**Universidade de Aveiro**

Departamento de Eletrónica, Telecomunicações e Informática



**Modelação e Desempenho de Redes e Serviços**

**Projeto 2**

**Tiago Alves (104110)**

**Rafael Amorim (98197)**

28 de dezembro de 2023

Índice

[Introdução 1](#_Toc154432971)

[Tarefa 1 2](#_Toc154432972)

[Exercício 1a 2](#_Toc154432973)

[Exercício 1b 2](#_Toc154432974)

[Código: 2](#_Toc154432975)

[Exercício 1c 4](#_Toc154432976)

[Código: 4](#_Toc154432977)

[Exercício 1d 7](#_Toc154432978)

[Exercício 1e 7](#_Toc154432979)

[Resultados: 7](#_Toc154432980)

[Conclusões: 8](#_Toc154432981)

[Tarefa 2 9](#_Toc154432982)

[Exercício 2a 9](#_Toc154432983)

[Código: 9](#_Toc154432984)

[Exercício 2b 11](#_Toc154432985)

[Código: 12](#_Toc154432986)

[Resultado: 13](#_Toc154432987)

[Explicação: 13](#_Toc154432988)

[Exercício 2c 13](#_Toc154432989)

[Resultado: 13](#_Toc154432990)

[Explicação: 14](#_Toc154432991)

[Tarefa 3 14](#_Toc154432992)

[Exercício 3a 14](#_Toc154432993)

[Exercício 3b 14](#_Toc154432994)

[Exercício 3c 14](#_Toc154432995)

[Exercício 3d 14](#_Toc154432996)

[Exercício 3e 15](#_Toc154432997)

[Tarefa 4 15](#_Toc154432998)

[Exercício 4a 15](#_Toc154432999)

[Exercício 4b 15](#_Toc154433000)

[Exercício 4c 15](#_Toc154433001)

[Exercício 4d 15](#_Toc154433002)

[Exercício 4e 15](#_Toc154433003)

[Contribuição dos autores 16](#_Toc154433004)

Índice das Figuras

[Figura 1: Resultado do exercício 1a **Erro! Marcador não definido.**](#_Toc149672781)

[Figura 2: Resultado do exercício 1c **Erro! Marcador não definido.**](https://uapt33090-my.sharepoint.com/personal/rafael_amorim_ua_pt/Documents/reportAssigment1.docx#_Toc149672782)

[Figura 3: Resultado do exercício 1d **Erro! Marcador não definido.**](https://uapt33090-my.sharepoint.com/personal/rafael_amorim_ua_pt/Documents/reportAssigment1.docx#_Toc149672783)

[Figura 4: Resultado do exercício 1e APD sem BER **Erro! Marcador não definido.**](#_Toc149672784)

[Figura 5: Resultado do exercício 1e TT sem BER **Erro! Marcador não definido.**](#_Toc149672785)

[Figura 6: Resultado do exercício 1e APD com BER e pacotes mais pequenos **Erro! Marcador não definido.**](#_Toc149672786)

[Figura 7: Resultado do exercício 1e TT com BER e pacotes mais pequenos **Erro! Marcador não definido.**](#_Toc149672787)

[Figura 8: Resultado do exercício 2a APD vs AQD **Erro! Marcador não definido.**](#_Toc149672788)

[Figura 9: Resultado do exercício 2b 12](https://uapt33090-my.sharepoint.com/personal/rafael_amorim_ua_pt/Documents/reportAssigment1.docx#_Toc149672789)

[Figura 10: Resultado do exercício 2c 13](#_Toc149672790)

Índice da Tabela

[Tabela 1: Resultado do exercício 1b **Erro! Marcador não definido.**](#_Toc149619897)

# Introdução

De acordo com o solicitado no mini projeto da unidade curricular de Modelação e Desempenho de Redes e Serviços realizou-se este relatório apresentando excertos de código importantes para a explicação do raciocínio e descrevendo de forma sintetizada todas as conclusões retiradas dos resultados de cada exercício.

A estrutura do relatório consiste em duas partes, uma para cada tarefa.

O código do projeto, tal como, toda a gestão de tarefas encontra-se disponível em:

[https://github.com/Tiago-AlvesUA /MDRS/](https://github.com/Tiago-AlvesUA%20/MDRS/)

# Tarefa 1

## Exercício 1a

Não é possível utilizar a solução de encaminhar todo o tráfego pelo caminho com menor atraso de propagação da rede. Cada ligação tem uma capacidade máxima de 100 Gbps, logo, se todo o tráfego fosse encaminhado por um único caminho, o valor da capacidade máxima da ligação seria ultrapassado.

## Exercício 1b

### Código:

O código implementado nesta alínea foi retirado em grande parte do trabalho já realizado nos guiões. Primeiro é aplicado o algoritmo “Greedy Randomized” e com a solução obtida é utilizado o algoritmo “Hill Climbing”. Em ambos os casos é necessário fornecer uma quantidade, **k**, de caminhos mais curtos para cada percurso, calculados previamente com a função “kShortestPath”,sendo que a solução obtida com os algoritmos corresponde ao caminho mais eficiente para o objetivo pretendido.



Como o objetivo é minimizar a carga máxima nas ligações, é possível notar que a cada solução, para que esta seja aceite como melhor e habilitada para substituir a anterior, é verificado se a carga máxima (‘load’) é menor que a melhor carga máxima anteriormente obtida.

A função do algoritmo ”*Greedy Randomized*“ encontra-se presente abaixo:



Para cada fluxo, numa ordem aleatória, são analisados todos os caminhos como possíveis soluções. Se a carga máxima obtida para a solução for menor que a anterior guardada, os resultados são guardados.

Em baixo podemos também observar o código do algoritmo “*Hill Climbing*”:



Por cada fluxo vamos analisar todos os vizinhos. É feita uma cópia da solução, auxsol, e a partir desta conseguimos obter a carga deste vizinho. Se a carga resultante for melhor que a melhor carga anteriormente guardada, de outro vizinho, esta é guardada como nova melhor carga resultante.

Depois de percorrer todos os fluxos se a carga da vizinhança for melhor que a carga atual, ‘load’, esta solução é considerada melhor e é feita a atualização dos restantes valores.

## Exercício 1c

### Código:

Para alguns dos valores pedidos neste exercício, foram necessários alguns cálculos simples, enquanto que para outros a sua obtenção foi praticamente direta. A solução é escolhida de 2 caminhos mais curtos calculados previamente (k=2), num tempo limite de 60 segundos (timeLimit=60). A cada iteração, na ocorrência de mudança da melhor solução, para além do melhor/menor valor de carga máxima, foram também guardados os valores das seguintes soluções:

* Cargas para todas as ligações, denominadas ’bestLoads’
* Energia consumida pelas ligações, ‘bestEnergy’
* Tempo despendido para encontrar a solução, ‘bestTime’

Além disso é incrementado o valor do contador a cada ciclo de relógio.



**Pior carga e carga média das ligações:**

Para obter a pior/maior carga de ligação da solução basta verificar o valor armazenado na variável ‘bestLoad’. Quanto à carga média das ligações basta dividir a soma das cargas das ligações em ‘bestLoads’ pelo número total das ligações, e por 2 já que os valores de ‘bestLoads’ são de ida e volta. REVER

**Energia:**

No que toca à energia, foram necessárias algumas alterações no cálculo de cargas em ambos os algoritmos e foi ainda necessário criar uma função que retorna a energia consumida por cada nó/router da rede.

O código seguinte foi adicionado à função “calculateLinkLoads”:



Ao percorrer as ligações verificamos se a ligação não está a ser usada. Se não estiver, a ligação está em “*sleeping mode*” e a energia despendida pela mesma é 2. Caso contrário é obtido o comprimento entre os dois nós da ligação, utilizando a matriz L. Depois é aplicada a fórmula , onde *l* é o comprimento da ligação. Estes valores são incrementados à energia total da solução.

Uma nota importante para o exercício 2 do projeto é a última verificação. Caso o valor de carga máxima de ligação obtido ultrapasse a capacidade máxima das ligações que é 100 Gbps é atribuído o valor máximo de energia à solução. Assim esta solução será sempre descartada.

Quanto aos algoritmos, garantimos que o valor da energia da solução obtido usando a função acima também fosse guardado.

Por forma a obter a energia total da solução falta ainda calcular a energia consumida nos nós. Criámos, por isso, a função “calculateNodeEnergy”:



Para cada fluxo verificamos qual o caminho escolhido na solução obtida. Dentro do caminho acumulamos, para cada router, o tráfego que é suportado por ele. Depois de obtidos estes valores, é usada a fórmula , onde *t* representa o tráfego suportado por cada router a dividir pela sua capacidade máxima, de 1000 Gbps.

Na tarefa principal, depois de obtida a solução com os algoritmos, basta então calcular a energia dos nós para essa solução e somá-la à energia das ligações já calculada e guardada.



**Média do atraso de propagação de ida e volta** (de cada serviço)**:**

Para obter o atraso médio de propagação de ida e volta do serviço 1 realizámos o seguinte cálculo:



Percorremos todos os fluxos do serviço, e para cada um destes, obtemos o atraso de propagação para o respetivo caminho escolhido pela solução. Este valor é depois multiplicado por 2, por estarmos a calcular o valor do atraso de ida e volta.

Depois de percorrer todos os fluxos é feita a média dos atrasos de propagação.

Para percorrer os fluxos do próximo serviço, como estes se encontram seguidos na solução, começamos a iteração dos fluxos em 1 mais o comprimento dos fluxos anterior.

**Número e lista de ligações que não suportam nenhum tráfego:**

As ligações que não suportam nenhum tráfego são as que não têm tráfego tanto para um lado como para o outro. Portanto, se a soma das colunas 3 e 4 do ‘bestLoads’ for 0, essa ligação é uma das quais não está a suportar nenhum tráfego:



**Número de ciclos de relógio e tempo de obtenção da melhor solução:**

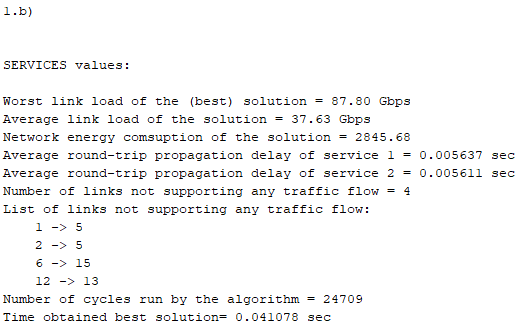
O número de ciclos de relógio é incrementado a cada ciclo da iteração, onde é executado o algoritmo. O tempo para obtenção da melhor solução corresponde ao valor ‘bestTime’, já referido acima.

## Exercício 1d

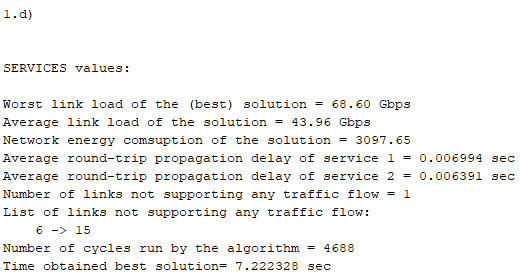
Para o exercício 1 d) apenas foi necessário mudar o valor da variável k para 6, de forma que a melhor solução seja escolhida de entre 6 caminhos mais curtos.

## Exercício 1e

### Resultados:

**Valores para 1 c), k = 2:**

**Valores para 1 d), k = 6:**



### Conclusões:

Começando por analisar os valores obtidos, podemos verificar que o valor da carga máxima de ligação da solução obtida desceu quando utilizamos k = 6 (6 caminhos mais curtos para cada fluxo, calculados à priori). Como a prioridade é obter uma solução que minimize a carga máxima, o fornecimento ao algoritmo de um conjunto maior de caminhos mais curtos para cada fluxo, permite uma maior variedade de escolha e assim encontrar um caminho cuja carga máxima é menor do que a obtida para a solução com k = 2.

Esta otimização do valor da carga máxima, não se traduz numa melhoria equivalente da carga média nas ligações. Isto deve-se ao facto de estarmos focados na procura de uma menor carga máxima e não na redução da carga média de todas as ligações, resultando em valores médios de cargas das ligações em k=6, eventualmente, maiores do que os obtidos em k=2.

Quando otimizamos um certo parâmetro podemos esperar uma diminuição na performance dos restantes como é possível verificar com a energia. Com maior número de caminhos conseguimos diminuir a carga máxima, mas, consequentemente, a energia aumenta.

O tempo de atraso médio de propagação de ida e volta para ambos os serviços aumentou. A matriz com os atrasos de propagação entre cada ligação, D, é diretamente proporcional aos valores de comprimento de ligações, presentes na matriz L. A cada novo caminho mais curto calculado, utilizando a função kShortestPath, obtemos um caminho mais longo, possivelmente até com mais nós e ligações que o anterior. Logo, utilizando k=6, alguns caminhos escolhidos na solução serão maiores, resultando em atrasos de propagação superiores.

O número de ligações que não suportam nenhum tráfego diminui substancialmente para os segundos valores. De acordo com a explicação das consequências do aumento do número de caminhos, explicado no parágrafo de cima, mais ligações poderão estar a ser usadas nos novos caminhos calculados. Como a solução é escolhida de entre qualquer um destes caminhos, o número de ligações inativas diminui.

O número de ciclos de relógio corridos pelo algoritmo diminui porque a solução passa a ter de ser obtida a partir de um número maior de caminhos possíveis. No algoritmo “*Greedy Randomized*” os fluxos e os seus respetivos caminhos são percorridos um a um e no “*Hill Climbing*” também teremos um número maior de vizinhos. Consequentemente, o tempo para percorrer o algoritmo aumentará.

Finalmente, o tempo para obter a melhor solução aumentou significativamente para k=6, sendo que a justificação é a mesma para a diminuição do número de ciclos de relógio. Como o algoritmo demora mais tempo a ser executado por completo, o tempo de obtenção da melhor solução também aumenta.

# Tarefa 2

## Exercício 2a

Neste exercício o algoritmo é parecido ao da alínea Exercício 1b, no entanto, em vez do objetivo ser minimizar a carga máxima das ligações, agora é minimizar a energia consumida da rede. Para alcançar este objetivo, passamos a calcular dentro dos algoritmos não só as energias das ligações, mas também as dos routers.

Recorremos à função ‘calculateLinkLoads’, já descrita na alínea Exercício 1c, para que sempre que a energia dos routers atingir o valor infinito, indicando que excedeu a capacidade máxima por ligação de 100 Gbps, retornamos o mesmo valor, se for menor calculamos a energia gasta pelos routers recorrendo à função ‘calculateNodesEnergy’ para obter a total consumida, por forma a minimizá-la na condição seguinte.

Na otimização armazenamos as soluções que resultam na menor energia consumida, ou seja, caso surja uma nova energia menor que a última guardada como melhor esta substitui os melhores resultados de consumo de energia obtidos.

Outra consideração a ter em conta nestes algoritmos que diferem do exercício anterior é que apenas guardamos a solução se os caminhos encontrados forem maiores que zero, caso contrário retornamos infinito para a energia total.

### Código:



Figura 1: GreedyRandomized\_EnergyOptimized

No algoritmo “*Hill Climbing*” recebemos como argumento os resultados do “*Greedy Randomized*” dando sequencia à otimização, para este adotámos a mesma lógica de fazer a soma de toda a energia consumida tanto dos routers como das ligações. Uma alteração importante efetuada neste algoritmo foi que quando se encontra um vizinho melhor e se atualizam os melhores resultados tem que se ter o cuidado das cargas não ultrapassarem os limites permitidos, se for o caso colocasse a carga com valor infinito e procura-se outra válida, caso contrário guarda-se a maior carga.



Figura 2: HillClimbing\_EnergyOptimization

## Exercício 2 b) c)

Para estas alínea foram criados 2 scripts semelhantes à main da tarefa 1 para podermos aplicar os algoritmos referidos na alínea anterior, assim conseguimos comparar os resultados tanto para k=2 da alínea b) como para k=6 da alínea c).

Em baixo apresentamos essencialmente o excerto de código alterado relação à tarefa 1, salientando a comparação realizada a seguir à aplicação dos algoritmos guardando os valores de energia apenas quando estes forem menores/melhores que os últimos guardados. Outra mudança, é que já não aparece o cálculo da energia dos nós no fim do ciclo porque agora é feito dentro dos algoritmos.

### Código:



## Exercício 2d

Uma Nesta alínea reutilizou-se o código anterior alterando apenas o ‘k’ para 6.

### Resultado:

Figura 4: Resultado do exercício 2c

### Explicação:

Tal como explicamos nas conclusões da alínea Exercício 1e os ‘k’s indicam o número de caminhos candidatos à melhor solução e tanto com 2 como com 6 vemos o resultado mantém se constante quanto à carga máxima de ligação da solução obtida o que é normal porque alterar os números de caminhos mais curtos não vai mudar o pior resultado obtido, pois este nunca vai entrar nos 6 caminhos candidatos.

O mesmo não se pode dizer a favor da média das cargas das ligações, estas com 6 caminhos mais curtos, tende a aumentar pois ligações mais pequenas são preferidas nesta otimização energética e retirando mais valores baixos naturalmente a média sobe.

Com k=6 houve uma diminuição em termos de consumo de energia na rede isto deve-se à prioridade numa solução que minimize a energia consumida, logo quanto maior a disponibilidade desses caminhos candidatos menores os resultados obtidos, como se pode verificar na imagem, com k=2 temos 2375.44 e com k=6 temos 2072.79.

Ao nivel tempo médio dos atrasos de propagação de ida e volta para ambos os serviços sofreu um aumento, este fator acontece pelo simples motivo da variável ‘D’, que representa a propagação de atraso de cada ligação em cada direção dada pelo comprimento a dividir pela velocidade nas fibras (v = 2×105 km/seg), ser proporcional ao comprimento das ligações, logo como se prioriza o menor consumo energético, temos ligações mais pequenas e com ‘k’ maior temos mais caminhos curtos disponíveis daí ser preciso rotas maiores para compensar levando a maiores atrasos.

Em relação ao número de ligações que não suportam qualquer tráfego estes valores aumentam quando o temos mais caminhos candidatos pelos que sabendo que algumas ligações deixam de ser preferidas entram em “modo de espera” por terem alto consumo de energia, alternando mais entre os mesmos caminhos de baixo consumo, estes apesar de terem menores gastos energéticos esta preferência poderá levar ao congestionamento.

O número de ciclos a percorrer os algoritmos preciso para encontra a melhor solução diminui para k=6, pois temos mais caminhos curtos o que é preferido para o baixo consumo de energia, logo “desativa” mais ligações assim recorrem-se a menos ciclos.

Quanto ao tempo demorar muito mais para k=6, isto deve-se ao facto de para além poder encontrar resultados que ultrapassem a capacidade máxima da ligação tendo que fazer uma nova procura para encontrar a melhor solução terá ainda que considerar qual dos 6 caminhos candidatos escolher, o que leva a uma maior demora.

## Exercício 2e

Neste exercício pediu-se para chamar os algoritmos anteriores, usando k = 6, ou seja, sendo previamente calculados os 6 caminhos candidatos para cada fluxo num tempo limite de 60 segundos, à medida que vai comparando as soluções verifica se estas são melhores e atualiza os resultados.

### Resultado:

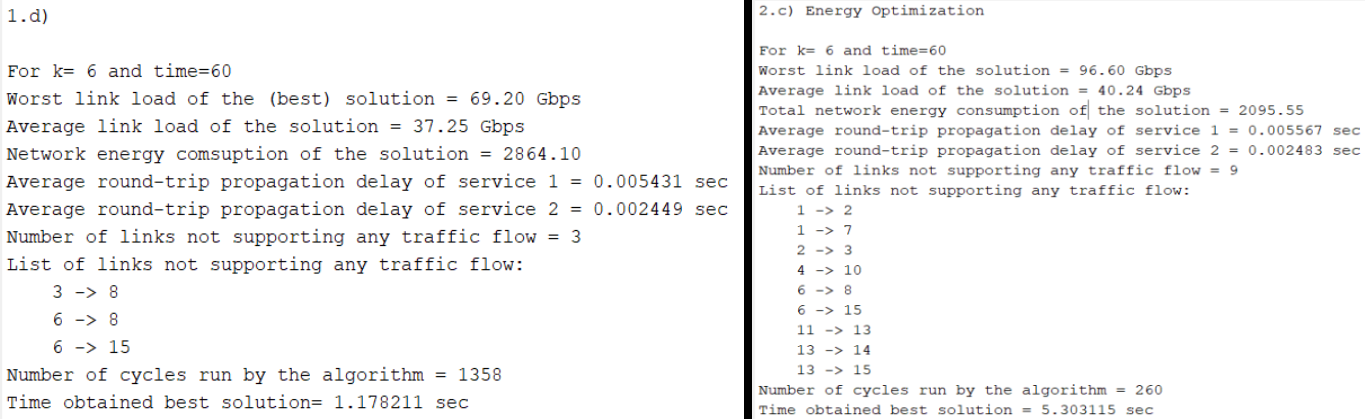


Figura 3: Resultado do exercício 2e

### Explicação:

Ao testar o algoritmo especificado na alínea acima e ao compará-lo com o cenário sem a otimização energética, onde se recorreu ao exercício anterior, observou-se uma diminuição quanto ao consumo total de energia da rede passando de 2864.10 para 2095.55, como o esperado.

A melhoria na eficiência energética da rede resultou, consequentemente, num aumento da carga máxima aproximando-se do limite suportado por cada ligação. O que faz sentido, porque com esta otimização priorizamos a eficiência da energia, independentemente das cargas.

Quanto à média da carga das ligações não há uma diferença significativa, o que é normal, porque apesar da carga máxima aumentar nada implica que a média varie, daí este valor se aproximar ao obtido quando comparado sem a otimização, no entanto, esta também tende a aumentar.

Observando o tempo médio dos atrasos de propagação de ida e volta para ambos os serviços conseguimos ver à partida que com a otimização este valor aumentou ligeiramente, porque como explicamos na alínea anterior com a proporcionalidade do atraso com o comprimento das ligações sabemos que vai preferir caminhos menores logo rotas maiores, consequentemente, levando a um maior atraso total dos serviços.

Existe um aumento do número de ligações que não suportam qualquer tráfego, isso pode ser devido ao algoritmo desativar mais ligações para economizar energia, colocando-os em “modo de espera”.

O número de ciclos a percorrer os algoritmos, para a otimização da energia é consideravelmente menor porque ao preferir caminhos mais curtos como foi explicado 2 parágrafos acima vai acabar por desativar muitas ligações colocando em “modo de espera” o que evita a passagem por estas diminuindo, consequentemente, o número de rotas possíveis diminui, sendo assim necessário menos ciclos para encontrar a solução.

A nível do tempo gasto para encontrar a melhor solução este aumenta para a otimização do consumo de energia, porque neste caso é possível encontrar resultados em que temos resultados indesejados representados por “infinito” o que leva a uma nova procura resultando numa melhor solução.

# Tarefa 3

## Exercício 3a

Para criar o

No final a

## Exercício 3b

Para criar o

No final a

## Exercício 3c

Para criar o

No final a

## Exercício 3d

Para criar o

No final a

## Exercício 3e

Para criar o

No final a

# Tarefa 4

## Exercício 4a

### Código

Com o objetivo de adaptar o algoritmo para também suportar o serviço ‘anycast’ realizámos alterações no código principal e criámos uma nova função “bestCostPaths”.

O código da função está presente abaixo:



O propósito desta função é encontrar o nó de destino, ‘anycast’, para cada um dos nós não ‘anycast’. Os nós são percorridos um a um, sendo que a primeira coisa verificada é se o nó é ‘anycast’ ou não. Se for e se pertencer à matriz ‘anycast’(T3) como nó de origem, o caminho mais curto para o nó, começa e acaba nele mesmo. Se o nó não pertencer à matriz como nó de origem não é calculado caminho mais curto para ele.

Para os nós que são origem e não são nós de destino são calculadas as possibilidades de caminhos mais curtos para os nós ‘anycast’, por forma a que o caminho guardado e o nó de destino escolhido sejam os que proporcionam menor custo.

CONTINUAR COM A MAIN AMANHA

## Exercício 4b

Para criar o

No final a

## Exercício 4c

Para criar o

No final a

## Exercício 4d

Para criar o

No final a

## Exercício 4e

Para criar o

No final a

# Contribuição dos autores

Tiago Alves – 50 %

Rafael Amorim – 50 %