Universidade de Aveiro

Modelação e Desempenho de Redes e Seviços

Mini-Project nr.2

Guilherme Duarte (107766), Guilherme Andrade (107696)

Departamento de Eletrónica, Telecomunicações e Informática

December 14, 2024

Conteúdo

Lista de Figuras

Capítulo 1

Tarefa 1

1.1 Exercício 1.a)

1.1.1 Resultados

```
Anycast nodes = 3 10
Worst round-trip delay (unicast service 1) = 9.04 ms
Average round-trip delay (unicast service 1) = 5.42 ms
Worst round-trip delay (unicast service 2) = 11.07 ms
Average round-trip delay (unicast service 2) = 5.83 ms
Worst round-trip delay (anycast service) = 6.16 ms
Average round-trip delay (anycast service) = 3.43 ms
```

Figura 1:Pior atraso de ida e volta e o atraso médio de ida e volta de cada um dos 3 serviços

1.1.2 Explicação do código

Este código é utilizado para calcular métricas de atraso de propagação para diferentes tipos de serviços numa rede simulada, utilizando informações de rotas e atrasos fornecidas por uma matriz de entrada.

- L: Matriz quadrada com o comprimento de todos os links é fornecida em km.

- T: Cada linha da matriz representa um fluxo de tráfego e possui as seguintes 5 colunas:

    1. Tipo de serviço s ao qual o fluxo de tráfego pertence.
        - Serviço Unicast 1;
        - Serviço Unicast 2;
        - Serviço Anycast;
    2. Define o nó de origem do fluxo de tráfego.
    3. Define o nó de destino do fluxo do tráfego.
    4. Define o throughput(em Gbps) do fluxo no sentido do nó origem para o nó de destino.
    5. Define o throughput(em Gbps) do fluxo no sentido do nó destino para o nó de origem.

Definição de constantes e dos Nós Anycast

A velocidade da luz na fibra(v) é usada para calcular a matriz de atrasos de propagação D, que é obtida dividindo a matriz de distâncias pela velocidade.

Para o serviço do tipo anycast(serviço 3) definimos os nós [3,10] como pedido no exercício.

Inicialização de Variáveis

nFlows: Número total de fluxos a serem processados.

sP: Armazena os caminhos mais curtos para cada fluxo.

nSP: Número de caminhos mais curtos para cada fluxo.

Atrasos: Atraso de propagação (ida e volta) para cada fluxo.

Processamento dos fluxos

a) Serviços Unicast(s = 1 e s = 2)

```
if T(f, 1) == 1 || T(f, 1) == 2  % Unicast services (s = 1 and s = 2)
    [shortestPath, totalCost] = kShortestPath(D, T(f, 2), T(f, 3), 1);
    sP{f} = shortestPath;
    nSP(f) = length(totalCost);
    atrasos(f) = 2 * totalCost;
```

Figura 2: Fluxo dos serviços uñicast

- Para cada fluxo unicast, o nó de origem (T(f,2)) e o nó destino (T(f,3)) são usados para encontrar o caminho mais curto e o seu custo usando a função kShortestPath(dada pelo professor).
- O atraso é calculado como ida e volta(2 * totalCost).

b) Serviços Anycast(s = 3)

```
    elseif T(f, 1) == 3  % Anycast service (s = 3)
        [shortestPath1, totalCost1] = kShortestPath(D, T(f, 2), anycastNodes(1), 1);
        [shortestPath2, totalCost2] = kShortestPath(D, T(f, 2), anycastNodes(2), 1);
        if ismember(T(f,2),anycastNodes)
            sP{f}={T(f,2)};
            nSP(f)=1;
        else
            if totalCost1 < totalCost2
                sP{f} = shortestPath1;
                nSP(f) = length(totalCost1);
                atrasos(f) = 2 * totalCost1;
            else
                sP{f} = shortestPath2;
                nSP(f) = length(totalCost2);
                atrasos(f) = 2 * totalCost2;
            end
        end
    end
end
```

Figura 3: Fluxo do serviço Anycast

- Encontrar os caminhos mais curtos do nó de origem do fluxo(T(f,2)) para os dois nós anycast(anycastNodes(1) e anycastNodes(2)).
- KShortestPath: Função que retorna o menor caminho(shortestPath) e o seu custo total (totalCost) em termos de atraso de propagação.

Verificação se o Nó de Origem é um Nó Anycast

- **Caso o nó de origem(T(f,2)) seja ele próprio um nó anycast.**
  - O caminho mais curto é apenas o nó em si.
  - O número de caminhos (nSP(f)) é definido como 1, indicando que não há necessidade de roteamento.
  - O atraso(atrasos(f)) não é atualizado porque o fluxo não requer propagação.

Seleção do Caminho com Menor Custo

- **Caso o nó de origem não seja um nó anycast:**
  - Compara-se o custo total (totalCost1 e totalCost2) dos dois caminhos calculados para os nós anycast.
  - O caminho com menor custo é selecionado:
    - sP{f} : Atualizado com o caminho mais curto.
    - nSP(f): Atualizado com o número de passos do caminho escolhido.
    - Atrasos(f): Calculando como o **atraso de ida e volta (**2 * totalCost).

Cálculo das métricas de atraso

```
% Separate delays by service type
unicast1Delays = atrasos(T(:, 1) == 1);
unicast2Delays = atrasos(T(:, 1) == 2);
anycastDelays = atrasos(T(:, 1) == 3);

% Compute metrics
worstUnicast1Delay = max(unicast1Delays) * 1000;
averageUnicast1Delay = mean(unicast1Delays) * 1000;
worstUnicast2Delay = max(unicast2Delays) * 1000;
averageUnicast2Delay = mean(unicast2Delays) * 1000;
worstAnycastDelay = max(anycastDelays) * 1000;
averageAnycastDelay = mean(anycastDelays) * 1000;

% Display results
fprintf('Anycast nodes = %d %d\n', anycastNodes(1), anycastNodes(2));
fprintf('Worst round-trip delay (unicast service 1) = %.2f ms\n', worstUnicast1Delay);
fprintf('Average round-trip delay (unicast service 1) = %.2f ms\n', averageUnicast1Delay);
fprintf('Worst round-trip delay (unicast service 2) = %.2f ms\n', worstUnicast2Delay);
fprintf('Average round-trip delay (unicast service 2) = %.2f ms\n', averageUnicast2Delay);
fprintf('Worst round-trip delay (anycast service) = %.2f ms\n', worstAnycastDelay);
fprintf('Average round-trip delay (anycast service) = %.2f ms\n', averageAnycastDelay);
```

Figura 4:Display da métrica dos atrasos

- Calcula-se o maior atraso (pior caso) para cada serviço.
- Calcula-se o atraso médio.
- Os resultados são convertidos para milissegundos (* 1000).

Conclusão:

1. **Nós Anycast Definidos:**
   - Os nós 3 e 10 foram definidos como nós anycast.
2. **Serviços Unicast(s = 1 e s = 2):**
   - **Serviço Unicast 1(s=1):**
     - **Pior atraso de ida e volta:**9,04 ms
     - **Atraso médio de ida e volta:**5,42 ms
     - O serviço unicast 1 apresenta atrasos moderados. O atraso médio é menor, indicando que a maioria dos fluxos deste serviço tem rotas eficientes.
   - **Serviço Unicast 2(s =2):**
     - **Pior atraso de ida e volta:**11,07 ms
     - **Atraso médio de ida e volta:**5,83 ms
     - O serviço unicast 2 tem um pior atraso maior em comparação com o serviço 1, o que sugere que alguns fluxos deste serviço percorrem caminhos mais longos ou têm características de tráfego mais exigentes.

3. **Serviço Anycast (s=3):**
    - **Pior atraso de ida e volta:**6,16 ms
    - **Atraso médio de ida e volta:**3,43 ms
    - Os fluxos anycast apresentam os **menores atrasos** (tanto o pior caso quanto a média) em comparação aos serviços unicast. Isso reflete a eficiência do roteamento anycast, que envia os fluxos para o nó mais próximo, reduzindo atrasos significativamente.

Eficiência Geral da Rede:

- Os atrasos registados são razoáveis para uma rede de fibra óptica.
- O serviço anycast é o mais eficiente, como esperado, devido à flexibilidade de escolha de destinos.
- O maior atraso no serviço unicast 2 pode sugerir que ele utiliza caminhos mais longos ou enfrenta maior congestionamento na rede.

1.2 Exercício 1.b)

1.2.1 Resultados

```
Anycast nodes = 3 10
Worst round-trip delay (unicast service 1) = 9.04 ms
Average round-trip delay (unicast service 1) = 5.42 ms
Worst round-trip delay (unicast service 2) = 11.07 ms
Average round-trip delay (unicast service 2) = 5.83 ms
Worst round-trip delay (anycast service) = 6.16 ms
Average round-trip delay (anycast service) = 3.43 ms
Link Loads (Gbps):
Link 1-2: Forward 15.00 Gbps, Reverse 15.20 Gbps
Link 1-5: Forward 20.30 Gbps, Reverse 26.20 Gbps
Link 1-7: Forward 0.00 Gbps, Reverse 0.00 Gbps
Link 2-3: Forward 48.00 Gbps, Reverse 52.00 Gbps
Link 2-4: Forward 33.10 Gbps, Reverse 34.20 Gbps
Link 2-5: Forward 47.80 Gbps, Reverse 48.90 Gbps
Link 3-6: Forward 31.50 Gbps, Reverse 3.00 Gbps
Link 3-8: Forward 33.10 Gbps, Reverse 31.60 Gbps
Link 4-5: Forward 36.00 Gbps, Reverse 21.00 Gbps
Link 4-8: Forward 34.40 Gbps, Reverse 35.20 Gbps
Link 4-9: Forward 13.40 Gbps, Reverse 15.60 Gbps
Link 4-10: Forward 28.30 Gbps, Reverse 46.90 Gbps
Link 5-7: Forward 47.80 Gbps, Reverse 48.90 Gbps
Link 6-8: Forward 11.60 Gbps, Reverse 9.90 Gbps
Link 6-14: Forward 38.90 Gbps, Reverse 27.80 Gbps
Link 6-15: Forward 8.50 Gbps, Reverse 0.90 Gbps
Link 7-9: Forward 55.50 Gbps, Reverse 71.10 Gbps
Link 8-10: Forward 76.80 Gbps, Reverse 88.00 Gbps
Link 8-12: Forward 14.80 Gbps, Reverse 14.10 Gbps
Link 9-10: Forward 70.30 Gbps, Reverse 98.20 Gbps
Link 10-11: Forward 85.60 Gbps, Reverse 65.50 Gbps
Link 11-13: Forward 61.70 Gbps, Reverse 46.50 Gbps
Link 12-13: Forward 15.00 Gbps, Reverse 17.10 Gbps
Link 12-14: Forward 14.80 Gbps, Reverse 14.10 Gbps
Link 13-14: Forward 40.90 Gbps, Reverse 40.80 Gbps
Link 13-16: Forward 0.00 Gbps, Reverse 0.00 Gbps
Link 14-15: Forward 29.30 Gbps, Reverse 29.40 Gbps
Link 15-16: Forward 0.00 Gbps, Reverse 0.00 Gbps
Worst Link Load: 98.20 Gbps
```

Figura 5: Cargas de todos os links e pior carga de link da solução da rede anterior

1.2.2 Explicação do código

Razão para criar a Matriz Taux

- No caso do serviço anycast, o destino original do fluxo na matriz `T` é `0`. Quando o fluxo é roteado para um dos nós anycast (3 ou 10), o nó destino precisa ser atualizado para refletir o destino final selecionado no roteamento.

- Esse ajuste é feito na matriz `Taux`, preservando os dados originais em `T`.

- Manter a matriz T inalterada é importante para garantir que os dados de entrada permaneçam intactos e possam ser usados para outros cálculos, se necessário.

Código Adicionado para Calcular as Cargas nos Links

- **calculateLinkLoads:** Uma função auxiliar que calcula as cargas em cada link. (dada pelo professor)
  - Número de nós (nNodes).
  - Lista de links (Links)
  - Taux: Matriz ajustada de tráfego.
  - sP: Os caminhos escolhidos para cada fluxo.
  - sol: Variável auxiliar (sol = ones(1, nFlows)) indicando que apenas o primeiro caminho foi usado para todos os fluxos.

- **Extrai as cargas nos links (cargas diretas e reversas nas colunas 3 e 4 da matriz Loads) e encontra o maior valor, que representa a carga mais alta na rede.**

1. **Dimensionamento da Rede:**
   - É importante garantir que nenhum link exceda a capacidade máxima (100 Gbps em cada direção). Ao calcular as cargas nos links, é possível identificar gargalos potenciais.
2. **Análise de Desempenho**:
   - Determinar o **link mais carregado** (carga máxima) é útil para analisar a eficiência da alocação de tráfego e planejar upgrades na rede, caso necessário.
3. **Visibilidade Completa**:
   - Exibir as cargas em todos os links (forward e reverse) fornece informações detalhadas sobre o uso da capacidade, permitindo ajustes mais precisos no roteamento ou configuração da rede.

Conclusão:

1. **Links com Maior Utilização**
    - **Link mais carregado (carga máxima):**
        i. **Link 9-10**: Reverse = 98.20 Gbps (próximo à capacidade máxima de 100 Gbps).
        ii. Este link está **muito próximo da saturação**, o que pode gerar congestionamento e aumentar atrasos em fluxos futuros.

2. **Outros links com altas cargas:**
    - **Link 8-10**: Reverse = 88.00 Gbps.
    - **Link 10-11**: Forward = 85.60 Gbps.
    - **Link 7-9**: Reverse = 71.10 Gbps.
    - **Link 9-10**: Forward = 70.30 Gbps.

3. **Links Subutilizados**
    - **Links com 0.00 Gbps em ambas as direções:**
        - Links: 1-7, 13-16, 15-16.
        - Estes links não estão sendo utilizados, indicando possíveis redundâncias ou problemas na configuração do roteamento que deixam esses links ociosos.
    - **Links com baixas utilizações:**
        - Link 6-15: Reverse = 0.90 Gbps.
        - Link 6-15: Forward = 8.50 Gbps.
        - Apesar de serem utilizados, o tráfego é baixo, indicando pouca relevância para fluxos atuais.

Conclusão:

- A rede **não está bem balanceada**. Alguns links, como **9-10** e **8-10**, estão próximos da capacidade máxima, enquanto outros, como **1-7**, estão completamente ociosos.

- Este desequilíbrio pode indicar que fluxos estão sendo concentrados em certas rotas, deixando alternativas subutilizadas**.**

1.3 Exercício 1.c)

1.3.1 Resultados

```
Best anycast nodes = 1 6
Worst link load = 76.60 Gbps
Worst round-trip delay (unicast service 1) = 9.04 ms
Average round-trip delay (unicast service 1) = 5.42 ms
Worst round-trip delay (unicast service 2) = 11.07 ms
Average round-trip delay (unicast service 2) = 5.83 ms
Worst round-trip delay (anycast service 3) = 6.41 ms
Average round-trip delay (anycast service 3) = 3.02 ms
```

Figura 6: Minimização da pior carga do link

1.3.2 Explicação do Código e Conclusões

`allCombinations = nchoosek(1:nNodes, 2);`

- Esta linha gera todas as combinações possíveis de dois nós (anycast nodes) a partir do conjunto total de nós na rede (`nNodes`).
- O objetivo é testar todas essas combinações e determinar a que minimiza o **pior carregamento (worst link load)**
- Após o roteamento dos fluxos para a combinação atual, a função `calculateLinkLoads` calcula as cargas nos links da rede.
- A **carga máxima (maxLoad)** nos links é determinada com max(max(Loads(:, 3:4))), considerando as direções forward e reverse.

```
if maxLoad < bestWorstLoad
    bestWorstLoad = maxLoad;
    bestCombination = anycastNodes;
    bestUnicast1Delays = atrasos(T(:, 1) == 1);
    bestUnicast2Delays = atrasos(T(:, 1) == 2);
    bestAnycastDelays = atrasos(T(:, 1) == 3);
end
```

Figura 7:Comparação com a Melhor Combinação

- Se a carga máxima da combinação atual (`maxLoad`) for menor que o valor em `bestWorstLoad`, as seguintes atualizações ocorrem:
  - `bestWorstLoad`: Recebe o valor de `maxLoad`.
  - `bestCombination`: Armazena os nós anycast atuais.
  - Os atrasos para os diferentes serviços (unicast e anycast) também são armazenados para a melhor combinação.

Conclusão:

1. **Nós Anycast Selecionados**
    a. **Melhores nós anycast: 1 e 6.**

        i. Esta combinação minimizou a **carga máxima (worst link load)** na rede para **76.60 Gbps**, bem abaixo do limite de capacidade (100 Gbps).

        ii. A escolha de nós **bem distribuídos** na topologia permite que os fluxos anycast sejam balanceados de forma eficiente, aliviando os links mais sobrecarregados.

2. **Carga Máxima nos Links**
    a. **Pior carregamento (worst link load): 76.60 Gbps.**

        i. Este valor está significativamente abaixo do limite de capacidade de 100 Gbps, indicando que a rede não está saturada e tem margem para acomodar fluxos adicionais.

        ii. A redução do pior carregamento (em comparação com valores anteriores que chegaram próximos a 98 Gbps) demonstra a eficácia da redistribuição de fluxos anycast ao escolher bons nós anycast.

3. **Atrasos para Serviço Anycast**

    a. O serviço anycast apresenta os menores atrasos em comparação com os serviços unicast.

    b. O atraso médio (3.02 ms) demonstra a eficiência do roteamento anycast, que direciona fluxos para o nó anycast mais próximo, reduzindo significativamente os tempos de propagação.

    c. O pior atraso (6.41 ms) está dentro de um intervalo aceitável, indicando que a escolha de nós anycast (1 e 6) foi eficaz.

1.4 Exercícios 1.d)

1.4.1 Resultados

```
Best anycast nodes = 4 12
Worst round-trip delay (anycast service 3) = 4.42 ms
Worst link load = 76.60 Gbps
Worst round-trip delay (unicast service 1) = 9.04 ms
Average round-trip delay (unicast service 1) = 5.42 ms
Worst round-trip delay (unicast service 2) = 11.07 ms
Average round-trip delay (unicast service 2) = 5.83 ms
Average round-trip delay (anycast service 3) = 2.90 ms
```

Figura 8: Minimiza a pior viagem de ida e volta atraso do serviço

1.4.2 Explicação de código e conclusões

```
anycastDelays = atrasos(T(:, 1) == 3);
worstAnycastDelay = max(anycastDelays) * 1000;
```

Figura 9:Pior atraso de ida e volta entre os fluxos anycast

```
if averageAnycastDelay < bestAverageDelay
    bestAverageDelay = averageAnycastDelay;
    bestCombination = anycastNodes;
    bestUnicast1Delays = atrasos(T(:, 1) == 1);
    bestUnicast2Delays = atrasos(T(:, 1) == 2);
    bestAnycastDelays = anycastDelays;
    bestSP = sP;
end
```

Figura 10:Armazena o melhor pior atraso de ida e volta

- **worstAnycastDelay**: O pior atraso de ida e volta calculado para a combinação atual de nós anycast.
- **bestWorstDelay**: O menor valor de pior atraso registado até o momento.
- A condição verifica se o **pior atraso para a combinação atual é menor** do que o melhor encontrado até agora.
- Se a condição for verdadeira (ou seja, a combinação atual tem menor pior atraso), atualizamos a variável **bestWorstDelay** com o valor de **worstAnycastDelay** da combinação atual.
- Armazena a combinação atual de nós anycast (**anycastNodes**) como a **melhor combinação**.

1. **Métricas do Serviço Anycast(S=3):**
    a. Pior Atraso (Worst Round-Trip Delay)
        i. **4.42 ms**:
            1. Esse é o **pior caso de atraso** entre todos os fluxos do serviço anycast.
            2. Comparado com os resultados anteriores (por exemplo, **6.16 ms** ou **6.41 ms**), esse é o melhor desempenho registado, indicando que a escolha dos nós **4 e 12** como anycast otimizou as rotas.

2. **Problema de Carga nos Links**:
    a. O **pior carregamento (98.20 Gbps)** é um ponto crítico. Apesar do ótimo desempenho em atrasos, a rede está operando muito próxima da saturação em pelo menos um link.
    b. Isso pode comprometer a robustez da rede em cenários de aumento de tráfego ou falhas em outros links.

1.5 Exercícios 1.e)

1.5.1 Resultados

```
Best anycast nodes = 5 14
Average round-trip delay (anycast service 3) = 2.52 ms
Worst link load = 76.60 Gbps
Worst round-trip delay (unicast service 1) = 9.04 ms
Average round-trip delay (unicast service 1) = 5.42 ms
Worst round-trip delay (unicast service 2) = 11.07 ms
Average round-trip delay (unicast service 2) = 5.83 ms
Worst round-trip delay (anycast service 3) = 4.90 ms
```

Figura 11:Minimiza o atraso médio de ida e volta do serviço anycast

1.5.2 Explicação de código e conclusões

```
    anycastDelays = atrasos(T(:, 1) == 3);
    averageAnycastDelay = mean(anycastDelays) * 1000; % Convert to milliseconds

    if averageAnycastDelay < bestAverageDelay
        bestAverageDelay = averageAnycastDelay;
        bestCombination = anycastNodes;
        bestUnicast1Delays = atrasos(T(:, 1) == 1);
        bestUnicast2Delays = atrasos(T(:, 1) == 2);
        bestAnycastDelays = anycastDelays;
        bestSP = sP;
    end
end
```

Figura 12:Melhor pior atraso médio de ida e volta

- **averageAnycastDelay**: O **atraso médio de ida e volta** calculado para os fluxos do serviço anycast na combinação atual de nós anycast.
- **bestAverageDelay**: O menor valor de **atraso médio** registado até o momento.
- A condição verifica se o **atraso médio** para a combinação atual é menor do que o melhor encontrado até agora.
- Se a condição for verdadeira (ou seja, a combinação atual tem menor **atraso médio**), atualizamos a variável **bestAverageDelay** com o valor de **averageAnycastDelay** da combinação atual.
- Armazena a combinação atual de nós anycast (**anycastNodes**) como a melhor combinação.

Conclusão:

1. **Métricas do Serviço Anycast(S=3):**
    a. Atraso Médio (Average Round-Trip Delay)
        i. **2.52 ms**:
            1. Este é o valor **médio** de ida e volta para os fluxos do serviço anycast.
            2. Comparado com valores anteriores (por exemplo, **3.43 ms** ou **2.90 ms**), esse é o **melhor desempenho médio registado**, indicando que a escolha dos nós **5 e 14** como anycast proporcionou rotas muito eficientes para a maioria dos fluxos.

2. **Carga nos Links**:
    a. 76.60 Gbps.
    b. Comparado com valores anteriores (como **98.20 Gbps**), há uma melhoria significativa, indicando que a escolha dos nós **5 e 14** também ajudou a distribuir melhor o tráfego entre os links.
    c. Este valor está longe da saturação (100 Gbps), tornando a rede mais robusta a cenários de aumento de tráfego ou falhas em outros links.

1.7 Exercícios 1.f)

1.7.1 Conclusão

Métrica	3 e 10	1 e 6	4 e 12	5 e 14
Pior atraso(ms)(unicast 1)	9.04	9.04	9.04	9.04
Atraso médio(ms)(unicast 1)	5.42	5.42	5.42	5.42
Pior atraso(ms)(unicast 2)	11.07	11.07	11.07	11.07
Atraso médio(ms)(unicast 2)	5.83	5.83	5.83	5.83
Pior atraso(ms)	6.16	6.41	4.42	4.90
Atraso médio(ms)	3.43	3.02	2.90	2.52
Pior carregamento (Gbps)	98.20	76.60	76.60	76.60

1. **Melhor Configuração para Latência Média**:
   - A combinação **5 e 14** é a melhor escolha para fluxos que priorizam **eficiência média**, com um atraso médio de **2.52 ms**.
2. **Melhor Configuração para Latência Extrema (Pior Caso)**:
   - A combinação **4 e 12** é a melhor escolha para fluxos críticos que necessitam de **baixa latência no pior caso**, com um atraso de **4.42 ms**.
3. **Configuração Mais Robusta (Carregamento da Rede)**:
   - Tanto **1 e 6** quanto **5 e 14 e 4 e 12** são equilibradas, apresentando **carga máxima de 76.60 Gbps**, garantindo maior robustez e flexibilidade para cenários de aumento de tráfego.
4. **Desempenho Geral**:
   - **5 e 14** é a melhor configuração **geral**, equilibrando o **melhor atraso médio (2.52 ms)** e um **pior carregamento aceitável (76.60 Gbps)**.

Capítulo 2

Tarefa 2

Exercício 2.a)

2.1.1 Resultados

```
Multi start hill climbing with greedy randomized (specific nodes 3 and 10 for anycast):
     W = 74.60 Gbps, No. sol = 7037, Av. W = 79.85, time = 5.77 sec
Total number of cycles run: 7037
Time when the best solution was found: 5.77 sec
Cycle when the best solution was found: 1331
Anycast nodes = 3 10
Worst round-trip delay (unicast service 1) = 11.60 ms
Average round-trip delay (unicast service 1) = 8.03 ms
Worst round-trip delay (unicast service 2) = 12.30 ms
Average round-trip delay (unicast service 2) = 7.83 ms
Worst round-trip delay (anycast service 3) = 6.16 ms
Average round-trip delay (anycast service 3) = 3.43 ms
```

Figura 13:Multi start hill climbing with greedy randomized (specific nodes 3 and 10 for anycast)

2.1.2 Explicação do Código

Anycast selecionados [3, 10].

O cálculo dos caminhos mais curtos e as métricas são calculadas da mesma maneira que na task 1, a principal diferença é a utilização de 6 caminhos k=6.

Multi-start Hill Climbing com Abordagem Gulosa Randomizada

- **Objetivo**: Encontrar a melhor solução para distribuição de tráfego minimizando a carga da rede.
- `greedyRandomizedStrategy`: Gera uma solução inicial utilizando uma abordagem gulosa com randomização.
    - A função considera **todos os caminhos candidatos** para um fluxo.
    - Calcula a **carga máxima nos links** caso o fluxo utilize cada um dos caminhos candidatos.
    - Introduz um elemento de **aleatoriedade** ao selecionar aleatoriamente um dos **três melhores caminhos**.
    - Isso permite evitar situações em que uma abordagem puramente gulosa fique presa em ótimos locais.
- **HillClimbingStrategy:** Otimiza a solução inicial localmente usando *Hill Climbing*
    - **Hill Climbing** funciona como uma busca local: tenta mudar a solução atual fazendo pequenas alterações (testando outros caminhos para cada fluxo).
    - Aceita somente as mudanças que **reduzem a carga máxima nos links**.
    - O algoritmo **pára** quando **nenhuma melhoria adicional é possível**.
- Atualiza a melhor solução encontrada até o tempo-limite.

```
clear all
close all
clc

load('InputDataProject2.mat')

% Constants
v = 2e5; % Speed of light in fiber (km/s)
D = L / v; % Propagation delay matrix (in seconds)

K = 6; % Number of shortest paths
anycastNodes = [3, 10];
nNodes = size(Nodes, 1);
nLinks = size(Links, 1);
nFlows = size(T, 1);

% Preallocation
delays = cell(nFlows, 1);
sP = cell(1, nFlows);
nSP = zeros(1, nFlows);
Taux = zeros(nFlows, 4);

% Compute shortest paths for each flow and fill Taux
for f = 1:nFlows
    if T(f, 1) == 1 || T(f, 1) == 2  % Unicast services
        [shortestPath, totalCost] = kShortestPath(D, T(f, 2), T(f, 3), K);
        sP{f} = shortestPath;
        nSP(f) = length(shortestPath);
        delays{f} = totalCost; % Store all delay values for paths
        Taux(f, :) = T(f, 2:5);
    elseif T(f, 1) == 3 % Anycast service
        Taux(f, :) = T(f, 2:5);
        % Caso o nó origem já seja um anycast node
        if ismember(T(f, 2), anycastNodes)
            sP{f} = {T(f, 2)};
            nSP(f) = 1;
            Taux(f, 2) = T(f, 2);
            delays{f} = 0; % Sem caminho real, atraso 0 pois o destino é o próprio nó
        else
            [shortestPath1, totalCost1] = kShortestPath(D, T(f, 2), anycastNodes(1), 1);
            [shortestPath2, totalCost2] = kShortestPath(D, T(f, 2), anycastNodes(2), 1);

            % Comparar os custos totais e escolher o menor
            if totalCost1 < totalCost2
                sP{f} = shortestPath1;
                nSP(f) = length(shortestPath1);
                delays{f} = totalCost1; % Store all delay values for the chosen path
                Taux(f, 2) = anycastNodes(1);
            else
                sP{f} = shortestPath2;
                nSP(f) = length(shortestPath2);
                delays{f} = totalCost2; % Store all delay values for the chosen path
                Taux(f, 2) = anycastNodes(2);
            end
        end
    end
end
```

Figura 14: Cálculo dos caminhos mais curtos

```
% Multi-start hill climbing with greedy randomized approach
t = tic;
timeLimit = 30;
bestLoad = inf;
contador = 0;
somador = 0;
bestCycle = 0;

while toc(t) < timeLimit
    sol = greedyRandomizedStrategy(nNodes, Links, Taux, sP, nSP);

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

    if load < bestLoad
        bestSol = sol;
        bestLoad = load;
        bestLoadTime = toc(t);
        bestCycle = contador;
    end
    contador = contador + 1;
    somador = somador + load;
end

% Display results
fprintf('Multi start hill climbing with greedy randomized (specific nodes 3 and 10 for anycast):\n');
fprintf('\t W = %.2f Gbps, No. sol = %d, Av. W = %.2f, time = %.2f sec\n', bestLoad, contador, somador / contador, bestLoadTime);
fprintf('Total number of cycles run: %d\n', contador);
fprintf('Time when the best solution was found: %.2f sec\n', bestLoadTime);
fprintf('Cycle when the best solution was found: %d\n', bestCycle);

% Compute delay metrics by service type
unicast1Delays = [delays{T(:, 1) == 1}];
unicast2Delays = [delays{T(:, 1) == 2}];
anycastDelays = [delays{T(:, 1) == 3}];

% Compute metrics for unicast service 1
worstUnicast1Delay = max(unicast1Delays) * 2 * 1000; % Round-trip delay, convert to ms
averageUnicast1Delay = mean(unicast1Delays) * 2 * 1000; % Round-trip delay, convert to ms

% Compute metrics for unicast service 2
worstUnicast2Delay = max(unicast2Delays) * 2 * 1000; % Round-trip delay, convert to ms
averageUnicast2Delay = mean(unicast2Delays) * 2 * 1000; % Round-trip delay, convert to ms

% Compute metrics for anycast service
worstAnycastDelay = max(anycastDelays) * 2 * 1000; % Round-trip delay, convert to ms
averageAnycastDelay = mean(anycastDelays) * 2 * 1000; % Round-trip delay, convert to ms

% Display delay results
fprintf('Anycast nodes = %d %d\n', anycastNodes(1), anycastNodes(2));
fprintf('Worst round-trip delay (unicast service 1) = %.2f ms\n', worstUnicast1Delay);
fprintf('Average round-trip delay (unicast service 1) = %.2f ms\n', averageUnicast1Delay);
fprintf('Worst round-trip delay (unicast service 2) = %.2f ms\n', worstUnicast2Delay);
fprintf('Average round-trip delay (unicast service 2) = %.2f ms\n', averageUnicast2Delay);
fprintf('Worst round-trip delay (anycast service 3) = %.2f ms\n', worstAnycastDelay);
fprintf('Average round-trip delay (anycast service 3) = %.2f ms\n', averageAnycastDelay);
```

Figura 15:Multi-start hill climbing with greedy randomized

```
function [sol, load] = HillClimbingStrategy(nNodes, Links, T, sP, nSP, sol)
    nFlows = length(sol);
    improved = true;

    while improved
        improved = false;
        currentLoad = calculateLinkLoads(nNodes, Links, T, sP, sol);
        currentMaxLoad = max(max(currentLoad(:,3:4)));

        for f = 1:nFlows
            for p = 1:nSP(f)
                if p ~= sol(f)
                    newSol = sol;
                    newSol(f) = p;
                    newLoad = calculateLinkLoads(nNodes, Links, T, sP, newSol);
                    newMaxLoad = max(max(newLoad(:,3:4)));

                    if newMaxLoad < currentMaxLoad
                        sol = newSol;
                        currentMaxLoad = newMaxLoad;
                        improved = true;
                        break;
                    end
                end
            end
            if improved
                break;
            end
        end
    end

    load = currentMaxLoad;
end
```

Figura 16:HillClimbingStrategy

```
function sol = greedyRandomizedStrategy(nNodes, Links, T, sP, nSP)
    nFlows = size(T, 1);
    sol = zeros(1, nFlows);

    for f = 1:nFlows
        candidatePaths = 1:nSP(f);
        pathLoads = zeros(1, nSP(f));
        for p = candidatePaths
            tempSol = sol;
            tempSol(f) = p;
            Loads = calculateLinkLoads(nNodes, Links, T, sP, tempSol);
            pathLoads(p) = max(max(Loads(:,3:4)));
        end
        [~, sortedIndices] = sort(pathLoads);
        if isempty(sortedIndices)
            selectedPath = 1;
        else
            selectedPath = sortedIndices(randi(min(3, nSP(f))));
        end
        sol(f) = selectedPath;
    end
end
```

Figura 17: greedyRandomizedStrategy

2.2 Exercício 2.b)

2.2.1 Resultados

```
Multi start hill climbing with greedy randomized (specific nodes 1 and 6 for anycast):
     W = 60.00 Gbps, No. sol = 5103, Av. W = 66.07, time = 0.04 sec
Total number of cycles run: 5103
Time when the best solution was found: 0.04 sec
Cycle when the best solution was found: 0
Anycast nodes = 1 6
Worst round-trip delay (unicast service 1) = 11.60 ms
Average round-trip delay (unicast service 1) = 8.03 ms
Worst round-trip delay (unicast service 2) = 12.30 ms
Average round-trip delay (unicast service 2) = 7.83 ms
Worst round-trip delay (anycast service 3) = 6.41 ms
Average round-trip delay (anycast service 3) = 3.02 ms
```

Figura 18:Multi start hill climbing with greedy randomized (specific nodes 1 and 6 for anycast)

2.2.2 Código

A única mudança realizada no código foi a alteração dos nós anycast[1,6].

2.3 Exercício 2.c)

2.3.1 Resultados

```
Multi start hill climbing with greedy randomized (specific nodes 4 and 12 for anycast):
     W = 62.60 Gbps, No. sol = 4091, Av. W = 68.01, time = 0.22 sec
Total number of cycles run: 4091
Time when the best solution was found: 0.22 sec
Cycle when the best solution was found: 21
Anycast nodes = 4 12
Worst round-trip delay (unicast service 1) = 11.60 ms
Average round-trip delay (unicast service 1) = 8.03 ms
Worst round-trip delay (unicast service 2) = 12.30 ms
Average round-trip delay (unicast service 2) = 7.83 ms
Worst round-trip delay (anycast service 3) = 4.42 ms
Average round-trip delay (anycast service 3) = 2.90 ms
```

Figura 19:Multi start hill climbing with greedy randomized (specific nodes 4 and 12 for anycast)

2.3.2 Código

A única mudança realizada no código foi a alteração dos nós anycast[4,12].

2.4 Exercício 2.d)

2.4.1 Resultados

```
Multi start hill climbing with greedy randomized (specific nodes 5 and 14 for anycast):
     W = 60.00 Gbps, No. sol = 4484, Av. W = 67.11, time = 0.11 sec
Total number of cycles run: 4484
Time when the best solution was found: 0.11 sec
Cycle when the best solution was found: 6
Anycast nodes = 5 14
Worst round-trip delay (unicast service 1) = 11.60 ms
Average round-trip delay (unicast service 1) = 8.03 ms
Worst round-trip delay (unicast service 2) = 12.30 ms
Average round-trip delay (unicast service 2) = 7.83 ms
Worst round-trip delay (anycast service 3) = 4.90 ms
Average round-trip delay (anycast service 3) = 2.52 ms
```

Figura 20:Multi start hill climbing with greedy randomized (specific nodes 5 and 14 for anycast)

2.4.2 Código

3 A única mudança realizada no código foi a alteração dos nós anycast[5,14].

2.5 Exercício 2.e)

2.5.1 Conclusão

Métrica	3 e 10	1 e 6	4 e 12	5 e 14
Pior atraso(ms)(unicast 1)	11.60	11.60	11.60	11.60
Atraso médio(ms)(unicast 1)	8.03	8.03	8.03	8.03
Pior atraso(ms)(unicast 2)	12.30	12.30	12.30	12.30
Atraso médio(ms)(unicast 2)	7.83	7.83	7.83	7.83
Pior atraso(ms)(anycast)	6.16	6.41	4.42	4.90
Atraso médio(ms)(anycast)	3.43	3.02	2.90	2.52
Pior carregamento sem Multi-Start(Gbps)	98.20	76.60	76.60	76.60
Pior carregamento com Multi-Start(Gbps)	74.60	60	62.60	60

Atrasos (ms):

Unicast Serviço 1 e 2:

- O aumento de **k** para 6 causou um aumento significativo tanto no pior atraso quanto no atraso médio.
- A busca por até 6 caminhos permitiu a utilização de rotas alternativas mais longas, resultando em um aumento de até **30-50% nos atrasos** médios e máximos para serviços unicast.

Anycast (Serviço 3):

- Isso ocorre porque o número de caminhos avaliados para anycast continua limitado aos dois possíveis destinos (nós 3 e 10), e a escolha final é baseada no menor custo, mantendo a consistência nos atrasos.

Carregamento de Enlaces (Gbps):

- Código Original (k=1, Sem Multi-Start):
  - Pior carregamento: ~**98,20 Gbps**
  - Valor médio: ~76,60 Gbps
- Código com Multi-Start (k=6, Com Multi-Start):
  - Pior carregamento sem Multi-Start: ~**76,60 Gbps**
  - Pior carregamento com Multi-Start: ~**60 Gbps**

- O Multi-Start combinado com o aumento de **k** proporcionou uma **redução significativa no pior carregamento** (de 98,20 Gbps para 60 Gbps).
- Isso evidencia uma melhora significativa no balanceamento da carga, que é um dos principais objetivos do Multi-Start.
- O menor pico de carregamento demonstra que a rede está sendo utilizada de forma mais eficiente, com menor risco de congestionamento em enlaces específicos.

Capítulo 3

Tarefa 3

3.1 Exercícios 3.a)

3.1.1 Resultados

```
Multi start hill climbing with greedy randomized, anycast in nodes 3 and 10:
W = 96.10 Gbps, No. sol = 108, Av. W = 103.57, time = 2.97 sec
Unicast 1 - Worst round-trip delay: 13.97 ms
Unicast 1 - Average round-trip delay: 9.85 ms
Unicast 2 - Worst round-trip delay: 25.02 ms
Unicast 2 - Average round-trip delay: 16.18 ms
Anycast - Worst round-trip delay: 6.16 ms
Anycast - Average round-trip delay: 3.74 ms
```

Figura 21:Multi start hill climbing with greedy randomized (specific nodes 3 and 10 for anycast)

3.1.2 Explicação das alterações no código

```
for f = 1:nFlows
    if T(f, 1) == 1  % Tipo do fluxo: Unicast 1
        [shortestPath, totalCost] = kShortestPath(D, T(f, 2), T(f, 3), k);
        sP{1, f} = shortestPath;
        sP{2, f} = {};
        nSP(f) = length(shortestPath{1});
        pathsNumbered = length(totalCost);
        nPd(f) = pathsNumbered;
        delays{f} = totalCost;
        Taux(f, :) = T(f, 2:5);
    elseif T(f, 1) == 2  % Tipo do fluxo: Unicast 2
        [shortestPath, secondPath, totalCost] = kShortestPathPairs(D, T(f, 2), T(f, 3), k);
        sP{1, f} = shortestPath;
        sP{2, f} = secondPath;
        nSP(f) = length(shortestPath{1});
        pathsNumbered = length(totalCost);
        nPd(f) = pathsNumbered;
        delays{f} = totalCost;
        Taux(f, :) = T(f, 2:5);
    elseif T(f, 1) == 3  % Tipo do fluxo: Anycast
        if ismember(T(f, 2), anycast_nodes)
            sP{1, f} = {T(f, 2)};
            sP{2, f} = {};
            nSP(f) = 1;
            Taux(f, :) = T(f, 2:5);
            Taux(f, 2) = T(f, 2);
        else
            cost = inf;
            Taux(f, :) = T(f, 2:5);
            for i = anycast_nodes
                [shortestPath, totalCost] = kShortestPath(D, T(f, 2), i, 1);
                pathsNumbered = length(totalCost);
                nPd(f) = pathsNumbered;
                if max(totalCost) < cost
                    sP{1, f} = shortestPath;
                    nSP(f) = 1;
                    cost = totalCost;
                    delays{f} = totalCost;
                    Taux(f, 2) = i;
                end
            end
        end
    end
end
```

Figura 22: Serviço Unicast 2 alterações

Introdução do Serviço Unicast 2 com Caminhos Primário e Secundário

Cálculo de Caminhos Primário e Secundário

- Foi implementada uma nova função chamada `kShortestPathPairs`, que calcula pares de caminhos **primário** e **secundário** entre os nós de origem e destino, respeitando as seguintes características:
  - Os dois caminhos devem ser disjuntos ou parcialmente disjuntos para evitar sobreposição.
  - São calculados até `k` pares de caminhos.

Armazenamento dos Caminhos

- Para armazenar esses pares de caminhos, utiliza-se uma célula dupla:
  - `sP{f}{1}`: Caminho primário do fluxo `f`.
  - `sP{f}{2}`: Caminho secundário do fluxo `f`.

Seleção do Melhor Par

- O custo total para cada par de caminhos (primário e secundário) é armazenado.
- O par com o menor custo total é selecionado usando `min(totalPairCosts)`.

Nas funções greedyRandomizedStrategy e HillClimbingStrategy alteramo-las para utilizarem a função CalculateLinkBand1to1 (dada pelo professor).

3.2 Exercícios 3.b)

3.2.1 Resultados

```
Multi start hill climbing with greedy randomized, anycast in nodes 1 and 6:
W = 96.10 Gbps, No. sol = 88, Av. W = 99.91, time = 0.69 sec
Unicast 1 - Worst round-trip delay: 13.97 ms
Unicast 1 - Average round-trip delay: 9.85 ms
Unicast 2 - Worst round-trip delay: 25.02 ms
Unicast 2 - Average round-trip delay: 16.18 ms
Anycast - Worst round-trip delay: 6.41 ms
Anycast - Average round-trip delay: 3.63 ms
```

Figura 23:Multi start hill climbing with greedy randomized (specific nodes 1 and 6 for anycast)

3.1.2 Código

A única mudança realizada no código foi a alteração dos nós anycast[1,6].

3.3 Exercise 3.c)

3.3.1 Resultados

```
Multi start hill climbing with greedy randomized, anycast in nodes 4 and 12:
W = 96.10 Gbps, No. sol = 131, Av. W = 97.64, time = 1.03 sec
Unicast 1 - Worst round-trip delay: 13.97 ms
Unicast 1 - Average round-trip delay: 9.85 ms
Unicast 2 - Worst round-trip delay: 25.02 ms
Unicast 2 - Average round-trip delay: 16.18 ms
Anycast - Worst round-trip delay: 4.42 ms
Anycast - Average round-trip delay: 2.90 ms
```

Figura 24:Multi start hill climbing with greedy randomized (specific nodes 4 and 12 for anycast)

3.3.2 Código

A única mudança realizada no código foi a alteração dos nós anycast[4,12].

3.4 Exercise 3.d)

3.4.1 Resultados

```
Multi start hill climbing with greedy randomized, anycast in nodes 5 and 14:
W = 96.10 Gbps, No. sol = 147, Av. W = 99.28, time = 2.07 sec
Unicast 1 - Worst round-trip delay: 13.97 ms
Unicast 1 - Average round-trip delay: 9.85 ms
Unicast 2 - Worst round-trip delay: 25.02 ms
Unicast 2 - Average round-trip delay: 16.18 ms
Anycast - Worst round-trip delay: 4.90 ms
Anycast - Average round-trip delay: 3.03 ms
```

Figura 25:Multi start hill climbing with greedy randomized (specific nodes 5 and 14 for anycast)

3.4.2 Código

A única mudança realizada no código foi a alteração dos nós anycast[5,14].

3.5 Exercício 3.e)

3.5.1 Conclusão

Métrica	3 e 10	1 e 6	4 e 12	5 e 14
Pior atraso(ms)(unicast 1)	13.97	13.97	13.97	13.97
Atraso médio(ms)(unicast 1)	9.85	9.85	9.85	9.85
Pior atraso(ms)(unicast 2)	25.02	25.02	25.02	25.02
Atraso médio(ms)(unicast 2)	16.18	16.18	16.18	16.18
Pior atraso(ms)	6.16	6.41	4.42	4.90
Atraso médio(ms)	3.43	3.02	2.90	2.52
Pior carregamento (Gbps)	96.10	96.10	96.10	96.10

Tarefa 1:

- **Pior e melhor configuração de nós anycast**:
  - A escolha dos nós anycast impacta diretamente o pior carregamento dos links e os atrasos.
  - A configuração padrão com nós 3 e 10 apresenta um equilíbrio razoável entre atraso e carga de link.
  - Configurações alternativas, como 4 , 12 e 5 , 14, mostraram uma redução significativa nos atrasos do serviço anycast.
- **Unicast vs. Anycast:**
  - Os fluxos unicast têm atrasos menores.

Tarefa 2:

- **Impacto da Otimização:**
  - Houve uma melhoria significativa na carga máxima do link, especialmente para configurações como `5 e 14`, com pior carregamento reduzido para **60 Gbps**.
  - O uso do Multi-Start Hill Climbing distribuiu melhor os fluxos, evitando hotspots na rede.
- **Diferenças no Atraso**:
  - Embora a otimização de carregamento tenha impacto positivo, os atrasos dos fluxos unicast 1 e 2 aumentaram ligeiramente, com o **pior atraso para unicast 1 alcançando 11.60 ms**.
- **Configuração ideal:**
  - Configurações como `4 e 12` destacam-se por equilibrar carga de link e atrasos, indicando uma solução próxima do ideal.

Tarefa 3:

- **Proteção para Unicast 2**
  - Nesta etapa, foi introduzida a proteção de 1:1 para os fluxos do serviço Unicast 2, adicionando caminhos de backup.
- **Impacto nos Atrasos**:
  - O atraso médio para Unicast 2 aumentou consideravelmente devido à reserva de recursos para caminhos de backup, atingindo **25.02 ms no pior cenário**.
  - Unicast 1 também sofreu aumento no atraso, mas de forma mais moderada.
- **Carregamento Máximo**:
  - O carregamento máximo da rede subiu ligeiramente em comparação à Tarefa 2 , refletindo a sobrecarga dos caminhos de proteção, mas permaneceu controlado (~96.10 Gbps para todas as configurações).
- **Impacto das Configurações:**
  - Configurações como `4 e 12` e `5 e 14` continuam a ser as melhores escolhas, fornecendo um equilíbrio aceitável entre carga e atraso.
  - A configuração padrão `3 e 10` mostrou ser menos eficiente nesta etapa.

Capítulo 4
Conclusão e informação

Com este projeto conseguimos analisar estratégias de roteamento e otimização para uma rede MPLS com serviços Unicast e Anycast evidenciou a importância de equilibrar atrasos, carregamento e resiliência em redes complexas, mostrando soluções eficazes para engenharia de tráfego.

Serviço Unicast:

- O Unicast 1 apresentou atrasos baixos e consistentes, mas maior concentração de tráfego em alguns links.

- O Unicast 2, com proteção 1:1, aumentou os atrasos e o carregamento devido aos caminhos de backup, mas garantiu maior resiliência.

Serviço Anycast:

- A escolha dos nós anycast foi crucial para atrasos e carregamento.

- Configurações como `4 e 12` reduziram significativamente atrasos e congestionamentos, destacando-se como a melhor opção.

Otimização:

- O algoritmo Multi-Start Hill Climbing reduziu o carregamento máximo dos links (~60 Gbps nas melhores configurações) e distribuiu o tráfego de forma eficiente, mesmo com aumento leve nos atrasos.

- A proteção no Unicast 2 trouxe trade-offs entre desempenho e resiliência.

Configuração Ideal:

- A configuração dos nós anycast `4 e 12` foi a mais equilibrada, garantindo menor carregamento e atrasos otimizados.

A contribuição entre os membros do grupo é igual.