Universidade de Aveiro

Desempenho e Dimensionamento de Redes

deti, 2016/2017

Relatório do Trabalho no.4

Autores:
Pedro Coelho 59517
Nuno Silva 72708

Professores:
Amaro Fernandes de Sousa

15 de Junho de 2017

Conteúdo

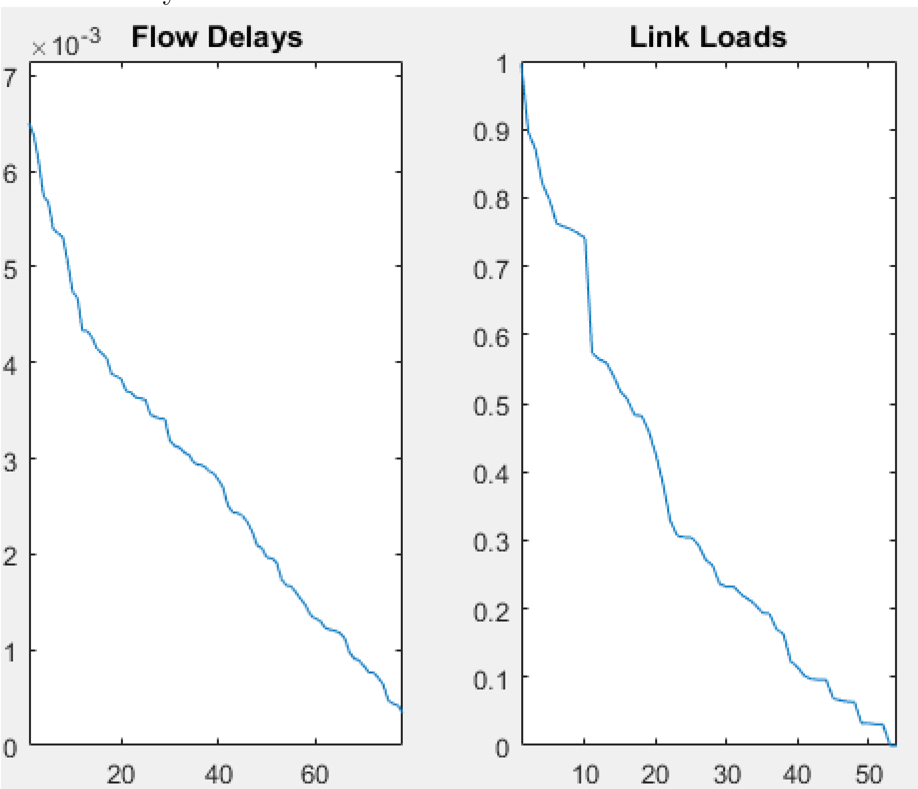
Resumo

1 Trabalho

1.1 a.)

Ao executarmos o scrip fornecido obtemos os seguintes resultados:

MaximumLoad = 0.9960
AverageLoad = 0.3478
AverageDelay = 0.0032
MaxAvDelay = 0.0065



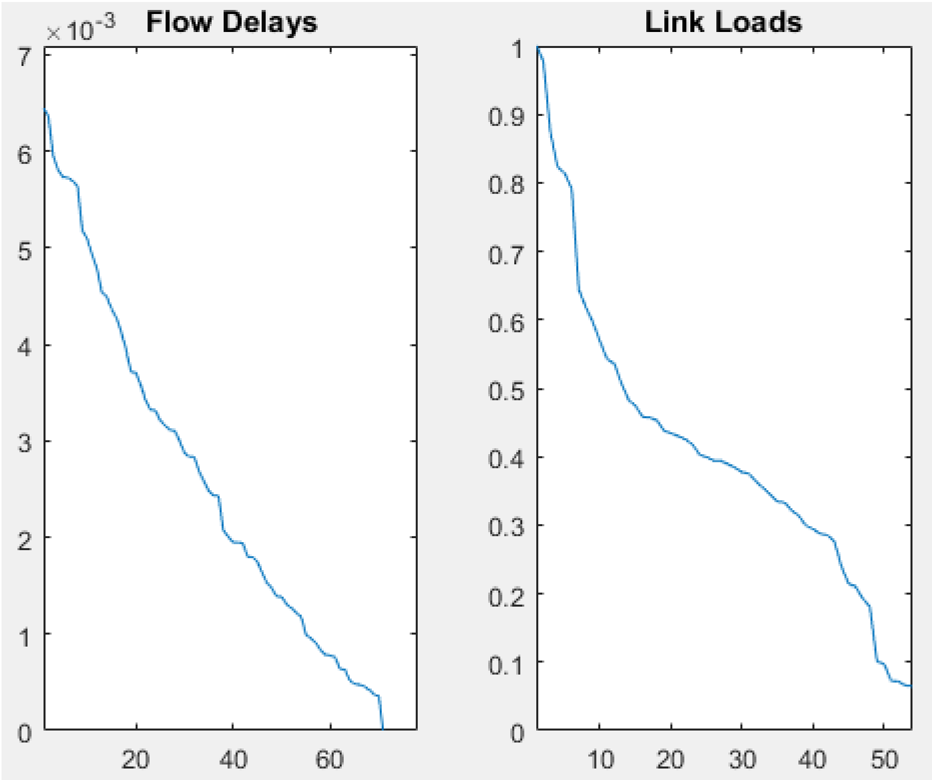
1.2 b.)

O problema b impõe como custo a soma da carga nas ligações pelo que foi necessário editar o código fornecido para ter isto em conta. Desta alteração resultou o código que se segue.

```
Matrizes;
miu= R*1e9/(8*1000);
NumberLinks= sum(sum(R>0));
lambda_s= T*1e6/(8*1000);
gama= sum(sum(lambda_s));
d= L*1e3/2e8;

pairs= [];
for origin=1:16
    for destination=(origin+1):17
        if T(origin,destination)+T(destination,origin)>0
            pairs= [pairs; origin destination];
```

```
            end
        end
    end
    npairs= size(pairs,1);
    lambda= zeros(17);
    routes= zeros(npairs,17);
    for i=1:npairs
        origin= pairs(i,1);
        destination= pairs(i,2);
        Load= lambda./miu;
        r= ShortestPathSym(Load,origin,destination);
        routes(i,:)= r;
        j= 1;
        while r(j)~= destination
            lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) +...
            lambda_s(origin,destination);
            lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) +...
            lambda_s(destination,origin);
            j= j+1;
        end
    end
    Load= lambda./miu;
    Load(isnan(Load))= 0;
    MaximumLoad= max(max(Load))
    AverageLoad= sum(sum(Load))/NumberLinks

    AverageDelay= (lambda./(miu-lambda)+lambda.*d);
    AverageDelay(isnan(AverageDelay))= 0;
    AverageDelay= 2*sum(sum(AverageDelay))/gama
    Delay_s= zeros(npairs,1);
    for i=1:npairs
        origin= pairs(i,1);
        destination= pairs(i,2);
        r= routes(i,:);
        j= 1;
        while r(j)~= destination
            Delay_s(i)= Delay_s(i)+ 1/(miu(r(j),r(j+1))-...
                lambda(r(j),r(j+1))) + d(r(j),r(j+1));
            Delay_s(i)= Delay_s(i)+ 1/(miu(r(j+1),r(j))-...
                lambda(r(j+1),r(j))) + d(r(j+1),r(j));
            j= j+1;
        end
    end
    MaxAvDelay= max(Delay_s)
```

A execução deste scrip providenciou os seguintes resultados:
MaximumLoad = 1.0020
AverageLoad = 0.4109
AverageDelay = 0.0024
MaxAvDelay = 0.0064

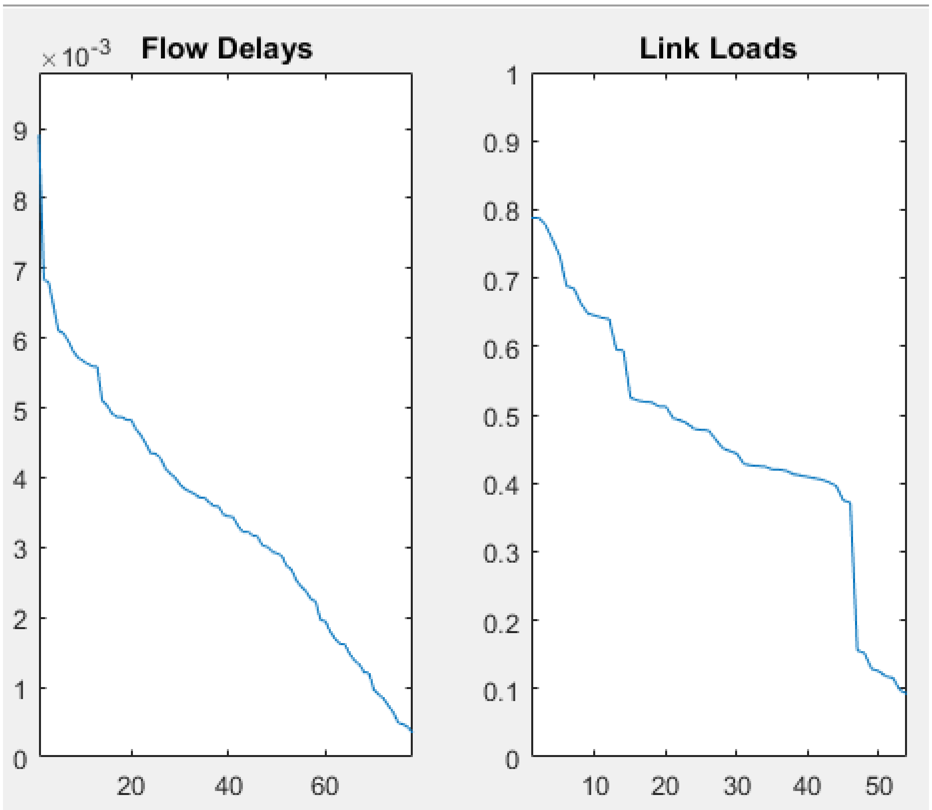


A obtenção de um Maximum load superior a 1 torna claro que neste caso não faz sentido a aproximação de Kleinrock.

1.3 b'.)

Devido à alínea anterior não fazer sentido foi criada uma versão alternativa à mesma onde os custos seriam o quadrado, pequena alteração surtiu resultados com valores que podem ser considerados.

MaximumLoad = 0.7880
AverageLoad = 0.4629
AverageDelay = 0.0037
MaxAvDelay = 0.0089



1.4 c.)

Um outro cenário onde cenário onde o custo a considerar é o tempo de viagem do pacote criou uma outra versão do simulador.

```
Matrizes;
miu= R*1e9/(8*1000);
NumberLinks= sum(sum(R>0));
lambda_s= T*1e6/(8*1000);
gama= sum(sum(lambda_s));
d= L*1e3/2e8;

pairs= [];
for origin=1:16
    for destination=(origin+1):17
        if T(origin,destination)+T(destination,origin)>0
            pairs= [pairs; origin destination];
        end
    end
end
npairs= size(pairs,1);
lambda= zeros(17);
routes= zeros(npairs,17);
for i=1:npairs
    origin= pairs(i,1);
    destination= pairs(i,2);
    aux=1./(miu-lambda) + d;
    r= ShortestPathSym(aux,origin,destination);
    routes(i,:)= r;
```

```
    j= 1;
    while r(j)~= destination
        lambda(r(j),r(j+1))= lambda(r(j),r(j+1)) + ...
            lambda_s(origin,destination);
        lambda(r(j+1),r(j))= lambda(r(j+1),r(j)) + ...
            lambda_s(destination,origin);
        j= j+1;
    end
end
Load= lambda./miu;
Load(isnan(Load))= 0;
MaximumLoad= max(max(Load))
AverageLoad= sum(sum(Load))/NumberLinks

AverageDelay= (lambda./(miu-lambda)+lambda.*d);
AverageDelay(isnan(AverageDelay))= 0;
AverageDelay= 2*sum(sum(AverageDelay))/gama
Delay_s= zeros(npairs,1);
for i=1:npairs
    origin= pairs(i,1);
    destination= pairs(i,2);
    r= routes(i,:);
    j= 1;
    while r(j)~= destination
        Delay_s(i)= Delay_s(i)+ 1/(miu(r(j),r(j+1))-...
            lambda(r(j),r(j+1))) + d(r(j),r(j+1));
        Delay_s(i)= Delay_s(i)+ 1/(miu(r(j+1),r(j))-...
            lambda(r(j+1),r(j))) + d(r(j+1),r(j));
        j= j+1;
    end
end
MaxAvDelay= max(Delay_s)

subplot(1,2,1)
printDelay_s= sortrows(Delay_s,-1);
plot(printDelay_s)
axis([1 npairs 0 1.1*MaxAvDelay])
title('Flow Delays')
subplot(1,2,2)
printLoad= sortrows(Load(:),-1);
printLoad= printLoad(1:NumberLinks);
plot(printLoad)
axis([1 NumberLinks 0 1])
title('Link Loads')
```

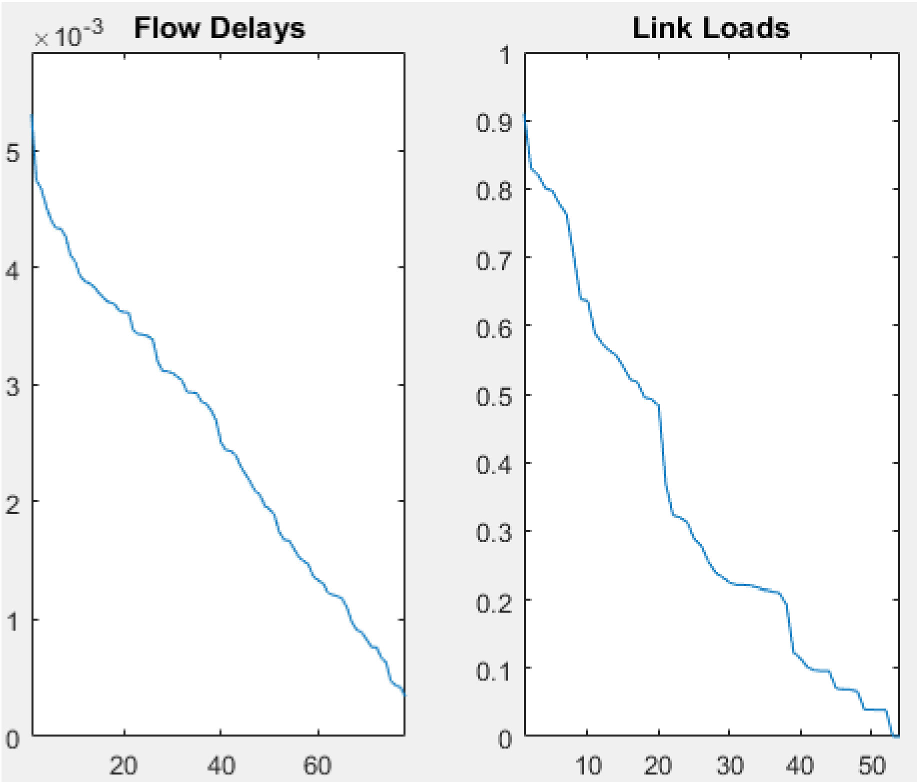
Os resultados obtivos foram os seguintes:

MaximumLoad = 0.9100

AverageLoad = 0.3447

AverageDelay = 0.0026

MaxAvDelay = 0.0053



1.5 d.)

Ao compararmos os resultados é possivél extrapolar que a solução B' tornou possivél um equilibrio entre a carga aplicada reduzindo consideravelmente o load máximo, em contrapartido o delay teve o comportamento inverso. A solução C, que apenas difere da à na soma do termo $1/\mu + \lambda$ em cada iteração, permitiu uma redução do delay e uma ligeira redução do load maximo, pelo que esta solução parece ser a mais favorável

1.6 e.)

O script de simulação para esta alínea é baseado naquele presente nos slides teóricos da disciplina. Este simualdor foi ainda desenvolvido por forma a testar a simulação as n vezes necessárias apenas numa execução.

Todas as funções auxiliares desenvolvidas têm em conta o atraso médio como valor de referência para o calculo do custo.

```
% Lowest Delay
clear all; clc; format long;

n = [3, 10, 30, 300, 3000];


fprintf('\nLowest Delay\n');
fprintf('    n\tGlobalBest\n');

for k = 1:length(n)
    %Global search
    GlobalBest = Inf;
    for j = 1:n(k)
        CurrentSolution = GreedyRandomized_E();
```

```
        CurrentObjective = Evaluate_E(CurrentSolution);
        repeat = true;

        %Local search
        while repeat
            NeighbourBest = Inf;

            %Calculating best neightboor
            for i=1:size(CurrentSolution,1)
                NeighbourSolution =...
                BuildNeighbour_E(CurrentSolution,i);
                NeighbourObjective = Evaluate_E(NeighbourSolution);
                if NeighbourObjective < NeighbourBest
                    NeighbourBest = NeighbourObjective;
                    NeighbourBestSolution = NeighbourSolution;
                end
            end

            %Is current better than best set it as new best
            if NeighbourBest < CurrentObjective
                CurrentObjective = NeighbourBest;
                CurrentSolution = NeighbourBestSolution;
            %If cant find a better solution dont repeat
            else
                repeat = false;
            end
        end

        %If current better than best set it as the new best
        if CurrentObjective < GlobalBest
            GlobalBestSolution = CurrentSolution;
            GlobalBest = CurrentObjective;
        end
    end

    fprintf('%5d\t%0.6f\n', n(k), GlobalBest);
end

pairs = GlobalBestSolution.pairs;
routes = GlobalBestSolution.routes;
lambda = GlobalBestSolution.lambda;

Matrizes;
miu= R*1e9/(8*1000);
NumberLinks= sum(sum(R>0));
lambda_s= T*1e6/(8*1000);
gama= sum(sum(lambda_s));
d= L*1e3/2e8;

npairs= size(pairs,1);
```

```
Load= lambda./miu;
Load(isnan(Load))= 0;
MaximumLoad= max(max(Load))
AverageLoad= sum(sum(Load))/NumberLinks

AverageDelay= (lambda./(miu-lambda)+lambda.*d);
AverageDelay(isnan(AverageDelay))= 0;
AverageDelay= 2*sum(sum(AverageDelay))/gama
Delay_s= zeros(npairs,1);
for i=1:npairs
    origin= pairs(i,1);
    destination= pairs(i,2);
    r= routes(i,:);
    j= 1;
    while r(j)~= destination
        Delay_s(i)= Delay_s(i)+ 1/(miu(r(j),r(j+1))-...
            lambda(r(j),r(j+1))) + d(r(j),r(j+1));
        Delay_s(i)= Delay_s(i)+ 1/(miu(r(j+1),r(j))-...
            lambda(r(j+1),r(j))) + d(r(j+1),r(j));
        j= j+1;
    end
end
MaxAvDelay= max(Delay_s)

subplot(1,2,1)
printDelay_s= sortrows(Delay_s,-1);
plot(printDelay_s)
axis([1 npairs 0 1.1*MaxAvDelay])
title('Flow Delays')
subplot(1,2,2)
printLoad= sortrows(Load(:),-1);
printLoad= printLoad(1:NumberLinks);
plot(printLoad)
axis([1 NumberLinks 0 1])
title('Link Loads')
```

Greedy Randomized:

```
function solution = GreedyRandomized_E()
% Randomizes pair order

Matrizes;
miu = R*1e9/(8*1000);
NumberLinks = sum(sum(R>0));
lambda_s = T*1e6/(8*1000);
gama = sum(sum(lambda_s));
d = L*1e3/2e8;

pairs = [];
for origin = 1:16
```

```
    for destination = (origin + 1):17
        if T(origin, destination) + T(destination, origin)>0
            pairs = [pairs; origin destination];
        end
    end
end

% npairs = column length
npairs = size(pairs,1);
% random permutation of the nodes from 1 to npairs.
b = randperm(npairs);
for i = 1:npairs
    % reorder by the random permutation
    aux(i,:) = pairs(b(i),:);
end
pairs = aux;

lambda = zeros(17);
routes = zeros(npairs,17);

% same as solution C
for i = 1:npairs
    origin = pairs(i,1);
    destination = pairs(i,2);

    AverageDelay = (1./(miu-lambda)+d);
    AverageDelay(isnan(AverageDelay))= 0;
    r = ShortestPathSym(AverageDelay,origin,destination);

    routes(i,:)= r;
    j= 1;

    while r(j) ~= destination
        lambda(r(j),r(j+1)) = lambda(r(j),r(j+1)) +...
        lambda_s(origin,destination);
        lambda(r(j+1),r(j)) = lambda(r(j+1),r(j)) +...
        lambda_s(destination,origin);
        j= j+1;
    end
end

solution.pairs = pairs;
solution.routes = routes;
solution.lambda = lambda;
end
```

Evaluate:

```
function Solution = Evaluate_E(CurrentSolution)
lambda = CurrentSolution.lambda;
Matrizes;
```

```
miu= R*1e9/(8*1000);
d= L*1e3/2e8;
lambda_s= T*1e6/(8*1000);
gama= sum(sum(lambda_s));


AverageDelay= (lambda./(miu-lambda)+lambda.*d);
AverageDelay(isnan(AverageDelay))= 0;
AverageDelay = 2*sum(sum(AverageDelay))/gama;


Solution = AverageDelay;
end
```

Build Neighbour:

```
function NeighbourSolution = BuildNeighbour_E(solution, i)
Matrizes;
lambda_s = T * 1e6 / (8*1000);
miu = R * 1e9 / (8*1000);
d = L * 1e3 / 2e8;

% get current origin and destination
origin = solution.pairs(i,1);
destination = solution.pairs(i,2);

% reset lambda
r = solution.routes(i,:);
j = 1;
while r(j) ~= destination
    solution.lambda(r(j),r(j+1)) = solution.lambda(r(j),r(j+1)) -...
    lambda_s(origin,destination);
    solution.lambda(r(j+1),r(j))= solution.lambda(r(j+1),r(j)) -...
    lambda_s(destination,origin);
    j= j+1;
end

% route choice - by delay
AverageDelay = (1./(miu-solution.lambda)+d);
r = ShortestPathSym(AverageDelay,origin,destination);

% recalculate solution lambda
solution.routes(i,:) = r;
j = 1;
while r(j) ~= destination
    solution.lambda(r(j),r(j+1)) = solution.lambda(r(j),r(j+1)) +...
    lambda_s(origin,destination);
    solution.lambda(r(j+1),r(j))= solution.lambda(r(j+1),r(j)) +...
    lambda_s(destination,origin);
    j= j+1;
end
```

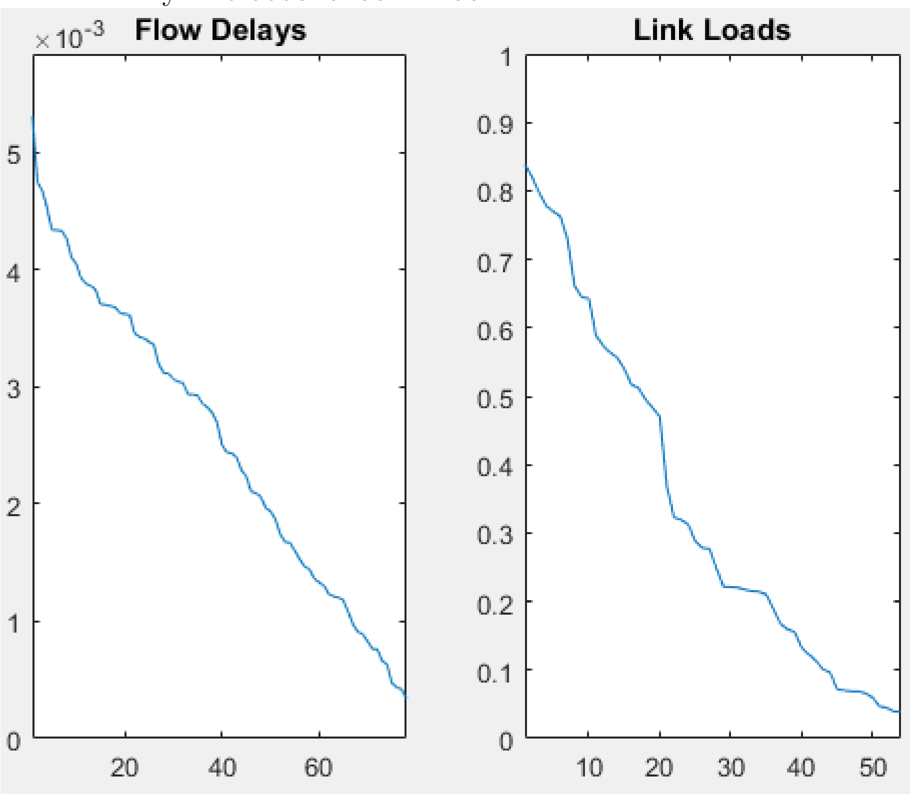
```
NeighbourSolution.pairs = solution.pairs;
NeighbourSolution.routes = solution.routes;
NeighbourSolution.lambda = solution.lambda;
end
```

Os valores obtidos para as várias iterações, e para o globalbest foram os seguintes:

Lowest Delay

n	GlobalBest
3	0.002640
10	0.002639
30	0.002640
300	0.002636
3000	0.002625

MaximumLoad = 0.838000000000000
AverageLoad = 0.342498148148148
AverageDelay = 0.002624882299909
MaxAvDelay = 0.005310230444438



1.7 f.)

O simulador f foi construido da mesma forma que o anterior, a única diferença está no custo que as funções auxiliares utilizam, trantando-se agora do carga da ligação.

```
% Lowest Delay
clear all; clc; format long;

n = [3, 10, 30, 300, 3000];
```

```
fprintf('\nLowest Delay\n');
fprintf('    n\tGlobalBest\n');

for k = 1:length(n)
    %Global search
    GlobalBest = Inf;
    for j = 1:n(k)
        CurrentSolution = GreedyRandomized_F();
        CurrentObjective = Evaluate_F(CurrentSolution);
        repeat = true;

        %Local search
        while repeat
            NeighbourBest = Inf;

            %Calculating best neightboor
            for i=1:size(CurrentSolution,1)
                NeighbourSolution =...
                BuildNeighbour_F(CurrentSolution, i);
                NeighbourObjective = Evaluate_F(NeighbourSolution);
                if NeighbourObjective < NeighbourBest
                    NeighbourBest = NeighbourObjective;
                    NeighbourBestSolution = NeighbourSolution;
                end
            end

            %Is current better than best set it as new best
            if NeighbourBest < CurrentObjective
                CurrentObjective = NeighbourBest;
                CurrentSolution = NeighbourBestSolution;
                %If cant find a better solution dont repeat
            else
                repeat = false;
            end
        end

        %If current better than best set it as the new best
        if CurrentObjective < GlobalBest
            GlobalBestSolution = CurrentSolution;
            GlobalBest = CurrentObjective;
        end
    end

    fprintf('%5d\t%0.6f\n', n(k), GlobalBest);
end

pairs = GlobalBestSolution.pairs;
routes = GlobalBestSolution.routes;
```

```
lambda = GlobalBestSolution.lambda;

Matrizes;
miu= R*1e9/(8*1000);
NumberLinks= sum(sum(R>0));
lambda_s= T*1e6/(8*1000);
gama= sum(sum(lambda_s));
d= L*1e3/2e8;

npairs= size(pairs,1);

Load= lambda./miu;
Load(isnan(Load))= 0;
MaximumLoad= max(max(Load))
AverageLoad= sum(sum(Load))/NumberLinks

AverageDelay= (lambda./(miu-lambda)+lambda.*d);
AverageDelay(isnan(AverageDelay))= 0;
AverageDelay= 2*sum(sum(AverageDelay))/gama
Delay_s= zeros(npairs,1);
for i=1:npairs
    origin= pairs(i,1);
    destination= pairs(i,2);
    r= routes(i,:);
    j= 1;
    while r(j)~= destination
        Delay_s(i)= Delay_s(i)+ 1/(miu(r(j),r(j+1))-...
            lambda(r(j),r(j+1))) + d(r(j),r(j+1));
        Delay_s(i)= Delay_s(i)+ 1/(miu(r(j+1),r(j))-...
            lambda(r(j+1),r(j))) + d(r(j+1),r(j));
        j= j+1;
    end
end
MaxAvDelay= max(Delay_s)

subplot(1,2,1)
printDelay_s= sortrows(Delay_s,-1);
plot(printDelay_s)
axis([1 npairs 0 1.1*MaxAvDelay])
title('Flow Delays')
subplot(1,2,2)
printLoad= sortrows(Load(:),-1);
printLoad= printLoad(1:NumberLinks);
plot(printLoad)
axis([1 NumberLinks 0 1])
title('Link Loads')
```

Greedy Randomized:

```
function solution = GreedyRandomized_F()
% Randomizes pair order
```

```
Matrizes;
miu = R*1e9/(8*1000);
NumberLinks = sum(sum(R>0));
lambda_s = T*1e6/(8*1000);
gama = sum(sum(lambda_s));
d = L*1e3/2e8;

pairs = [];
for origin = 1:16
    for destination = (origin + 1):17
        if T(origin, destination) + T(destination, origin)>0
            pairs = [pairs; origin destination];
        end
    end
end

% npairs = column length
npairs = size(pairs,1);
% random permutation of the nodes from 1 to npairs.
b = randperm(npairs);
for i = 1:npairs
    % reorder by the random permutation
    aux(i,:) = pairs(b(i),:);
end
pairs = aux;

lambda = zeros(17);
routes = zeros(npairs,17);

% same as solution C
for i = 1:npairs
    origin = pairs(i,1);
    destination = pairs(i,2);
    Load= lambda./miu;
    r = ShortestPathSym(Load.^2,origin,destination);

    routes(i,:)= r;
    j= 1;

    while r(j) ~= destination
        lambda(r(j),r(j+1)) = lambda(r(j),r(j+1)) +...
        lambda_s(origin,destination);
        lambda(r(j+1),r(j)) = lambda(r(j+1),r(j)) +...
        lambda_s(destination,origin);
        j= j+1;
    end
end

solution.pairs = pairs;
```

```
solution.routes = routes;
solution.lambda = lambda;
end
```

Evaluate:

```
function Solution = Evaluate_F(CurrentSolution)
lambda = CurrentSolution.lambda;

Matrizes;
miu= R*1e9/(8*1000);

Load= lambda./miu;
Load(isnan(Load))= 0;
MaximumLoad = max(max(Load));

Solution = MaximumLoad;
end
```

Build Neighbour:

```
function NeighbourSolution = BuildNeighbour_F(solution, i)
Matrizes;
lambda=solution.lambda;
lambda_s = T * 1e6 / (8*1000);
miu = R * 1e9 / (8*1000);
d = L * 1e3 / 2e8;

% get current origin and destination
origin = solution.pairs(i,1);
destination = solution.pairs(i,2);

% reset lambda
r = solution.routes(i,:);
j = 1;
while r(j) ~= destination
    solution.lambda(r(j),r(j+1)) = solution.lambda(r(j),r(j+1)) -...
    lambda_s(origin,destination);
    solution.lambda(r(j+1),r(j))= solution.lambda(r(j+1),r(j)) -...
    lambda_s(destination,origin);
    j= j+1;
end

% route choice - by delay
Load= lambda./miu;
r = ShortestPathSym(Load.^2,origin,destination);

% recalculate solution lambda
solution.routes(i,:) = r;
j = 1;
while r(j) ~= destination
    solution.lambda(r(j),r(j+1)) = solution.lambda(r(j),r(j+1)) +...
```

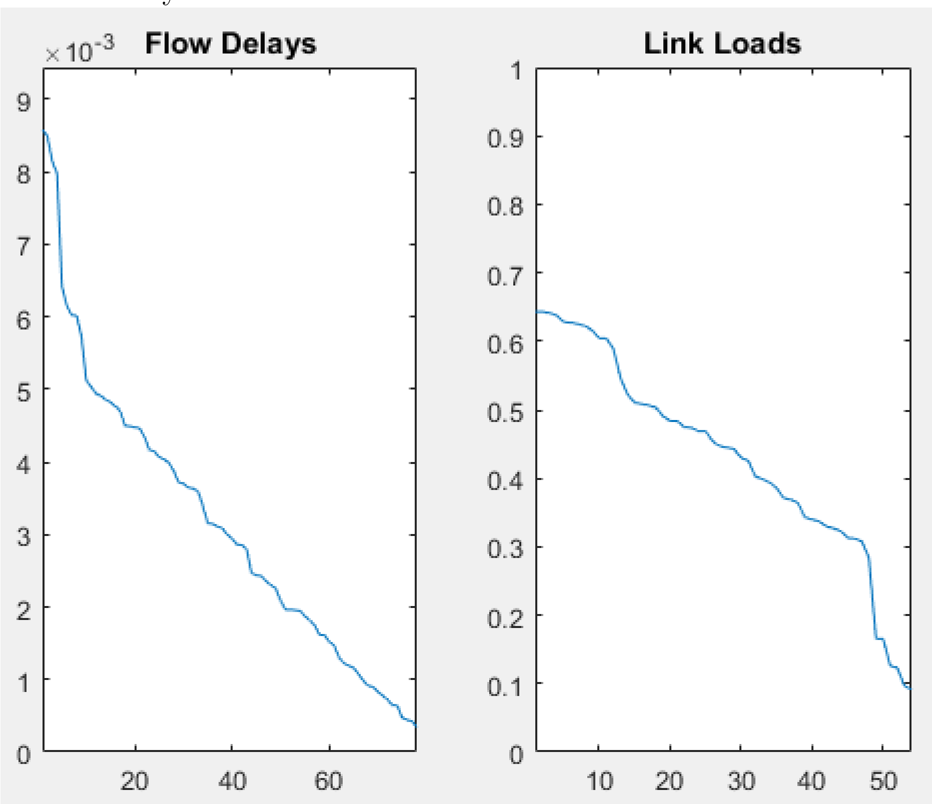
```
        lambda_s(origin , destination);
        solution.lambda(r(j+1),r(j))= solution.lambda(r(j+1),r(j)) +...
        lambda_s(destination , origin);
        j= j+1;
    end

    NeighbourSolution.pairs = solution.pairs;
    NeighbourSolution.routes = solution.routes;
    NeighbourSolution.lambda = solution.lambda;
end
```

Foram obtidos os seguintes resultados:

Lowest Delay

n	GlobalBest
3	0.749000
10	0.677000
30	0.667000
300	0.647000
3000	0.643000

MaximumLoad = 0.643000000000000
AverageLoad = 0.430318518518518
AverageDelay = 0.003727578517302
MaxAvDelay = 0.008580633418399



1.8 g.)

Comparando os resultados obtidos para os exercícios E e F é possível verificar que foram obtidos melhores resultados para os valores no ex E sendo possível verificar delay medio e load medio mais baixos, tendo, no entanto, o load máximo sido consideravelmente mais elevado. É possível verifica r ainda que a solução no exercício E converge mais rapidamente, sendo que para 3, 10 e 30 iterações se obteve resultados semelhantes.

Observandos os resultados obtidos há uma clara convergência mais rápida em E, uma vez que em 3, 10 e 30 houve resultados bastante semelhantes. Conclui-se também que a solução E é a mais favorável pois tem delay e load médios mais baixos, peca apenas por um load máximo mais elevado.