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, nas linhas 3 a 12, foi definida 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, foi definido o vetor que contem todas as ligações da rede. Nas linhas 29 a 37, foi definida a matriz para cada fluxo, segundo a tabela fornecida no enunciado, 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 definidas 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 o número complexo (linha 41), onde a parte real corresponde à coordenada x e a parte imaginária à coordenada y.  Depois, foi definida a 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).  Na linha 52 é definido quantos caminhos se pretende e na linha 53 são calculados 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 todos os caminhos possíveis e em nSP o número total de caminhos. Nas linhas 54 a 58 são apresentados o número total de caminhos para cada fluxo, usando o algoritmo k-shortest path, de modo a obter o resultado final.  É de salientar, que código desta alínea, das linhas 1 a 51, será utilizado em todas as restantes alíneas da tarefa 1. | |
| **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  37  38  39  40  41  42  43 | n = [inf,10,5];  fprintf('RANDOM STRATEGY\n');  for k = 1:length(n)  [sP nSP] = calculatePaths(L,T,n(k));  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  if k == 1  fprintf(' Using all possible routing paths):\n');  elseif k == 2  fprintf(' Using 10 shortest routing paths):\n');  else  fprintf(' Using 5 shortest routing paths):\n');  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(1);  hold on  plot(sort(allValues));  end  title({'Random algorithm'}, {'to minimize the worst link load'});  xlabel('No. of solutions');  ylabel('Quality of solutions (Gbps)');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing paths','Location','northwest'); |
| **Code analysis** | |
| Primeiramente, foi definida uma matriz que contém os três casos de execução, ou seja, usando todos os caminhos possíveis, os 10 caminhos e os 5 caminhos mais curtos (linha 1).  Para cada um dos três casos (linhas 3 a 38), calculou-se os 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).  De seguida, implementou-se o algoritmo de otimização recorrendo à estratégia Random, enquanto o tempo não acaba (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.  Depois, 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. Por fim, desenhou-se o gráfico com os valores de todas as soluções, para cada um dos casos (linhas 35 a 45). | |
| **Result** | |
|  | |
|  | |
| **Conclusions** | |
| De acordo com o gráfico e com dados obtidos pode-se concluir que, usando menos caminhos de roteamento obtêm-se mais soluções com melhor qualidade e que a eficiência do algoritmo é melhor.   * Conclusões sobre a influência do número de caminhos de roteamento na eficiência do 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  48  49  50  51  52  53  54 | n= [inf,10,5];  fprintf('GREEDY RANDOMIZED STRATEGY\n');  for k = 1:length(n)  [sP nSP] = calculatePaths(L,T,n(k));  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  figure(2);  hold on  plot(sort(allValues));  if k == 1  fprintf(' Using all possible routing paths):\n');  elseif k == 2  fprintf(' Using 10 shortest routing paths):\n');  else  fprintf(' Using 5 shortest routing paths):\n');  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));  end  title({'Greedy Randomized algorithm'}, {'to minimize the worst link load'});  xlabel('No. of solutions');  ylabel('Quality of solutions (Gbps)');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing paths','Location','northwest'); |
| **Code analysis** | |
| **FALTAAA** | |
| **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  79  80  81  82  83  84  85 | n = [inf,10,5];  fprintf('MULTI START HILL CLIMBING STRATEGY\n');  for k = 1:length(n)  [sP nSP] = calculatePaths(L,T,n(k));  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  figure(3);  hold on  plot(sort(allValues));  if k == 1  fprintf(' Using all possible routing paths:\n');  elseif k == 2  fprintf(' Using 10 shortest routing paths:\n');  else  fprintf(' Using 5 shortest routing paths:\n');  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));  end  title({'Multi start Hill CLimbing algorithm'}, {'to minimize the worst link load'});  xlabel('No. of solutions');  ylabel('Quality of solutions (Gbps)');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing paths','Location','southeast'); |
| **Code analysis** | |
| **FALTAAA** | |
| **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  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  107 | 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);  n = [inf,10,5];  fprintf('RANDOM STRATEGY\n');  for k = 1:length(n)  [sP nSP] = calculatePaths(L,T,n(k));  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  figure(1);  hold on  plot(sort(allValues));  if k == 1  fprintf(' Using all possible routing paths:\n');  elseif k == 2  fprintf(' Using 10 shortest routing paths:\n');  else  fprintf(' Using 5 shortest routing paths:\n');  end  fprintf(' Best energy = %.1\n', bestEnergy);  fprintf(' No. of solutions = %d\n', length(allValues));  fprintf(' Av. quality of solutions = %.1f\n\n', mean(allValues));  end  title({'Random algorithm'}, {'to minimize the energy consumption of the network'});  xlabel('No. of solutions');  ylabel('Quality of solutions (Gbps)');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing paths','Location','southeast'); |
| **Code analysis** | |
| **FALTAAA** | |
| **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  64  65  66  67  68  69  70 | n = [inf,10,5];  fprintf('GREEDY RANDOMIZED STRATEGY\n');  for k = 1:length(n)  [sP nSP] = calculatePaths(L,T,n(k));  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  figure(2);  hold on  plot(sort(allValues));  if k == 1  fprintf(' Using all possible routing paths:\n');  elseif k == 2  fprintf(' Using 10 shortest routing paths:\n');  else  fprintf(' Using 5 shortest routing paths:\n');  end  fprintf(' Best energy = %.1f\n', bestEnergy);  fprintf(' No. of solutions = %d\n', length(allValues));  fprintf(' Av. quality of solutions = %.1f\n\n', mean(allValues));  end  title({'Greedy Randomized algorithm'}, {'to minimize the energy consumption of the network'});  xlabel('No. of solutions');  ylabel('Quality of solutions');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing paths','Location','southeast'); |
| **Code analysis** | |
| **FALTAAA** | |
| **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  107  108  109  110  111  112 | n = [inf,10,5];  fprintf('MULTI START HILL CLIMBING STRATEGY\n');  for k = 1:length(n)  [sP nSP] = calculatePaths(L,T,n(k));  time = 10;  t = tic;  bestEnergy = inf;  allValues = [];  contadortotal = [];  while toc(t) < time  % Grredy 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  figure(3);  hold on  plot(sort(allValues));  if k == 1  fprintf(' Using all possible routing paths:\n');  elseif k == 2  fprintf(' Using 10 shortest routing paths:\n');  else  fprintf(' Using 5 shortest routing paths:\n');  end  fprintf(' Best energy = %.1f\n', bestEnergy);  fprintf(' No. of solutions = %d\n', length(allValues));  fprintf(' Av. quality of solutions = %.1f\n\n', mean(allValues));  end  title({'Multi start Hill Climbing algorithm'}, {'to minimize the energy consumption of the network'});  xlabel('No. of solutions');  ylabel('Quality of solutions');  legend('All possible routing paths','10 shortest routing paths','5 shortest routing paths','Location','southeast'); |
| **Code analysis** | |
| **FALTAAA** | |
| **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 |