MDRS Project - 2^{nd} Project

Modelação e Desempenho de Redes e Serviços

Universidade de Aveiro

Tiago Pedrosa (93389), Lucas Pinto (98500)
pedrosa.tiago@ua.pt, pinto.lucas@ua.pt

December 28, 2023

Contents

Chapter 1

Task 1

1.1 Task 1.a)

The feasibility of routing all flows through the path with the minimum propagation delay depends on the capacity constraints of routers and links. Simply choosing the shortest path based on propagation delay will likely not guarantee a feasible solution due to the fact that router throughput is limited to 1 Tbps and link capacity is limited to 100 Gbps. We are essentially choosing the shortest path for each flow based on the physical distance, but if the limitations are exceeded, these link will be overloaded and very negatively impact the feasibility of the network, and additional routing algorithms and capacity checks would be needed.

1.2 Task 1.b)

In this Task, we address the optimization problem of computing a symmetrical single path routing solution to support multiple services, aiming to minimize the resulting worst link load. For this end, we used a Multi-Start Hill Climbing algorithm, incorporating initial Greedy Randomized solutions and a stopping criterion of 60 seconds.

1.2.1 Input Data

```
load('InputDataProject2.mat')
T = [T1; T2];
nFlows = size(T,1);
nNodes = size(Nodes,1);
v = 2e5;    % speed of light on fibers
D = L/v;    % propagation delay on each direction of each link
Link_cap = 100;     % Link capacity in Gbps
Node_cap = 1000;    % Router throughput capacity in Gbps
k = 2;

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

1.2.2 Implementing the algorithms

```
timeLimit = 60; bestLoad = inf; contador = 0; somador = 0;
maxLoad = inf; LinkEne = 0; t = tic;

while toc(t) < timeLimit
    % GreedyRandomized start (Load must be valid)
    while maxLoad > Link_cap
        [sol, maxLoad, Loads, Linkenergy] = GreedyRandomizedLOAD(nNodes,Links,T,sP,nSP,
                                            L,Link_cap);
    end

    % HillClimbing
    [sol, maxLoad, Loads, Linkenergy] = HillClimbingLOAD(nNodes,Links,T,sP,nSP,
                                        sol,Loads,Linkenergy,L,Link_cap);

    % prioritize lowering the worst link load
    if maxLoad<bestLoad
        bestSol= sol;
        bestLoad= maxLoad;
        bestLoadTime = toc(t);
        LinkEne = LinkEne + Linkenergy;
```

```
        % Calculate round-trip delay for each flow in the best solution
        roundTripDelayService1 = zeros(12, 1); roundTripDelayService2 = zeros(8, 1);

        for f = 1:nFlows
            if sol(f) ~= 0
                pathNodes = sP{f}{sol(f)};
                totalPathDelay = 0;
                for i = 1:(length(pathNodes)-1)
                    linkIndex = find((Links(:,1) == pathNodes(i)
                                    & Links(:,2) == pathNodes(i+1))
                                    | (Links(:,1) == pathNodes(i+1)
                                    & Links(:,2) == pathNodes(i)));
                    totalPathDelay = totalPathDelay + D(pathNodes(i),pathNodes(i+1));
                end
                roundTripDelay = 2 * totalPathDelay * 1000;  % Round-trip delay in ms
                if f <= 12 % divide the services
                    roundTripDelayService1(f) = roundTripDelay;
                else
                    roundTripDelayService2(f) = roundTripDelay;
                end

            end
        end
    end
    contador= contador + 1;
    somador= somador + maxLoad;

end
```

1.2.3 Output Data

```
nodeTraf = zeros(1, nNodes);
for f=1:nFlows
    if sol(f) ~= 0
       nodes = sP{f}{sol(f)};
       for k = nodes
          nodeTraf(k) = nodeTraf(k) + sum(T(f,3:4));
       end
    end
end

% Detect the unused Links of the solution
sleepingLinks = ''; NsleepLinks = 0;
for i = 1 : size(Loads, 1)
    if max(Loads(i, 3:4)) == 0
        NsleepLinks = NsleepLinks + 1;
        sleepingLinks = append(sleepingLinks,
                        ' {', num2str(Loads(i,1)), ', ',
                              num2str(Loads(i,2)), '}');
    end
end

NodeEnergy = sum(20 + 80 * sqrt(nodeTraf/Node_cap));
TotalEne = NodeEnergy + LinkEne;

auxSum = 0;
cnt = 0;
for k= 1:nFlows
   if sum(Loads(k,3:4)) ~= 0
       cnt = cnt +2;
       auxSum = auxSum + sum(Loads(k,3:4));
   end
end
AvgLinkLoad = auxSum / cnt;

% prints (ommited)
```

1.2.4 Functions

The GreedyRandomizedLOAD function implements the greedy randomized path selection algorithm for load optimization. This algorithm searches various paths for each flow and prioritizes those that reduce link loads. The randomization at the beginning of the function serves to introduce diversity in the path selection process, potentially leading to different solutions and times in which the best solution was found to vary. This "Greedy Randomized" approach contributes to the algorithm's ability to find solutions that minimize network load in together with the HillClimbing algorithm.

```
    function [sol, maxLoad, Loads, linkEnergy] = GreedyRandomizedLOAD(...)
    nFlows = size(T, 1);
    % flows in random order
    randFlows = randperm(nFlows);
    sol = zeros(1, nFlows);

    % iterate through each flow
    for flow = randFlows
        path_index = 0;
        best_maxLoad = inf;
        best_Loads = inf;
        best_energy = inf;

        for path = 1 : nSP(flow)
            % try the path for that flow
            sol(flow) = path;
            % calculate loads
            [Loads, linkEnergy] = calculateLinkLoadEnergy(nNodes,
                                  Links, T, sP, sol, L, Lcap);
            maxLoad = max(max(Loads(:, 3:4)));

            % check if the current load is better then bestLoad
            if maxLoad < best_maxLoad
                % change index of path and load
                path_index = path;
                best_maxLoad = maxLoad;
                best_Loads = Loads;
                best_energy = linkEnergy;
            end
        end
        sol(flow) = path_index;
    end
    Loads = best_Loads;
    maxLoad = max(max(Loads(:, 3:4)));
    linkEnergy = best_energy;
end
```

The HillClimbingLOAD function implements a Hill Climbing strategy to optimize the routing solution for minimizing network load. This approach incrementally refines the solution by exploring alternative paths for each flow. It then compares these alternative paths to ensure that the algorithm replaces the resulting best path until further iterations do not yield improvements, making it a valuable strategy for optimizing routing configurations.

```
function [sol, maxLoad, Loads, linkEnergy] = HillClimbingLOAD(...)
nFlows = size(T,1);
% set the best local variables
maxLoad = max(max(Loads(:, 3:4)));
bestLocalLoad = maxLoad;
bestLocalLoads = Loads;
bestLocalSol = sol;
bestLocalEnergy = linkEnergy;

% 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
                [auxLoads, auxLinkEnergy] = calculateLinkLoadEnergy(nNodes,
                                            Links, T, sP, auxSol, L, Lcap);
                auxMaxLoad = max(max(auxLoads(:, 3:4)));

                % check if the current load is better then start load
                if auxMaxLoad < bestLocalLoad
                    bestLocalLoad = auxMaxLoad;
                    bestLocalLoads = auxLoads;
                    bestLocalSol = auxSol;
                    bestLocalEnergy = auxLinkEnergy;
                end
            end
        end
    end

    if bestLocalLoad < maxLoad
        maxLoad = bestLocalLoad;
        Loads = bestLocalLoads;
```

```
            sol = bestLocalSol;
            linkEnergy = bestLocalEnergy;
        else
            improved = false;
        end
    end
end
```

The calculateLinkLoadEnergy function computes the link loads and energy consumption for the given routing solution. This function is used to evaluate the impact of the routing configuration on the network.

```
    function [Loads, linkEnergy] = calculateLinkLoadEnergy(...)
    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)];
    linkEnergy = 0;
    for i= 1:nLinks
        Loads(i,3)= aux(Loads(i,1),Loads(i,2));
        Loads(i,4)= aux(Loads(i,2),Loads(i,1));

        maxLoad = max(max(Loads(:, 3:4)));
        % If the worst link load is greater than max capacity , energy will be infinite
        if maxLoad > capacity
            linkEnergy = inf;
        else
            % link in sleeping mode
            if max(Loads(i, 3:4)) == 0
                linkEnergy = linkEnergy + 2;
            else
                % len from nodeA to nodeB
                len = L(Loads(i, 1), Loads(i, 2));

                % energy calculation dependent of link capacity
                if capacity == 100
                    linkEnergy = linkEnergy + 9 + 0.3 * len;
```

```
                else
                    fprintf('Error: Link capacity is not 100Gbps\n');
                end
            end
        end
    end
end
```

1.3 Task 1.c)

Upon compiling, the following results were reached:

```
******  TASK 1 ******
### Exericio 1.c ###
Worst link load of the solution: 87.80 Gbps
Average link load of the solution: 47.99 Gbps
Network energy consumption of the solution: 2845.68 W
        Node energy: 815.38 W
        Link energy: 2030.30 W
Avg. Round trip propagation delay Service1: 5.63667 ms
Avg. Round trip propagation delay Service2: 5.61125 ms
Links not supporting any traffic flow: 4 links -> (Src, Dest):  (1, 5) (2, 5) (6, 15) (12, 13)
Number of cycles run by the algorithm: 49143
Running time at which the algorithm has obtained the best solution: 31.9003 ms
```

Figure 1.1: Compilation Results

1.4 Task 1.d)

Upon compiling, now with k = 6 (instead of k=2), the following results were reached:

```
******  TASK 1 ******
### Exericio 1.d ###
Worst link load of the solution: 85.20 Gbps
Average link load of the solution: 44.86 Gbps
Network energy consumption of the solution: 3019.26 W
        Node energy: 831.26 W
        Link energy: 2188.00 W
Avg. Round trip propagation delay Service1: 6.06750 ms
Avg. Round trip propagation delay Service2: 5.66375 ms
Links not supporting any traffic flow: 2 links -> (Src, Dest):  (6, 15) (12, 13)
Number of cycles run by the algorithm: 8924
Running time at which the algorithm has obtained the best solution: 59.6787 ms
```

Figure 1.2: Compilation Results

1.5 Task 1.e)

The parameter k in the k-shortest path algorithm determines the number of alternative paths considered for each flow. Increasing the value k will result in a larger number of paths being considered. This increases computational complexity, but can potentially lead to more diverse solutions.

As we can verify in the results, the number of cycles run by the algorithm decreases drastically, from 49K down to around 9K, confirming that the computational complexity did very much increase. However, even with a much lower number of solutions generated, it reached close or better results in a lot of the other fields. Worst link load decreased as well as the average link load of the solution, with Network energy consumption and average round trip propagation increasing, but only slightly.

Chapter 2

Task 2

2.1 Task 2.a)

For Task 2, the optimization objective shifs now towards minimizing the energy consumption of the network instead of the worst link load. To address this, the algorithm developed in Task 1.b was adapted to compute a symmetrical single-path routing solution supporting both services while aiming to minimize network energy consumption.

2.2 Task 2.b)

While in the previous Task 1.b, after the greedyRandomized and HillClimbing functions are executed (this time a variation of them focusing more on energy), we would compare the maxLoad with the bastLoad found that far, we now compare the linkEnergy.

2.2.1 Changes to main

```
...
while toc(t) < timeLimit
while maxLoad > Link_cap
    [sol, maxLoad, Loads, Linkenergy] = GreedyRandomizedEne(nNodes, Links,
                                        T, sP, nSP, L, Link_cap);
end

[sol, maxLoad, Loads, Linkenergy] = HillClimbingEne(nNodes, Links, T,
                                    sP, nSP, sol, Loads,
                                    Linkenergy, L, Link_cap);

if Linkenergy < bestEne % <--
    (rest of code)
end
...
```

2.2.2 Changes to functions

GreedyRandomizedEne calls calculateLinkLoadEnergy and compares the result based on linkEnergy istead of maxLoad (GreedyRandomizedLOAD).

```
function [sol, maxLoad, Loads, linkEnergy] = GreedyRandomizedEne(nNodes, Links, T, sP, n
nFlows = size(T, 1);
% random order of flows
randFlows = randperm(nFlows);
sol = zeros(1, nFlows);

% iterate through each flow
for flow = randFlows
    path_index = 0;
    best_Loads = inf;
    best_energy = inf;

    % test every path "possible" in a certain load
    for path = 1 : nSP(flow)
        % try the path for that flow
        sol(flow) = path;
        % calculate loads
        [Loads, linkEnergy] = calculateLinkLoadEnergy(nNodes,
                              Links, T, sP, sol, L, Lcap);

        % check if the current load is better then bestLoad
        if linkEnergy < best_energy
            % change index of path and load
            path_index = path;
```

```
                best_Loads = Loads;
                best_energy = linkEnergy;
            end
        end
        sol(flow) = path_index;
    end
    Loads = best_Loads;
    maxLoad = max(max(Loads(:, 3:4)));
    linkEnergy = best_energy;
end
```

While HillClimbingLOAD seeks to ensure an equitable distribution of link loads, HillClimbingEne prioritizes the reduction of energy consumption.

```
    function [sol, maxLoad, Loads, linkEnergy] = HillClimbingEne(nNodes, Links, T, sP, nSP,
    nFlows = size(T,1);
    % set the best local variables
    maxLoad = max(max(Loads(:, 3:4)));
    bestLocalLoad = maxLoad;
    bestLocalLoads = Loads;
    bestLocalSol = sol;
    bestLocalEnergy = linkEnergy;

    % 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
                    [auxLoads, auxLinkEnergy] = calculateLinkLoadEnergy(nNodes,
                                                Links, T, sP, auxSol, L, Lcap);
                    auxMaxLoad = max(max(auxLoads(:, 3:4)));
                    % check if the current load is better then start load
                    if auxLinkEnergy < bestLocalEnergy
                        bestLocalLoad = auxMaxLoad;
                        bestLocalLoads = auxLoads;
                        bestLocalSol = auxSol;
                        bestLocalEnergy = auxLinkEnergy;
                    end
                end
```

```
            end
        end

        if bestLocalEnergy < linkEnergy
            maxLoad = bestLocalLoad;
            Loads = bestLocalLoads;
            sol = bestLocalSol;
            linkEnergy = bestLocalEnergy;
        else
            improved = false;
        end
    end
end
```

2.2.3 Results

```
******  TASK 2 ******
### Exericio 2.b ###
Multi start hill climbing with greedy randomized (all possible paths):
Worst link load of the solution: 99.80 Gbps
Average link load of the solution: 55.09 Gbps
Network energy consumption of the solution: 2286.61 W
        Node energy: 804.01 W
        Link energy: 1482.60 W
Avg. Round trip propagation delay Service1: 5.84 ms
Avg. Round trip propagation delay Service2: 5.03 ms
Links not supporting any traffic flow: 9 links -> (Src, Dest):  (2, 3) (2, 5) (3, 6) (3, 8) (6, 8) (6, 15) (9, 10) (12, 13) (13, 15)
Number of cycles run by the algorithm: 44731
Running time at which the algorithm has obtained the best solution: 18.2832 ms
```

Figure 2.1: Compilation Results

2.3 Task 2.c)

```
******  TASK 2 ******
### Exericio 2.c ###
Multi start hill climbing with greedy randomized (all possible paths):
Worst link load of the solution: 99.80 Gbps
Average link load of the solution: 61.71 Gbps
Network energy consumption of the solution: 2201.69 W
        Node energy: 792.79 W
        Link energy: 1408.90 W
Avg. Round trip propagation delay Service1: 5.70 ms
Avg. Round trip propagation delay Service2: 6.35 ms
Links not supporting any traffic flow: 11 links -> (Src, Dest):  (1, 2) (1, 7) (2, 3) (3, 6) (3, 8) (4, 10) (6, 14) (12, 13) (12, 14) (13, 14) (14, 15)
Number of cycles run by the algorithm: 9208
Running time at which the algorithm has obtained the best solution: 50.6844 ms
```

Figure 2.2: Compilation Results

2.4 Task 2.d)

The parameter k in the k-shortest path algorithm determines the number of alternative paths considered for each flow. Just link Task 1.d, we can observe a decline of the amount of cycles realizes within the time frame is much smaller for k = 6, the results are better than with k = 2.

2.5 Task 2.e)

We can see that in 1.d we have better worst link loads and overall smaller link load values. in 2.c we have higher link load values, but the energy consumption is considerably smaller.
At the end of the day, the choice between the two approaches depend on the specific priorities of the network operator. If achieving a well-balanced link load is paramount, focusing on mitigating worst link load (2.d) is more appropriate. On the other hand, if energy efficiency is the key concern, mitigating energy spending (2.c) is the preferred choice.

```
******  TASK 1 ******
### Exericio 1.d ###
Worst link load of the solution: 85.20 Gbps
Average link load of the solution: 44.86 Gbps
Network energy consumption of the solution: 3019.26 W
        Node energy: 831.26 W
        Link energy: 2188.00 W
Avg. Round trip propagation delay Service1: 6.06750 ms
Avg. Round trip propagation delay Service2: 5.66375 ms
Links not supporting any traffic flow: 2 links -> {Src, Dest}:  {6, 15} {12, 13}
Number of cycles run by the algorithm: 8924
Running time at which the algorithm has obtained the best solution: 59.6787 ms
```

Figure 2.3: Compilation Results

```
******  TASK 2 ******
### Exericio 2.c ###
Multi start hill climbing with greedy randomized (all possible paths):
Worst link load of the solution: 99.80 Gbps
Average link load of the solution: 61.71 Gbps
Network energy consumption of the solution: 2201.69 W
        Node energy: 792.79 W
        Link energy: 1408.90 W
Avg. Round trip propagation delay Service1: 5.70 ms
Avg. Round trip propagation delay Service2: 6.35 ms
Links not supporting any traffic flow: 11 links -> {Src, Dest}:  {1, 2} {1, 7} {2, 3} {3, 6} {3, 8} {4, 10} {6, 14} {12, 13} {12, 14} {13, 14} {14, 15}
Number of cycles run by the algorithm: 9208
Running time at which the algorithm has obtained the best solution: 50.6844 ms
```

Figure 2.4: Compilation Results

Chapter 3

Task 3

Chapter 4

Task 4

4.1 Task 4.a)

In the scope of Task 4, the optimization objective is similar to the one in Task 2, minimizing the energy consumption of the network but with the introduction of an anycast service. To resolve this, the algorithm developed in Task 2.a was adapted to compute a symmetrical single-path routing solution supporting all services while aiming to minimize network energy consumption.

4.2 Task 4.b)

For task 4.b the algorithms used were based in the one used for Task 2.b, only differing by calculating the traffic flows for the anycast service and then grouping it with the unicast services, creating a new global matrix with all the services. Notice that also proceeded to calculate the round trip propagation for the anycast service and changes to the functions weren't needed.

4.2.1 Changes to main

```
    ...
anyNodes = [3 10];

% Traffic flows for unicast service

(...)

% Traffic flows for anycast service
    sP_any = cell(1, nNodes);
    nSP_any = zeros(1, nNodes);
    for n = 1:nNodes
```

```
        if ismember(n, anyNodes)         % if the node is a anycastNode skip it
            nSP_any(n) = -1;
            continue;
        end

        if ~ismember(n, T_any(:, 1))     % node is not from T_any matrix
            nSP_any(n) = -1;
            continue;
        end

        best = inf;
        for a = 1:length(anyNodes)
            [shortestPath, totalCost] = kShortestPath(L, n, anyNodes(a), 1);

            if totalCost(1) < best
                sP_any{n} = shortestPath;
                nSP_any(n) = length(totalCost);
                best = totalCost;
            end
        end
    end

    nSP_any = nSP_any(nSP_any~=-1);
    sP_any = sP_any(~cellfun(@isempty, sP_any));

 % New general matrix
T_any = [T_any(:, 1) zeros(size(T_any,1), 1) T_any(:, 2:3)];
for i = 1 : size(T_any, 1)
    T_any(i, 2) = sP_any{i}{1}(end);
end

T = [T_uni; T_any];
sP = cat(2, sP_uni, sP_any);
nSP = cat(2, nSP_uni, nSP_any);
nFlows = size(T,1);
for f = 1:nFlows
   [shortestPath, totalCost] = kShortestPath(L,T(f,1),T(f,2),k);
   sP{f} = shortestPath;
   nSP(f) = length(totalCost);
end
    ...
```

4.2.2 Results

```
******  TASK 4 ******
### Exericio 4.b ###
Multi start hill climbing with greedy randomized (all possible paths):
Worst link load of the solution: 98.80 Gbps
Average link load of the solution: 56.31 Gbps
Network energy consumption of the solution: 2360.92 W
        Node energy: 827.22 W
        Link energy: 1533.70 W
Avg. Round trip propagation delay Service1: 4.55 ms
Avg. Round trip propagation delay Service2: 6.51 ms
Avg. Round trip propagation delay Service3: 4.77 ms
Links not supporting any traffic flow: 8 links -> {Src, Dest}:  {1, 2} {1, 7} {4, 9} {6, 8} {6, 15} {11, 13} {13, 14} {13, 15}
Number of cycles run by the algorithm: 5208
Running time at which the algorithm has obtained the best solution: 39.3592 ms
```

Figure 4.1: Compilation Results

After introducing the anycast service we can notice that the round trip propagation delays for each services got more balanced due to the new service helping in load distribution and preventing congestion on a single server, enhancing overall network performance and reliability.

4.3 Task 4.c)

In this exercise we intended to find which pair of nodes minimized the average round-trip propagation delay of the anycast service.

4.3.1 Changes to main

```
    ...
AnycastSrcNodes = T3(:,1);
lowestDelayT3 = inf;

for x = 3:nNodes
    for y = x:nNodes
        if x ~= y
           if (~ismember(x, AnycastSrcNodes) && ~ismember(y, AnycastSrcNodes))
                anyNodes = [x y];

                           (...)

                       % Calculate round-trip delay for each flow in the best solution
                        roundTripDelayService1 = zeros(12, 1);
                        roundTripDelayService2 = zeros(8, 1);
                        roundTripDelayService3 = zeros(9, 1);

    for f = 1:nFlows
        if sol(f) ~= 0
```

```
            pathNodes = sP{f}{sol(f)};
            totalPathDelay = 0;
            for i = 1:(length(pathNodes)-1)

                linkIndex = find((Links(:,1) == pathNodes(i) &
                    Links(:,2) == pathNodes(i+1)) |
                        (Links(:,1) == pathNodes(i+1) &
                            Links(:,2) == pathNodes(i)));

                totalPathDelay = totalPathDelay + D(pathNodes(i),pathNodes(i+1));
            end
            roundTripDelay = 2 * totalPathDelay * 1000;  % Round-trip delay
            if f <= 12
                roundTripDelayService1(f) = roundTripDelay;
            elseif f <= 20
                roundTripDelayService2(f) = roundTripDelay;
            else
                roundTripDelayService3(f) = roundTripDelay;
            end

        end
    end
                    end
                    contador= contador + 1;
                    somador= somador + maxLoad;
                end
                delayT3 = mean(roundTripDelayService3(roundTripDelayService3 ~= 0))
                if  delayT3 < lowestDelayT3
                    bestPair = anyNodes;
                    lowestDelayT3 = delayT3;
                end
    ...
```

In this algorithm we test each pair, compare the round trip propagation delays of the anycast service and save the combination with the lowest delay, which we concluded to be nodes [3 8];

4.4 Task 4.d)

Using the combination found in the topic above we obtained the following results.

```
******  TASK 4 ******
### Exericio 4.d ###
Multi start hill climbing with greedy randomized (all possible paths):
Worst link load of the solution: 97.60 Gbps
Average link load of the solution: 58.41 Gbps
Network energy consumption of the solution: 2407.72 W
        Node energy: 852.92 W
        Link energy: 1554.80 W
Avg. Round trip propagation delay Service1: 5.46 ms
Avg. Round trip propagation delay Service2: 6.20 ms
Avg. Round trip propagation delay Service3: 4.60 ms
Links not supporting any traffic flow: 7 links -> {Src, Dest}:  {1, 2} {1, 7} {2, 3} {4, 9} {6, 8} {6, 15} {13, 15}
Number of cycles run by the algorithm: 6223
Running time at which the algorithm has obtained the best solution: 72.6505 ms
```

Figure 4.2: Compilation Results

4.5 Task 4.e)

With this new combination we can conclude that the propagation delay for the anycast service decreased but the other services didn't necessarily improve.