ e quando o $n=10$. Isto verifica-se pois ambos têm acesso a todos os shortest paths existentes, guardando apenas o percurso mais otimizado para cada fluxo. Já o algoritmo random acede, sem grande critério e aleatoriamente, aos shortest paths existentes acabando por não encontrar a melhor solução.

O desempenho dos algoritmos influencia também o valor médio das soluções encontradas. Apesar de tanto o hill climbing como o greedy randomized encontrarem ambos o valor mais otimizado, o hill climbing tem a melhor média das soluções. Tal ocorre pois o algoritmo apenas guarda uma solução quando ela é melhor do que as existentes, gerando assim menos soluções e melhorando a sua média.

Resumindo, o algoritmo random permite a descoberta de mais soluções, mas o algoritmo greedy randomized e o hill climbing conseguem descobrir a solução mais otimizada. No entanto, o algoritmo hill climbing obtém uma média de valores melhor devido a um melhor critério na escolha de soluções.

Tarefa 3

3.a

Código e Resultados Obtidos

Para esta pergunta, é pedido que seja calculado o caminho dado pelo *most available path*. Para tal, numa primeira fase, é necessário calcular a disponibilidade de cada link como feito nas aulas práticas, seguido do menos logaritmo dessa disponibilidade (isto pelo valor de $\log(ap)$ ser negativo).

```
MTBF= (450*365*24)./L;  
A= MTBF./(MTBF + 24);  
A(isnan(A))= 0;  
logA = -log(A)*100;  
logA;  
[sP, nSP]= calculatePaths(logA,T,1);  
avail = ones(1,length(sP));  
for i=1:length(sP)  
    path = sP{i}{1};  
    for n=1:(length(path)-1)  
        source = path(n);  
        dest = path(n+1);  
        avail(i) = avail(i) * A(source,dest);  
    end  
    fprintf("For path: ",path);  
    fprintf("%d ",path);  
    fprintf(" Availability is %d \n", avail(i));  
end
```

Neste exercício, ao executar o `calculatePaths` do caminho mais curto usando o **logA** como matriz de custos, obtém-se o caminho mais disponível. Assim sendo, de seguida apenas é necessário calcular o valor de disponibilidade de todo o percurso. Obteve-se os seguintes resultados.

```
Fluxo 1 com path : 1 2 3 Disponibilidade= 9.965085e-01  
Fluxo 2 com path : 1 5 4 Disponibilidade= 9.978969e-01  
Fluxo 3 com path : 2 4 5 7 Disponibilidade= 9.975688e-01  
Fluxo 4 com path : 3 4 Disponibilidade= 9.984802e-01  
Fluxo 5 com path : 4 8 9 Disponibilidade= 9.975631e-01  
Fluxo 6 com path : 5 7 8 6 Disponibilidade= 9.968533e-01  
Fluxo 7 com path : 5 7 8 Disponibilidade= 9.977394e-01  
Fluxo 8 com path : 5 7 9 Disponibilidade= 9.984980e-01  
Fluxo 9 com path : 6 10 Disponibilidade= 9.994159e-01
```

3.b

Código e Resultados Obtidos

Quanto a código, são calculados um par de caminhos disjuntos para cada fluxo através da função fornecida pelo professor. São retornados 2 caminhos e 2 disponibilidades. Um primeiro caminho com um determinado custo, que neste caso corresponde à disponibilidade (que depois é multiplicado por 100 para se obter uma percentagem). Outro caminho **disjunto em links** ao primeiro caminho seguido da sua disponibilidade.

```
[sP1 a1 sP2 a2]= calculateDisjointPaths(Alog,T);
```

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%

Média do primeiro percurso, para todos os flows: 9.978360e-01

Média do segundo percurso, para todos os flows: 9.966973e-01

3.c

Código e Resultados Obtidos

Para a realização da alínea c) da task 3 foi utilizado o código disponibilizado pelo professor. As alterações efetuadas consistem no uso da função **calculateLinkLoads1plus1** para calcular o valor das cargas dos links. A função é chamada após o cálculo dos caminhos disjuntos. Neste mecanismo de proteção (1+1) o fluxo é enviado pelos dois percursos.

```
Loads= calculateLinkLoads1plus1(nNodes,Links,T,sP1,sP2);
```

Os resultados obtidos para a carga total dos links, para os links sem capacidade suficiente e para o maior valor de carga foram os seguintes, respetivamente:

```
totalLoad =
```

```
134.5000
```

```
linksNaosuportam =
```

```
4    5  
5    4
```

Carga do link 1 --> 2	= 1.700000	Carga do link 2 --> 1	= 1.500000
Carga do link 1 --> 5	= 1.700000	Carga do link 5 --> 1	= 1.500000
Carga do link 2 --> 3	= 5.500000	Carga do link 3 --> 2	= 4.900000
Carga do link 2 --> 4	= 5.500000	Carga do link 4 --> 2	= 4.100000
Carga do link 3 --> 4	= 3.400000	Carga do link 4 --> 3	= 3.100000
Carga do link 3 --> 6	= 0.000000	Carga do link 6 --> 3	= 0.000000
Carga do link 3 --> 8	= 2.400000	Carga do link 8 --> 3	= 1.500000
Carga do link 4 --> 5	= 10.800000	Carga do link 5 --> 4	= 10.300000
Carga do link 4 --> 8	= 5.900000	Carga do link 8 --> 4	= 8.100000
Carga do link 5 --> 7	= 8.300000	Carga do link 7 --> 5	= 9.600000
Carga do link 6 --> 8	= 2.900000	Carga do link 8 --> 6	= 2.800000
Carga do link 6 --> 10	= 2.900000	Carga do link 10 --> 6	= 2.800000
Carga do link 7 --> 8	= 3.600000	Carga do link 8 --> 7	= 4.900000
Carga do link 7 --> 9	= 3.800000	Carga do link 9 --> 7	= 5.600000
Carga do link 8 --> 9	= 4.000000	Carga do link 9 --> 8	= 5.700000
Carga do link 9 --> 10	= 2.600000	Carga do link 10 --> 9	= 3.100000

```
Link com maior carga é 4 --> 5 com carga = 10.800000
```

3.d

Código e Resultados Obtidos

À semelhança da alínea anterior, para a realização desta alínea foi utilizado o código disponibilizado pelo professor. As alterações efetuadas consistem no uso da função **calculateLinkLoads1to1**, com base no que é pedido no enunciado. Neste mecanismo de proteção o fluxo é enviado por um dos percursos (percurso de serviço) e o segundo percurso (percurso de proteção) só é usado no caso de o primeiro falhar.

```
Loads= calculateLinkLoads1to1(nNodes,Links,T,sP1,sP2)
```

Os resultados obtidos para a carga total dos links, para os links sem capacidade suficiente e para o maior valor de carga foram os seguintes, respetivamente:

```
totalLoad =
```

```
119.4000
```

```
linksNaosuporam =
```

```
[]
```

Carga do link 1 --> 2	= 1.700000	Carga do link 2 --> 1	= 1.500000
Carga do link 1 --> 5	= 1.700000	Carga do link 5 --> 1	= 1.500000
Carga do link 2 --> 3	= 3.400000	Carga do link 3 --> 2	= 3.400000
Carga do link 2 --> 4	= 4.800000	Carga do link 4 --> 2	= 3.600000
Carga do link 3 --> 4	= 3.400000	Carga do link 4 --> 3	= 3.100000
Carga do link 3 --> 6	= 0.000000	Carga do link 6 --> 3	= 0.000000
Carga do link 3 --> 8	= 2.400000	Carga do link 8 --> 3	= 1.500000
Carga do link 4 --> 5	= 8.800000	Carga do link 5 --> 4	= 7.700000
Carga do link 4 --> 8	= 5.900000	Carga do link 8 --> 4	= 8.100000
Carga do link 5 --> 7	= 8.300000	Carga do link 7 --> 5	= 9.600000
Carga do link 6 --> 8	= 2.900000	Carga do link 8 --> 6	= 2.800000
Carga do link 6 --> 10	= 2.900000	Carga do link 10 --> 6	= 2.800000
Carga do link 7 --> 8	= 3.600000	Carga do link 8 --> 7	= 4.900000
Carga do link 7 --> 9	= 3.800000	Carga do link 9 --> 7	= 5.600000
Carga do link 8 --> 9	= 2.600000	Carga do link 9 --> 8	= 4.100000
Carga do link 9 --> 10	= 1.400000	Carga do link 10 --> 9	= 1.600000

```
Link com maior carga é 7 --> 5 com carga = 9.600000
```

3.e

Conclusões

Como seria de esperar a carga total do sistema usando a proteção 1+1 (134.5) é superior à carga total quando é usada proteção 1:1 (119.4). O aumento da carga é de cerca de 12% quando usamos a proteção 1+1. Este aumento acontece pois na proteção 1+1 o fluxo é enviado pelos 2 percursos disjuntos sendo portanto necessária mais energia. No caso do 1:1 o fluxo apenas é enviado por um dos percursos (percurso de serviço), sendo utilizado o segundo apenas em caso de falha do primeiro.

Isto causa também um aumento do link com mais carga de todo o sistema, sendo 10.8 no mecanismo 1+1 (link 4 → 5) e 9.6 no mecanismo 1:1 (link 7 → 5). Importa também realçar que no caso 1:1 todos os links têm a capacidade de suportar a carga, enquanto que no 1+1 os links 4 → 5 e 5 → 4 excedem os 10gbps de capacidade máxima.

Desta forma o uso de um mecanismo de proteção 1:1 parece ser mais adequado. Os recursos usados são minimizados e não há nenhuma sobrecarga de link. Juntando a isso, como a disponibilidade dos equipamentos é muito elevada, a probabilidade de haver falhas é mais reduzida, o que significa que não seria muito prejudicial demorar um pouco mais a recuperar-se delas.

Tarefa 4

4.a

Código e Resultados Obtidos

Este exercício é semelhante ao anterior, mas neste caso em vez de apenas se calcular um par de percursos distintos para o percurso mais disponível, calcula-se um par de percursos disjuntos para os **10 percursos mais disponíveis**.

Para alcançar tal objetivo, começou por se calcular os 10 percursos mais disponíveis armazenando-os num **cell(sP)** com o respetivo custo desse percurso(**tC**). como se pode verificar no código abaixo:

```
n = 10;
tC= zeros(n,nFlows);
for i=1:nFlows %para calcular os n percursos mais disponíveis
    [shortestPath, totalCost] = kShortestPath(logA,T(i,1),T(i,2),n);
    sP{i}= shortestPath;
    tC(:,i)= totalCost;
end
```

De seguida, itera-se sobre todos os caminhos encontrados para cada fluxo, calculando um percurso disjunto em relação ao percurso que se está a iterar, armazenando-o noutra **cell(sP2)**, guardando também a sua disponibilidade (**exp(-totalCost)**).

O seguinte código, demonstra como foi feito todo o processo acima descrito.

```
for i=1:nFlows % para cada fluxo
    for y=1:n %para os n percurso encontrados do fluxo i

        sP1{i}{y}= sP{i}{y};    %shortestPath
        a1(i,y)= exp(-tC(y,i)); %totalCost
        path1= sP{i}{y};        %shortestPath
        Laux= logA;
        %calcular os custos das ligações para uma matriz de custos disjunta
        for j=2:length(path1)
            Laux(path1(j),path1(j-1))= inf;
            Laux(path1(j-1),path1(j))= inf;
        end
        [shortestPath, totalCost] = kShortestPath(Laux,T(i,1),T(i,2),1); %
```

```

obtenção do percurso disjunto ao anterior
    if length(shortestPath)>0      % se tiver solução quer dizer que
existe
        sP2{i}{y}= shortestPath{1}; %shortestPath
        a2(i,y)= exp(-totalCost);  %totalCost
    else % se não há percurso, é vazio e o custo é zero
        sP2{i}{y}= [];
        s2(i,y)= 0;
    end
end
end

```

4.b

Código e Resultados Obtidos

Nesta alínea foram usados os percursos disjuntos calculados na alínea anterior para executar a otimização. De seguida usou-se o algoritmo de otimização **multi start hill climbing**, para se determinar a melhor solução usando o **calculateLinkLoads1to1**. Posteriormente é guardado o melhor caminho de cada fluxo para a melhor solução, assim como o seu caminho disjunto, para se efetuarem todos os cálculos necessários para a obtenção dos valores pedidos.

```
%cálculo para apresentar o caminho da melhor solução para cada link e average
service availability
for i=1: nFlows
    bestSolPath1{i} = sP1{i}{bestSol(i)};%melhor solução
    avgServiceAvail1(i) = a1(i, bestSol(i));%avg servic avail
    sPcarga1{i}{1} = sP1{i}{bestSol(i)};% path para calcular a highest
bandwith da melhor solução
    bestSolPath2{i} = sP2{i}{bestSol(i)};%melhor solução
    avgServiceAvail2(i) = a2(i, bestSol(i));%avg servic avail
    try
        sPcarga2{i}{1} = sP2{i}{bestSol(i)};%path para calcular a highest
bandwith da melhor solução
    catch exception
        sPcarga2{i}{1} = []; %quando o percurso é vazio rebenta em cima, mas
atribui aqui
    end

end
clc
for i= 1:nFlows
    fprintf('Flow %d:\n',i);
    fprintf('    BEST First path: %d',bestSolPath1{i}(1));
    for j= 2:length(bestSolPath1{i})
        fprintf('-%d',bestSolPath1{i}(j));
    end
    if ~isempty(bestSolPath2{i}) % se tem um link disjoint
        fprintf('\n    BEST Second path: %d',bestSolPath2{i}(1));
        for j= 2:length(bestSolPath2{i})
            fprintf('-%d',bestSolPath2{i}(j));
        end
    end
    end
    %fprintf("\n")
    fprintf('\n    Availability of BEST First Path= %.5f%%\n',100*a1(i,
bestSol(i)));
```

```

        if ~isempty(sP2{i}{1})
            fprintf('    Availability of BEST Second Path= %.5f%%\n',100*a2(i,
bestSol(i)));
        end
    end
    fprintf('\nMédia do MELHOR primeiro percurso, para todos os flows: %.5f%%\n',
mean(avgServiceAvail1)*100);
    fprintf('Média do MELHOR segundo percurso, para todos os flows: %.5f%%\n',
mean(avgServiceAvail2)*100);
    Loads= calculateLinkLoads1to1(nNodes,Links,T,sPcarga1,sPcarga2);
    maiorcarga=0;
    for i=1:length(Loads)
        if Loads(i,3)> maiorcarga
            maiorcarga = Loads(i,3);
            linkDeMaiorCarga= [ Loads(i, 1),  Loads(i, 2)];
        end
        if Loads(i,4) > maiorcarga
            maiorcarga = Loads(i,4);
            linkDeMaiorCarga= [ Loads(i, 2),  Loads(i, 1)];
        end
        fprintf('Carga do link %2d --> %2d    = %f\n', Loads(i, 1), Loads(i, 2),
Loads(i,3));
        fprintf('Carga do link %2d --> %2d    = %f\n', Loads(i, 2), Loads(i, 1),
Loads(i,4));
    end
    fprintf('\nLink com maior carga é %d --> %d com carga = %f\n',
linkDeMaiorCarga(1,1), linkDeMaiorCarga(1,2), maiorcarga);

```

Para este código, obteve-se o seguinte output

```

Flow 1:
    BEST Service path: 1-2-3
    BEST Safety path: 1-5-4-3
    Availability of Service Path= 99.65085%
    Availability of Safety Path= 99.63803%
Flow 2:
    BEST Service path: 1-2-4
    BEST Safety path: 1-5-4
    Availability of Service Path= 99.73937%
    Availability of Safety Path= 99.78969%
Flow 3:
    BEST Service path: 2-4-5-7
    BEST Safety path: 2-3-8-7
    Availability of Service Path= 99.75688%
    Availability of Safety Path= 99.54719%
Flow 4:

```


BEST Service path: 3-4
BEST Safety path: 3-2-4
Availability of Service Path= 99.84802%
Availability of Safety Path= 99.78606%

Flow 5:

BEST Service path: 4-8-7-9
BEST Safety path: 4-3-8-9
Availability of Service Path= 99.59381%
Availability of Safety Path= 99.60652%

Flow 6:

BEST Service path: 5-4-8-6
BEST Safety path: 5-7-9-10-6
Availability of Service Path= 99.65317%
Availability of Safety Path= 99.58835%

Flow 7:

BEST Service path: 5-1-2-3-8
BEST Safety path: 5-7-8
Availability of Service Path= 99.45094%
Availability of Safety Path= 99.77394%

Flow 8:

BEST Service path: 5-7-8-9
BEST Safety path: 5-4-8-6-10-9
Availability of Service Path= 99.65563%
Availability of Safety Path= 99.39224%

Flow 9:

BEST Service path: 6-10
BEST Safety path: 6-8-9-10
Availability of Service Path= 99.94159%
Availability of Safety Path= 99.58959%

Média do MELHOR primeiro percurso, para todos os flows: 99.69892%

Média do MELHOR segundo percurso, para todos os flows: 99.63462%

Carga do link 1 --> 2 = 3.800000
Carga do link 2 --> 1 = 4.000000
Carga do link 1 --> 5 = 3.200000
Carga do link 5 --> 1 = 2.600000
Carga do link 2 --> 3 = 5.500000
Carga do link 3 --> 2 = 5.900000
Carga do link 2 --> 4 = 5.500000
Carga do link 4 --> 2 = 4.100000
Carga do link 3 --> 4 = 4.600000
Carga do link 4 --> 3 = 3.100000
Carga do link 3 --> 6 = 0.000000
Carga do link 6 --> 3 = 0.000000
Carga do link 3 --> 8 = 4.500000
Carga do link 8 --> 3 = 4.700000

```

Carga do link 4 --> 5 = 5.800000
Carga do link 5 --> 4 = 4.400000
Carga do link 4 --> 8 = 3.800000
Carga do link 8 --> 4 = 5.600000
Carga do link 5 --> 7 = 6.100000
Carga do link 7 --> 5 = 5.900000
Carga do link 6 --> 8 = 3.400000
Carga do link 8 --> 6 = 2.800000
Carga do link 6 --> 10 = 3.000000
Carga do link 10 --> 6 = 3.500000
Carga do link 7 --> 8 = 5.900000
Carga do link 8 --> 7 = 5.400000
Carga do link 7 --> 9 = 2.200000
Carga do link 9 --> 7 = 3.700000
Carga do link 8 --> 9 = 3.000000
Carga do link 9 --> 8 = 4.100000
Carga do link 9 --> 10 = 1.900000
Carga do link 10 --> 9 = 1.600000

```

Link com maior carga é 5 --> 7 com carga = 6.100000

Conclusões

Para a otimização, usando **o nosso algoritmo** multi start hill climbing, a solução encontrada não é mais favorável em termos de carga total de rede, obtendo:

```
totalLoad = 133.9000
```

Quanto ao link com maior carga, nesta otimização o valor é bastante melhor, obtendo uma melhor carga de 6.1.

De facto, ao ter mais possibilidades disponíveis para caminhos disjuntos com proteção 1:1, deveria existir a probabilidade de encontrar uma melhor solução. Neste caso, não é o que acontece, pois suspeita-se de um erro no código matlab ao efetuar os cálculos da melhor solução no algoritmo de pesquisa. Contudo, todos os valores pedidos, foram calculados e os mesmos foram acima expostos. É ainda de concluir que a proteção 1:1 é melhor em termos de carga total, pois o tráfego vai apenas numa direção(percurso de serviço e em caso de falha, percurso de proteção). Enquanto que a proteção 1+1, envia o tráfego pelos dois caminhos, sendo que o tráfego é quase duplicado.