Uma imagem com texto

Descrição gerada automaticamente

**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** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58 | 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)));  L(Links(i,1),Links(i,2))= d+5;  L(Links(i,2),Links(i,1))= d+5;  end  L= round(L);  n= inf;  [sP nSP]= calculatePaths(L,T,n);  fprintf('With a k-shortest path algorithm (using the lengths of the links):\n');  for i = 1:nFlows  fprintf(' Flow %d has %d different routing paths provided by the network.\n', i, nSP(i));  end |
| **Code analysis** | |
| Primeiramente, foram definidas três matrizes: nas linhas 3 a 12, 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; nas linhas 13 a 28, a matriz que contém todas as ligações da rede e, nas linhas 29 a 37, a matriz para cada fluxo, segundo a tabela fornecida, 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.  De seguida, foram inicializadas algumas variáveis, de acordo com as matrizes definidas anteriormente, como o número de nós na rede (linha 38), o número de ligações (linha 39), o número de fluxos (linha 40) e os números complexos (linha 41), onde a parte real corresponde à coordenada x e a parte imaginária à coordenada y.  Posteriormente, foi definida 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 42 a 51).  Depois, foi definido quantos caminhos se pretende usar, que, neste caso, o inf corresponde a todos os caminhos possíveis na rede (linha 52). Calculou-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, foram impressos o número total de caminhos para cada fluxo, usando o algoritmo k-shortest path (linhas 55 a 57).  O código das linhas 1 a 51, será utilizado em todas as restantes alíneas da tarefa 1 e na tarefa 2. | |
| **Result** | |
|  | |
| **Conclusions** | |
| **FALTAAA** | |

**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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | fprintf('RANDOM 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;  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)));  allValues = [allValues load];  if load < bestLoad  bestSol = sol;  bestLoad = load;  end  end  fprintf(' Worst 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(1);  hold on  plot(sort(allValues));  ...  title({'Random algorithm'}, {'to minimize the worst link load'});  xlabel('No. of solutions');  ylabel('Best Load (Gbps)');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing paths','Location','northwest'); |
| **Code analysis** | |
| Primeiramente, foi definido o número de caminhos a utilizar no algoritmo Random (linha 3), 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, calculou-se os n caminhos da rede para cada fluxo (linha 4), calcularam-se as cargas das ligações usando o primeiro caminho mais curto de cada fluxo (linhas 5 a 7), definiu-se o critério de paragem (linha 8) e inicializaram-se algumas variáveis auxiliares (linhas 9 a 12).  Enquanto o tempo não ultrapassa o estipulado (linhas 13 a 24), 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, guardaram-se todos os valores de carga máxima de todas as soluções e ficou-se com a melhor solução de todas (linhas 14 a 23).  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 25 a 27) e, por fim, é desenhado um gráfico com as melhores cargas de todas as soluções geradas para cada simulação de n (linhas 28 a 36).  É importante salientar que, quando a função objetivo é minimizar a carga máxima, não é preciso preocupar-se 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 máxima ou, se no fim tiver a solução for superior a 10 Gbps, desde que haja uma solução abaixo dos 10 Gbps, ignora-se todas as soluções acima do 10 Gbps. | |
| **Result** | |
|  | |
|  | |
| **Conclusions** | |
| Pela análise do gráfico e dos valores obtidos pode-se destacar que:   * Quando n=inf, ou seja, quando são utilizados todos os caminhos possíveis na rede para cada fluxo, obtêm-se soluções com baixa qualidade, uma vez que o algoritmo Random seleciona aleatoriamente um dos caminhos possíveis do fluxo, o que faz com que, ao calcular a carga para todos os caminhos possíveis, a qualidade média das soluções seja 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, visto que se restringe mais o intervalo de procura para 5 caminhos mais curtos para cada fluxo.   Concluindo, quanto menor for o número de caminhos de roteamento, menor será a melhor carga da ligação, ou seja, mais eficiente se torna o algoritmo Random. | |

**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** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | 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('Best Load (Gbps)');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing paths','Location','southeast'); |
| **Code analysis** | |
| Primeiramente, foi definido o número de caminhos a utilizar no algoritmo Greedy Randomized (linha 3), 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, calculou-se os n caminhos da rede para cada fluxo (linha 4), calcularam-se as cargas das ligações usando o primeiro caminho mais curto de cada fluxo (linhas 5 a 7), definiu-se o critério de paragem (linha 8) e inicializaram-se algumas variáveis auxiliares (linhas 9 a 11).  Enquanto o tempo não ultrapassa o estipulado (linhas 12 a 35), 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-se as cargas e a carga máxima entre a terceira e quarta coluna e escolhe-se o melhor percurso para o fluxo i (linhas 13 a 34) da função objetivo. É de salientar que o algoritmo implementado é greedy porque sempre que se escolhe um percurso, escolhe-se o melhor e é randomized pois escolhe-se ordem para os fluxos e ao se escolher diferentes ordens, dá diferentes soluções.  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 36 a 38) e, por fim, é desenhado um gráfico com as melhores cargas de todas as soluções geradas para cada simulação de n (linhas 39 a 47). | |
| **Result** | |
|  | |
| **Conclusions** | |
| **FALTAAA**  Tire conclusões sobre a influência do número de caminhos de roteamento na eficiência do algoritmo Greedy randomized. | |

**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** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78 | 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;  load = best;  else  continuar = false;  end  end  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(3);  hold on  plot(sort(allValues));  ...  title({'Multi start Hill CLimbing algorithm'}, {'to minimize the worst link load'});  xlabel('No. of solutions');  ylabel('Best Load (Gbps)');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing paths','Location','southeast'); |
| **Code analysis** | |
| Primeiramente, foi definido o número de caminhos a utilizar no algoritmo Multi Start Hill Climbing (linha 3), 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, calculou-se os n caminhos da rede para cada fluxo (linha 4), calcularam-se as cargas das ligações usando o primeiro caminho mais curto de cada fluxo (linhas 5 a 7), definiu-se o critério de paragem (linha 8) e inicializaram-se algumas variáveis auxiliares (linhas 9 a 12).  Enquanto o tempo não ultrapassa o estipulado (linhas 13 a 66), em primeiro lugar, construiu-se uma solução, usando o algoritmo Greedy Randomized (linhas 14 a 31) e, com essa solução, calculou-se uma nova solução aplicando o algoritmo Hill Climbing. Já tendo o percurso escolhido para cada fluxo e testou-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) (linhas 32 a 65).  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 67 a 69) e, por fim, é desenhado um gráfico com as melhores cargas de todas as soluções geradas para cada simulação de n (linhas 70 a 78). | |
| **Result** | |
|  | |
| **Conclusions** | |
| **FALTAAA**  Tire conclusões sobre a influência do número de caminhos de roteamento na eficiência do algoritmo multi start hill climbing. | |

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

|  |
| --- |
| **Conclusions** |
| **FALTAAA**  Comparar a eficiência dos três algoritmos heurísticos com base nos resultados obtidos em 1.b, 1.ce 1.d |

|  |
| --- |
| **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** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | 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');  ylabel('Best energy (Km)');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing 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.  Ao código da experiência 1.b, 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. | |
| **Result** | |
|  | |
| **Conclusions** | |
| **FALTAAA**  Tire conclusões sobre a influência do número de caminhos de roteamento na eficiência do algoritmo Random. | |

**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** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63 | 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);  hold on  plot(sort(allValues));  ...  title({'Greedy Randomized algorithm'}, {'to minimize the energy consumption of the network'});  xlabel('No. of solutions');  ylabel('Best energy (Km)');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing paths','Location','southeast'); |
| **Code analysis** | |
| Primeiramente, foi definido o número de caminhos a utilizar no algoritmo Greedy Randomized (linha 3), 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, calculou-se os n caminhos da rede para cada fluxo (linha 4), definiu-se o critério de paragem (linha 5) e inicializaram-se algumas variáveis auxiliares (linhas 6 a 8).  Enquanto o tempo não ultrapassa o estipulado (linhas 9 a 51), criou-se um ciclo while para se construir a solução (linhas 11 a 44). 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 13 a 43). 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 37 a 42).  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 52 a 54) e, por fim, é desenhado um gráfico com as melhores energias de todas as soluções geradas para cada simulação de n (linhas 55 a 63). | |
| **Result** | |
|  | |
| **Conclusions** | |
| **FALTAAA**  Tire conclusões sobre a influência do número de caminhos de roteamento na eficiência do algoritmo Greedy randomized. | |

**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** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106 | 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)  if k ~= sol(i)  aux = sol(i);  sol(i) = k;  Loads = calculateLinkLoads(nNodes,Links,T,sP,sol);  load1 = max(max(Loads(:,3:4)));  if load1 <= 10  energy1 = 0;  for a = 1:nLinks  if Loads(a,3)+Loads(a,4)>0  energy1 = energy1 + L(Loads(a,1),Loads(a,2));  end  end  else  energy1 = inf;  end  if energy1 < best  i\_best = i;  k\_best = k;  best = energy1;  end  sol(i) = aux;  end  end  end  if i\_best > 0  sol(i\_best) = k\_best;  energy = best;  else  continuar = false;  end  end  allValues = [allValues energy];  if energy < bestEnergy  bestSol = sol;  bestEnergy = energy;  end  end  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(3);  hold on  plot(sort(allValues));  ...  title({'Multi start Hill Climbing algorithm'}, {'to minimize the energy consumption of the network'});  xlabel('No. of solutions');  ylabel('Best energy (Km)');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing paths','Location','southeast'); |
| **Code analysis** | |
| Primeiramente, foi definido o número de caminhos a utilizar no algoritmo Multi Start Hill Climbing (linha 3), 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, calculou-se os n caminhos da rede para cada fluxo (linha 4), definiu-se o critério de paragem (linha 8) e inicializaram-se algumas variáveis auxiliares (linhas 6 a 9).  Enquanto o tempo não ultrapassa o estipulado (linhas 10 a 93), em primeiro lugar, construiu-se uma solução, usando o algoritmo Greedy Randomized (linhas 11 a 47), 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, testou-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 49 a 87).  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 94 a 96) e, por fim, é desenhado um gráfico com as melhores energias de todas as soluções geradas para cada simulação de n (linhas 97 a 106). | |
| **Result** | |
|  | |
| **Conclusions** | |
| **FALTAAA**  Tire conclusões sobre a influência do número de caminhos de roteamento na eficiência do algoritmo multi start hill climbing. | |

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

|  |
| --- |
| **Conclusions** |
| **FALTAAA**  Comparar a eficiência dos três algoritmos heurísticos com base nos resultados obtidos em 2.a, 2.b e 2.c |

|  |
| --- |
| **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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | 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)));  L(Links(i,1),Links(i,2))= d+5;  L(Links(i,2),Links(i,1))= d+5;  end  L= round(L);  MTBF= (450\*365\*24)./L;  A= MTBF./(MTBF + 24);  A(isnan(A))= 0;  logA= -log(A);  [sP nSP]= calculatePaths(logA,T,1);  count= 1;  ava=ones(1,length(sP));  for i=1:length(sP)  fprintf('Flow %d:\n',i);  path=sP{i}{1};  aux = 1;  for j=1:(length(path)-1)  initialNode = path(j);  nextNode = path(j+1);  ava(i)= ava(i)\*A(initialNode,nextNode);  end    fprintf(' Availability of Path ');  fprintf('%d ', path);  fprintf(' = %.5f%%\n', ava(i))  end |
| **Code analysis** | |
| **FALTAAA** | |
| **Result** | |
|  | |

**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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 |  |
| **Code analysis** | |
| **FALTAAA** | |
| **Result** | |
| **FALTAAA** | |
| **Conclusions** | |
| **FALTAAA** | |

**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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 |  |
| **Code analysis** | |
| **FALTAAA** | |
| **Result** | |
| **FALTAAA** | |
| **Conclusions** | |
| **FALTAAA** | |

**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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 |  |
| **Code analysis** | |
| **FALTAAA** | |
| **Result** | |
| **FALTAAA** | |
| **Conclusions** | |
| **FALTAAA** | |

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

|  |  |
| --- | --- |
| **Matlab code** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 |  |
| **Code analysis** | |
| **FALTAAA** | |
| **Result** | |
| **FALTAAA** | |
| **Conclusions** | |
| **FALTAAA** | |

|  |
| --- |
| **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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 |  |
| **Code analysis** | |
| **FALTAAA** | |
| **Result** | |
| **FALTAAA** | |
| **Conclusions** | |
| **FALTAAA** | |

**4.b.** Develop a multi start hill climbing algorithm for this optimization problem using the 10 pairs of link disjoint paths computed in **4.a** for each flow. Run the algorithm during 30 seconds. Present the pair of routing paths of each flow (and its availability) and the average service availability of the best solution. Present the highest required bandwidth value among all links. Compare this solution with the one in **3.d** and take all possible conclusions.

|  |  |
| --- | --- |
| **Matlab code** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 |  |
| **Code analysis** | |
| **FALTAAA** | |
| **Result** | |
| **FALTAAA** | |
| **Conclusions** | |
| **FALTAAA** | |

|  |
| --- |
| **AUXILIARY FUNCTIONS** |

|  |  |
| --- | --- |
| **Matlab code** | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96 | 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};  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  for link= 1:nLinks  aux2= zeros(nNodes);  t1= Links(link,1);  t2= Links(link,2);  for i= 1:nFlows  path= sP1{i}{1};  pathdif= find(path==t1 | path==t2);  if length(pathdif)<2 || pathdif(2)-pathdif(1)>1  for j=2:length(path)  aux2(path(j-1),path(j))= aux2(path(j-1),path(j)) + T(i,3);  aux2(path(j),path(j-1))= aux2(path(j),path(j-1)) + T(i,4);  end  elseif ~isempty(sP2{i}{1})  path= sP2{i}{1};  for j=2:length(path)  aux2(path(j-1),path(j))= aux2(path(j-1),path(j)) + T(i,3);  aux2(path(j),path(j-1))= aux2(path(j),path(j-1)) + T(i,4);  end  end  end  aux=max(aux,aux2);  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 |