MODELAÇÃO E DESEMPENHO DE REDES E SERVIÇOS

Mini-Projeto 2

Universidade de Aveiro

Bruno Silva (97931)brunosilva16@ua.pt
Marta Oliveira (97613) marta.alex@ua.pt

VERSAO 1

MODELAÇÃO E DESEMPENHO DE REDES E SERVIÇOS

DETI

Universidade de Aveiro

Bruno Silva (97931)brunosilva16@ua.pt
Marta Oliveira (97613) marta.alex@ua.pt

8 de novembro de 2022

Índice

Lista de Figuras

Capítulo 1

Introdução

No âmbito da cadeira Modelação e Desempenho de Redes e Serviços foi nos pedido um relatório que pretende descrever todas as conclusões retiradas durante a resolução deste mini projeto. Este relatório vai ser dividido em 3 capítulos de acordo com as tarefas que nos foram pedidas.

Capítulo 2

Tarefa 1

Tarefa 1 alínea a

Código
De seguida, apresentamos o excerto do código que desenvolvemos para esta alínea.

```
% ex 1.a)
anycast_Nodes = [5 12];
source_Nodes = T_any(:,1)';
nNodes= size(Nodes,1);
[~, paths] = bestCostPaths(L, source_Nodes, anycast_Nodes);

T_anyAux = T_any;
T_any = zeros(length(source_Nodes), 4);
i = 1;
for p = paths
    src = p{1}{1}(1);
    dst = p{1}{1}(end);
    T_any(i, 1) = src;
    T_any(i, 2) = dst;
    T_any(i, 3) = T_anyAux(i, 2);
    T_any(i, 4) = T_anyAux(i, 3);
    i = i + 1;
end

sP= cell(1,length(T_uni));
for f=1:length(T_uni)
    [shortestPath, totalCost] = kShortestPath(L,T_uni(f,1),T_uni(f,2),1);
    sP{f}= shortestPath;
end
```

```
25
26  sP= horzcat(sP, paths);
27  T = [T_uni; T_any];
28
29  sol= ones(1,length(T));
30  Loads= calculateLinkLoads(nNodes,Links,T,sP,sol);
31  % Determine the worst link load:
32  maxLoad= max(max(Loads(:,3:4)));
33
34  fprintf("Ex. 1.a)\n");
35  fprintf("Anycast Nodes = %s\n", num2str(anycast_Nodes));
36  fprintf('Worst link load = %.1f Gbps\n', maxLoad);
37  for i = 1 : length(Loads)
38      fprintf('{%d - %d}:\t%.2f\t%.2f\n', Loads(i),
39      Loads(i+length(Loads)), Loads(i+length(Loads)*2), Loads(i+length(Loads)*3))
40  end
```

De forma a determinar a carga de cada *link* quando todos os fluxos de tráfego de ambos os serviços são encaminhados pelo caminho mais curto começámos por determinar qual dos nós será o nó de *anycast* escolhido para cada um dos nós origem. Para isto, foi necessário desenvolver uma função auxiliar, bestCostPaths, que é apresentada de seguida:

```
1   function [costs, paths] = bestCostPaths(D, source_Nodes, anycast_Nodes)
2       paths = cell(1, length(source_Nodes));
3       costs = zeros(1, length(source_Nodes));
4
5       for i = 1:length(source_Nodes)
6           best_cost = inf;
7
8           for n = 1:length(anycast_Nodes)
9               [path, cost] = kShortestPath(D, source_Nodes(i), anycast_Nodes(n), 1);
10
11              if cost < best_cost
12                  paths{i} = path;
13                  best_cost = cost;
14                  costs(i) = cost;
15              end
16          end
17      end
18
19  end
```

Tendo os nós escolhidos, alterámos a matriz T_any associando a cada nó origem o nó *anycast* escolhido.

Seguidamente, são determinados os percursos mais curtos para cada um dos fluxos *unicast* sendo estes concatenados com os fluxos *anycast* obtendo uma matriz, T, que contém ambos os serviços suportados pela rede.Desta forma, calculamos a carga de cada um dos links, tal como a carga máxima obtida.

Resultados

```
Ex. 1.a)
Anycast Nodes = 5  12
Worst link load = 49.9 Gbps
{1 - 2}:    10.60   8.60
{1 - 5}:    10.30   20.80
{1 - 7}:    3.40    5.60
{2 - 3}:    11.20   11.70
{2 - 4}:    7.30    13.10
{3 - 4}:    49.20   49.60
{3 - 6}:    19.80   21.00
{4 - 5}:    40.60   42.70
{4 - 8}:    33.10   49.20
{4 - 9}:    12.20   13.50
{5 - 7}:    14.70   10.00
{6 - 8}:    0.00    0.00
{6 - 14}:   7.60    14.40
{7 - 9}:    30.50   29.20
{8 - 11}:   20.20   15.20
{8 - 12}:   15.70   49.90
{9 - 10}:   28.90   28.30
{10 - 11}:  19.30   19.40
{11 - 13}:  19.30   19.40
{12 - 13}:  10.90   5.70
{12 - 14}:  21.30   7.10
{13 - 14}:  27.10   25.60
```

Figura 2.1: ex1_a

Tarefa 1 alínea b

Mais uma vez, apresentamos abaixo o excerto do código que desenvolvemos para esta alínea:

```
% ex 1.b)
totalLoads_nodes = zeros(1, nNodes);
energy_nodes = zeros(1, nNodes);

for p = sP
    path = p{1}{1};
    flow_extremes = [path(1), path(end)];
    for i = 1 : length(T)
        if isequal(flow_extremes, T(i, 1:2))
            for n = path
                totalLoads_nodes(n) = totalLoads_nodes(n) + T(i,3) + T(i,4);
            end
            break;
        end
    end
end

for i = 1 : length(totalLoads_nodes)
   energy_nodes(i) = 10 + 90 * (totalLoads_nodes(i) / 500)^2;
end

% Calculate energy consumption of each link (E_l)
sleeping_mode_links = '';
energy_links = zeros(1, length(Loads));
for i = 1 : length(Loads)
    if max(Loads(i, 3:4)) == 0
        sleeping_mode_links = append(sleeping_mode_links, ' {', num2str(Loads(i,1)), ',',
        num2str(Loads(i,2)), '}');
        energy_links(i) = 2;
    else
        energy_links(i) = 6 + 0.2 * L(Loads(i,1), Loads(i,2));
    end
end

total_energy = sum(energy_nodes) + sum(energy_links);
```

Nesta alínea, o cálculo da energia foi dividida em 2 partes. Primeiramente, é calculada a energia consumida por cada um dos nós. Esta tarefa foi executada percorrendo todos os fluxos e para cada um deles somar o *throuput* nos dois sentidos em cada nó que pertence ao percurso de encaminhamento.

Assim conseguimos obter a carga total associada a cada nó, sendo de seguida este valor utilizado para calcular a energia consumida por cada um dos nós da rede.

Posteriormente, é também calculada a energia consumida por cada um dos *links*, determinando também neste processo quais estão em *sleeping mode* pois neste caso, têm a si associados uma carga de 0 nos dois sentidos.

Resultados

```
Ex. 1.b)
Total energy consumption: 851.81
List of links in sleeping mode: {6,8}
```

Figura 2.2: ex1_b

Tarefa 1 alínea c

Nesta tarefa, utilizámos os algoritmos desenvolvidos durante as aulas práticas. No excerto abaixo, temos o *Greedy Randomized strategy*:

```
function [sol,Loads,load] = GreedyRandomizedStrategy(nNodes,Links,T,sP,nSP,anyFlows)
    nFlows = size(T,1); % number of flows
    sol= zeros(1,nFlows);
    randFlows = [anyFlows, randperm(nFlows -9)]; % chooses a random order of flows for the

    % iterate through all flows (in a random order)
    for f = randFlows
        bestLoad = inf;
        %iterate through all path's of the flow
        for path = 1 : nSP(f)
            sol(f) = path;
            % calculate loads
            Loads = calculateLinkLoads(nNodes, Links, T, sP, sol);
            load = max(max(Loads(:, 3:4)));

            % check if the current load is better then bestLoad
            if load < bestLoad
                pathBest = path;    % update the path of the sol
                bestLoad = load;  % update the value of bestLoad
```

```
            end
        end
        sol(f) = pathBest;
    end
    load = bestLoad;
    Loads = calculateLinkLoads(nNodes, Links, T, sP, sol);
end
```

E seguidamente, temos o *Hill Climbing Strategy*:

```
function [sol, load] = HillClimbingStrategy(nNodes, Links, T, sP, nSP, sol, load)
    nFlows = size(T,1);
    % set the best local variables
    bestLocalLoad = load;
    bestLocalSol = sol;

    % Hill Climbing Strategy
    improved = true;
    while improved
        % test each flow
        for flow = 1 : nFlows
            % test each path of the flow
            for path = 1 : nSP(flow)
                if path ~= sol(flow)
                    % change the path for that flow
                    auxSol = sol;
                    auxSol(flow) = path;
                    % calculate loads
                    Loads = calculateLinkLoads(nNodes, Links, T, sP, auxSol);
                    auxLoad = max(max(Loads(:, 3:4)));

                    % check if the current load is better then start load
                    if auxLoad < bestLocalLoad
                        bestLocalLoad = auxLoad;
                        bestLocalSol = auxSol;
                    end
                end
            end
        end

        if bestLocalLoad < load
            load = bestLocalLoad;
            sol = bestLocalSol;
        else
            improved = false;
        end
```

```
    end
end
```

Tarefa 1 alínea d

```
% ex 1.d)
anycast_Nodes = [5 12];
source_Nodes = T_any(:,1)';
nNodes= size(Nodes,1);
[~, paths] = bestCostPaths(L, source_Nodes, anycast_Nodes);

T_anyAux = T_any;
T_any = zeros(length(source_Nodes), 4);

for p = sP
    path = p{1}{1};
    flow_extremes = [path(1), path(end)];
    for i = 1 : length(T)
        if isequal(flow_extremes, T(i, 1:2))
            for n = path
                totalLoads_nodes(n) = totalLoads_nodes(n) + T(i,3) + T(i,4);
            end
            break;
        end
    end
end

k = 2;
sP= cell(1,length(T_uni));
nSP= zeros(1,length(T_uni));
for f=1:length(T_uni)
    [shortestPath, totalCost] = kShortestPath(L,T_uni(f,1),T_uni(f,2),k);
    sP{f}= shortestPath;
    nSP(f)= length(totalCost);
end

T = [T_uni; T_any];
sP= horzcat(sP, paths);
nSP= [nSP ones(1, length(T_any))];
anyFlows = [length(T)-length(T_any)+1:length(T)];

t= tic;
timeLimit= 30;
```

```
bestLoad= inf;
while toc(t) < timeLimit
    [sol, Loads, load] = GreedyRandomizedStrategy(nNodes, Links, T, sP, nSP, anyFlows);

    [sol, load] = HillClimbingStrategy(nNodes, Links, T, sP, nSP, sol, load);

    if load<bestLoad
        bestSol= sol;
        bestLoad= load;
        bestLoads= Loads;
        bestLoadTime = toc(t);
    end
end

fprintf('k-shortest paths: %d\n', k);
fprintf('Worst link load = %.1f Gbps, time = %.2f sec\n', bestLoad, bestLoadTime);
Loads = bestLoads;
totalLoads_nodes = zeros(1, nNodes);
energy_nodes = zeros(1, nNodes);

idx = 1;
for p = sP
    path = p{1}{bestSol(idx)};
    for n = path
        totalLoads_nodes(n) = totalLoads_nodes(n) + T(idx,3) + T(idx,4);
    end
    idx = idx + 1;
end

for i = 1 : length(totalLoads_nodes)
   energy_nodes(i) = 10 + 90 * (totalLoads_nodes(i) / 500)^2;
end

% Calculate energy consumption of each link (E_l)
sleeping_nodes = '';
energy_links = zeros(1, length(Loads));
for i = 1 : length(Loads)
    if max(Loads(i, 3:4)) == 0
        sleeping_nodes = append(sleeping_nodes, ' {', num2str(Loads(i,1)), ',',
        num2str(Loads(i,2)), '}');
        energy_links(i) = 2;
    else
        energy_links(i) = 6 + 0.2 * L(Loads(i,1), Loads(i,2));
    end
end
```

```
84
85  total_energy = sum(energy_nodes) + sum(energy_links);
86
```

Inicialmente, o processo para gerar a matriz, T, é o mesmo que foi feito na alínea a. Posteriormente, é executado o algoritmo Multi Start Hill Climbing com soluções iniciais fornecidas pelo Greedy Randomized, sendo por fim calculada a energia da melhor solução obtida através destes algoritmo utilizando o mesmo método da alínea b.

Resultados e análise

Ao fim de 5 tentativas, o melhor resultado obtido foi:

```
k-shortest paths: 2
No. sol = 24695, Av. W = 40.60
Worst link load = 40.6 Gbps
Total energy consumption: 849.89
List of links in sleeping mode: {6,8}
Time to obtain the best solution: 0.02 sec
```

Figura 2.3: ex1_d

A pior carga dos links, comparativamente à alínea a, foi inferior pois estamos a ultilizar um algoritmo de otimização com o objetivo de minimizar este valor.

A energia consumida foi semelhante à obtida na alínea b dado que foi utilizado um algoritmo que não tem como objetivo melhorar esse valor mas sim o valor da pior carga dos links.

Tarefa 1 alínea e

O script utilizado foi o mesmo da alínea anterior tendo apenas diferido no valor de k que neste caso foi 6.

Resultados e análise

Ao fim de 5 tentativas, o melhor resultado obtido foi:

```
k-shortest paths: 6
No. sol = 5499, Av. W = 40.83
Worst link load = 40.6 Gbps
Total energy consumption: 849.43
List of links in sleeping mode: {6,8}
Time to obtain the best solution: 0.05 sec
```

Figura 2.4: ex1_e

Neste exercício, o resultado que obtivémos foi bastante similar à alinea anterior. Ao considerar um maior número de percursos mais curtos vão haver mais opções disponíveis para serem exploradas, ainda assim visto que o objetivo do algoritmo não passa por melhorar a energia, o valor obtido para a mesma acaba por ser semelhante ao obtido para as alíneas anteriores. O valor da pior carga dos links é igual ao obtido na alínea anterior, concluindo-se portanto que este é o mínimo que se consegue obter.

Tendo em conta que neste algoritmo, como já referido anteriormente, são exploradas mais hipotéses dado que é considerado um maior número de caminhos mais curtos para o encaminhamento dos fluxos unicast. Este facto leva a que o número de soluções obtidas seja inferior ao obtido na alínea anterior.

Capítulo 3

Tarefa 2

Tarefa 2 Alínea a

No script abaixo temos a função GreedyRandomizedStrategy_Energy

```
function [sol, load, Loads, energy] = GreedyRandomizedStrategy_Energy(nNodes,Links,
T,sP,nSP,L,C,anyFlows)
    nFlows = size(T,1); % number of flows
    sol= zeros(1,nFlows);
    randFlows = [anyFlows, randperm(nFlows -9)]; % chooses a random order of flows for the

    % iterate through all flows (in a random order)
    for f = randFlows
        path_index = 0;
        best_load = inf;
        best_Loads = inf;
        best_energy = inf;
        %iterate through all path's of the flow
        for path = 1 : nSP(f)
            sol(f) = path;
            % calculate loads and energy
            [load, Loads, energy] = calculateLinkLoads_Energy(nNodes, Links,
            T, sP, sol, L, C);

            % check if the current load is better then bestLoad
            if energy < best_energy
                path_index = path; % update the path of the sol
                best_load = load;  % update the value of bestLoad
                best_Loads = Loads;
                best_energy = energy;
            end
        end
```

```
            if path_index == 0
                break;
            else
               sol(f) = path_index;
            end
        end

        load = best_load;
        Loads = best_Loads;
        energy = best_energy;
    end
```

No excerto seguinte está demonstrada a função HillClimbingStrategy_Energy:

```
function [sol, load, Loads, energy] = HillClimbingStrategy_Energy(nNodes, Links, T,
sP, nSP, sol, load, Loads, energy, L, C)

    nFlows = size(T,1);
    % set the best local variables
    bestLocalLoad = load;
    bestLocalSol = sol;
    bestLocalLoads = Loads;
    bestLocalEnergy = energy;

    % Hill Climbing Strategy
    improved = true;
    while improved
        % test each path of the flow
        for flow = 1 : nFlows
            % test each path of the flow
            for path = 1 : nSP(flow)
                if path ~= sol(flow)
                    % change the path for that flow
                    aux_sol = sol;
                    aux_sol(flow) = path;
                    % calculate loads and energy
                    [aux_load, aux_Loads, aux_energy] = calculateLinkLoads_Energy(nNodes,
                    Links, T, sP, aux_sol, L, C);

                    % check if the current load is better then start load
                    if aux_energy < bestLocalEnergy
                        bestLocalLoad = aux_load;
                        bestLocalLoads = aux_Loads;
                        bestLocalEnergy = aux_energy;
                        bestLocalSol = aux_sol;
```

```
                    end
                end
            end
        end

        if bestLocalEnergy < energy
            load = bestLocalLoad;
            Loads = bestLocalLoads;
            sol = bestLocalSol;
            energy = bestLocalEnergy;
        else
            improved = false;
        end
    end
end
```

Adicionalmente, mostramos o excerto da função auxiliar, calculateLinkLoads_Energy, que tem como objetivo calcular a carga de cada um dos links tal como a energia associada à solução fornecida.

```
function [load, Loads, energy]= calculateLinkLoads_Energy(nNodes,Links,
T,sP,Solution, L, C)
    nFlows= size(T,1);
    nLinks= size(Links,1);
    aux= zeros(nNodes);
    energy= 0;
    totalLoads_nodes = zeros(1, 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
            % CALCULATE THE ENERGY CONSUMPTION OF THE NODES
            for n = path
                totalLoads_nodes(n) = totalLoads_nodes(n) + T(i,3) + 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));
```

```
25      end
26      load = max(max(Loads(:,3:4)));
27      % If the worst link load is greater than max capacity , energy will be infinite
28      if load > C
29         energy = inf;
30      else
31          for i= 1:nLinks
32              % link in sleeping mode
33              if max(Loads(i, 3:4)) == 0
34                  energy = energy + 2;
35              else
36                  % CALCULATE THE ENERGY CONSUMPTION OF THE LINKS
37                  l = L(Loads(i, 1), Loads(i, 2)); % l = length from nodeA to nodeB
38                  % Capacity of the link is always 50 in this case
39                  energy = energy + 6 + 0.2 * l;
40              end
41          end
42
43          for i = 1 : length(totalLoads_nodes)
44              energy = energy + 10 + 90 * (totalLoads_nodes(i) / 500)^2;
45          end
46      end
47  end
48
```

O algoritmo é bastante similar ao desenvolvido na alínea 1_c, a diferença é que neste, o objetivo não é minimizar a carga máxima dos links mas sim minimizar a energia consumida.

Tarefa 2 Alínea b

O script desenvolvido para esta alínea é semelhante ao utilizado nas alíneas 1_d e 1_e, sendo que o excerto apresentado abaixo apenas ilustra as alterações que foram efetuadas:

```
t= tic;
timeLimit= 30;
best_energy= inf;
contador= 0;
somador= 0;
C = 50;
while toc(t) < timeLimit
    % greedy randomized start
    [sol, load, Loads, energy] = GreedyRandomizedStrategy_Energy(nNodes, Links,
    T, sP, nSP, L, C);

    while energy == inf
        [sol, load, Loads, energy] = GreedyRandomizedStrategy_Energy(nNodes, Links,
        T, sP, nSP, L, C);
    end

    [sol, load, Loads, energy] = HillClimbingStrategy_Energy(nNodes, Links, T, sP, nSP,
    sol, load, Loads, energy, L, C);

    if energy < best_energy
        best_sol= sol;
        best_load= load;
        best_Loads= Loads;
        best_energy= energy;
        bestTime = toc(t);
    end
    contador= contador + 1;
    somador= somador + load;
end


```

As alterações efetuadas, como conseguimos observar, foram atualizar a chamada do algoritmo para o desenvolvido para esta alínea que tem como objetivo, tal como mencionado anteriormente, obter uma solução que minimize a energia consumida.

Resultados e análise

```
k-shortest paths: 2
No. sol = 4467, Av. E = 778.52
Worst link load = 45.6 Gbps
Total energy consumption: 775.64
List of links in sleeping mode: {1,2} {1,7} {2,3}
Time to obtain the best solution: 0.08 sec
```

Figura 3.1: ex2_b

Analisando os resultados e tendo em conta que o objetivo do algortimo não é minimizar a carga máxima dos links conseguimos observar, ainda assim, uma descida deste valor comparativamente ao obtido na alínea 1_a.

Igualmente, a energia consumida, como era pretendido, diminuiu consideravelmente em relação à alínea 1_b.

Estas diferenças são justificadas pelo facto de ser usado um algortimo de otimização em que é pretendido reduzir a energia consumida. A diminuição da carga máxima é observada pois o algoritmo ao tentar encontrar uma solução que minimize a energia está também a distribuir a carga pelos links, o que consequentemente leva a que a carga máxima também diminua.

Tarefa 2 Alínea c

O script utilizado foi o mesmo da alínea anterior tendo apenas diferido no valor de k que neste caso foi 6.

Resultados e análise

```
k-shortest paths: 6
No. sol = 662, Av. E = 748.78
Worst link load = 48.0 Gbps
Total energy consumption: 700.65
List of links in sleeping mode: {1,2} {1,7} {2,3} {4,9} {6,8}
Time to obtain the best solution: 1.26 sec
```

Figura 3.2: ex2_c

Dado que são considerados mais caminhos para o encaminhamento dos fluxos *unicast* o número de alternativas possíveis que são exploradas pelo algoritmo de otimização aumenta o que leva a que seja encontrada uma solução com um valor mais baixo de energia consumida que na alínea anterior, pois existe um maior número de links que se encontram em *sleeping mode*.

É possível observar um aumento no valor da carga máxima dos links em relação ao obtido na alínea anterior, sendo isto causado por existir um maior número de links em *sleeping mode* e, dessa forma, os links operacionais têm que suportar uma maior carga.

Como seria expectável, o número de soluções obtidas também diminuiu em relação à alínea anterior, sendo isto mais uma vez justificado pelo facto de serem considerados mais caminhos para o encaminhamento dos fluxos unicast.

Tarefa 2 Alínea d

Em relação à alínea 1_e a energia consumida é menor pois o algoritmo, nesta tarefa, tem como propósito reduzir a energia consumida da rede.

No entanto, em relação à carga máxima na alínea 1_e a carga é maior pois o objetivo não é a minimizar ao contrário da alínea 2_c.

O número de soluções obtidas na alínea 2_c é inferior pois existe um número elevado de soluções que vão sendo obtidas (pelo Greedy Randomized) que não são consideradas por o seu valor de carga máxima dos links ultrapassar os 50, o que leva a que este algoritmo não consiga obter um número tão elevado de soluções como o da alínea 1_e.
Este fator também influência o tempo que algoritmo leva a encontrar a solução ideal, sendo o valor obtido na oiiiii, chegueiii alínea 2_c superior por esta razão.

Capítulo 4

Tarefa 3

Tarefa 3 Alínea a

Para esta otimização, foi alterada a função *calculateLinkLoads_Energy* de maneira a que esta passasse a aceitar um maxLoad de 100 Gbps e calcule a energia adequada nos links que ultrapassem a capacidade de 50 Gbps. Além disso serão guardados num vetor os links em que a capacidade tenha sido atualizada para 100 Gbps, passando este vetor a fazer parte da lista de elementos que são retornados pela função.

O algoritmo *Multi Start Hill Climbing* com soluções iniciais fornecidas com o algoritmo *Greedy Randomized* utilizado é semelhante ao desenvolvido na tarefa 2, tendo sido apenas alterada a chamada à função que calcula a energia, passando esta a ser feita à nova versão desta função.

De seguida é apresentado o excerto da função *calculateLinkLoads_Energy* que foi alterado em relação à versão anterior:

```
function [load, Loads, energy, Links_C]= calculateLinkLoads_Energy_Ex3(nNodes,
Links,T,sP,Solution, L)
  (... Parte inicial ocultada por não terem sido feitas alterações...)
  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
    load = max(max(Loads(:,3:4)));
    % If the worst link load is greater than max capacity , energy will be infinite
    if load > 100
       energy = inf;
    elseif load < 50
        for i= 1:nLinks
```

```
15              % link in sleeping mode
16              if max(Loads(i, 3:4)) == 0
17                  energy = energy + 2;
18              else
19                  % CALCULATE THE ENERGY CONSUMPTION OF THE LINKS
20                  l = L(Loads(i, 1), Loads(i, 2)); % l = length from nodeA to nodeB
21                  energy = energy + 6 + 0.2 * l;
22              end
23          end
24      else
25          for i= 1:nLinks
26              C = 50;
27              % link in sleeping mode
28              maxLoad = max(Loads(i, 3:4));
29              if  maxLoad == 0
30                  energy = energy + 2;
31              else
32                  % CALCULATE THE ENERGY CONSUMPTION OF THE LINKS
33                  l = L(Loads(i, 1), Loads(i, 2)); % l = length from nodeA to nodeB
34                  if  maxLoad > 50  % The max load cannot be higher than the max capacity
35                      Links_C(i) = 1;
36                      C = 100;
37                  end
38
39                  if C == 50
40                      energy = energy + 6 + 0.2 * l;
41                  else
42                      energy = energy + 8 + 0.3 * l;
43                  end
44              end
45          end
46      end
47  end
48
```

Tarefa 3 Alínea b

O script utilizado nesta alínea foi semelhante ao da alínea 2c, tendo apenas sido alterada a chamada das funções do Greedy Randomized e do Multi Start Hill Climbing para as versões que utilizam a nova versão atualizada da função de cálculo de energia que foi apresentada na alínea anterior.

Resultados e análise

Correndo o algoritmo durante 60 segundos e com k=6 obtemos os seguintes resultados:

```
k-shortest paths: 6
No. sol = 2017, Av. E = 700.56
Worst link load = 80.3 Gbps
Total energy consumption: 678.80
List of links in sleeping mode: {1,2} {1,7} {2,3} {6,8} {10,11} {11,13} {13,14}
List of links updated to a capacity of 100 Gbps: {3,4}
Time to obtain the best solution: 0.64 sec
```

Figura 4.1: ex3_b

Como seria expectável a pior carga dos links aumentou dado que, nesta alínea, passaram a ser aceites soluções com uma carga máxima limite de 100 Gbps.

Tendo em conta que existe a possibilidade de os links passarem de uma capacidade de 50 para 100 Gbps, consequentemente, isto fará com que os outros links suportem uma carga menor o que por sua vez levará a que o número de links em sleeping mode seja maior.

Embora os links com uma capacidade de 100 Gbps consumirem mais energia, o maior número de links em sleeping mode acaba por compensar este fator e fazer com que a energia consumida final seja inferior à obtida na alínea 2c.

Concluímos também que foi possível obter um maior número de soluções em comparação com a alínea 2c. Apesar do tempo de execução ser 2x superior, o aumento do valor do número de soluções excede essa proporção. Isto deveu-se ao aumento do limite da carga máxima de uma solução, como resultado são aceites um maior número de soluções que são calculadas, não sendo "perdido"tanto tempo a recalcular uma solução à custa da anterior não ser aceite por exceder o limite de carga máxima imposto.

Tarefa 3 Alínea c

O script apresentado de seguida foi baseado na alínea anterior, sendo que a principal alteração implementada foi o cálculo de todas as combinações de nós dentro da lista dos possíveis nós anycast dada, sendo de seguida percorridas todas estas combinações de nós.

Deste modo, para cada uma destas combinações os nós anycast e os nós origem são atualizados de acordo com os nós anycast que pertençam à combinação considerada em cada uma das iterações.

```
% ex 3c
k = 6;
sP= cell(1,length(T_uni));
nSP= zeros(1,length(T_uni));
for f=1:length(T_uni)
    [shortestPath, totalCost] = kShortestPath(L,T_uni(f,1),T_uni(f,2),k);
    sP{f}= shortestPath;
    nSP(f)= length(totalCost);
end

nNodes= size(Nodes,1);
possible_any_Nodes = [4 5 6 12 13];
combs_any_nodes = nchoosek(possible_any_Nodes, 2);      % all combinations of anycastNodes
bestGlobalEnergy= inf;
T_any_Original= T_any;
sP_orig = sP;
nSP_orig = nSP;
for c = 1 : length(combs_any_nodes)
    anycast_Nodes = combs_any_nodes(c, :);
    source_Nodes = setdiff(T_any_Original(:,1)', anycast_Nodes);
%     source_Nodes = T_any(:,1)';

    [~, paths] = bestCostPaths(L, source_Nodes, anycast_Nodes);

    T_anyAux = T_any_Original;
    T_any = zeros(length(source_Nodes), 4);
    i = 1;
    for p = paths
        src = p{1}{1}(1);
        dst = p{1}{1}(end);
        T_any(i, 1) = src;
        T_any(i, 2) = dst;
        T_any(i, 3) = T_anyAux(i, 2);
        T_any(i, 4) = T_anyAux(i, 3);
        i = i + 1;
```

```
36        end
37
38        for n = T_any_Original
39           if ismember(n,anycast_Nodes)
40                path = {[n, n]};
41                paths= horzcat(paths, path);
42           end
43        end
44        T = [T_uni; T_any];
45        sP= horzcat(sP_orig, paths);
46        nSP= [nSP_orig ones(1, length(paths))];
47        anyFlows = [length(T)-length(T_any)+1:length(T)];
48
49        t= tic;
50        timeLimit= 60;
51        bestLocalEnergy= inf;
52        contador= 0;
53        somador= 0;
54        % Links_C= zeros(1,length(Links));
55        while toc(t) < timeLimit
56            % greedy randomized start
57            [sol, load, Loads, energy, Links_C] = GreedyRandomizedStrategy_Energy_Ex3(nNodes, L
58
59            while energy == inf
60                [sol, load, Loads, energy, Links_C] = GreedyRandomizedStrategy_Energy_Ex3(nNode
61            end
62
63            [sol, load, Loads, energy, Links_C] = HillClimbingStrategy_Energy_Ex3(nNodes, Links
64
65            if energy < bestLocalEnergy
66                aux_best_sol= sol;
67                aux_best_load= load;
68                aux_best_Loads= Loads;
69                aux_bestTime = toc(t);
70                aux_best_Links_C = Links_C;
71                bestLocalEnergy = energy;
72            end
73            contador= contador + 1;
74            somador= somador + load;
75        end
76
77        if bestLocalEnergy < bestGlobalEnergy
78            best_Anycast_Nodes = anycast_Nodes;
79            best_sol= aux_best_sol;
80            best_load= aux_best_load;
```

```
81          best_Loads= aux_best_Loads;
82          bestTime = aux_bestTime;
83          best_Links_C = aux_best_Links_C;
84          bestGlobalEnergy = bestLocalEnergy;
85      end
86  end
87
```

Resultados e Análise

```
k-shortest paths: 6
Best Anycast Nodes: 4  13
Worst link load = 49.4 Gbps
Total energy consumption: 653.28
List of links in sleeping mode: {1,2} {1,7} {2,3} {3,6} {8,11} {8,12} {12,13} {12,14}
List of links updated to a capacity of 100 Gbps:
Time to obtain the best solution: 4.91 sec
```

Figura 4.2: ex3_c

Nesta alínea, como observamos na captura apresentada, através da exploração das diferentes combinações possíveis de nós anycast foi possível obter uma solução que minimizou tanto a pior carga dos links como a energia total consumida.

Isto foi possível através da utilização do par de nós anycast (4, 13) que levou a que houvesse um maior número de links em sleeping mode e nenhum link com uma capacidade de 100Gbps.