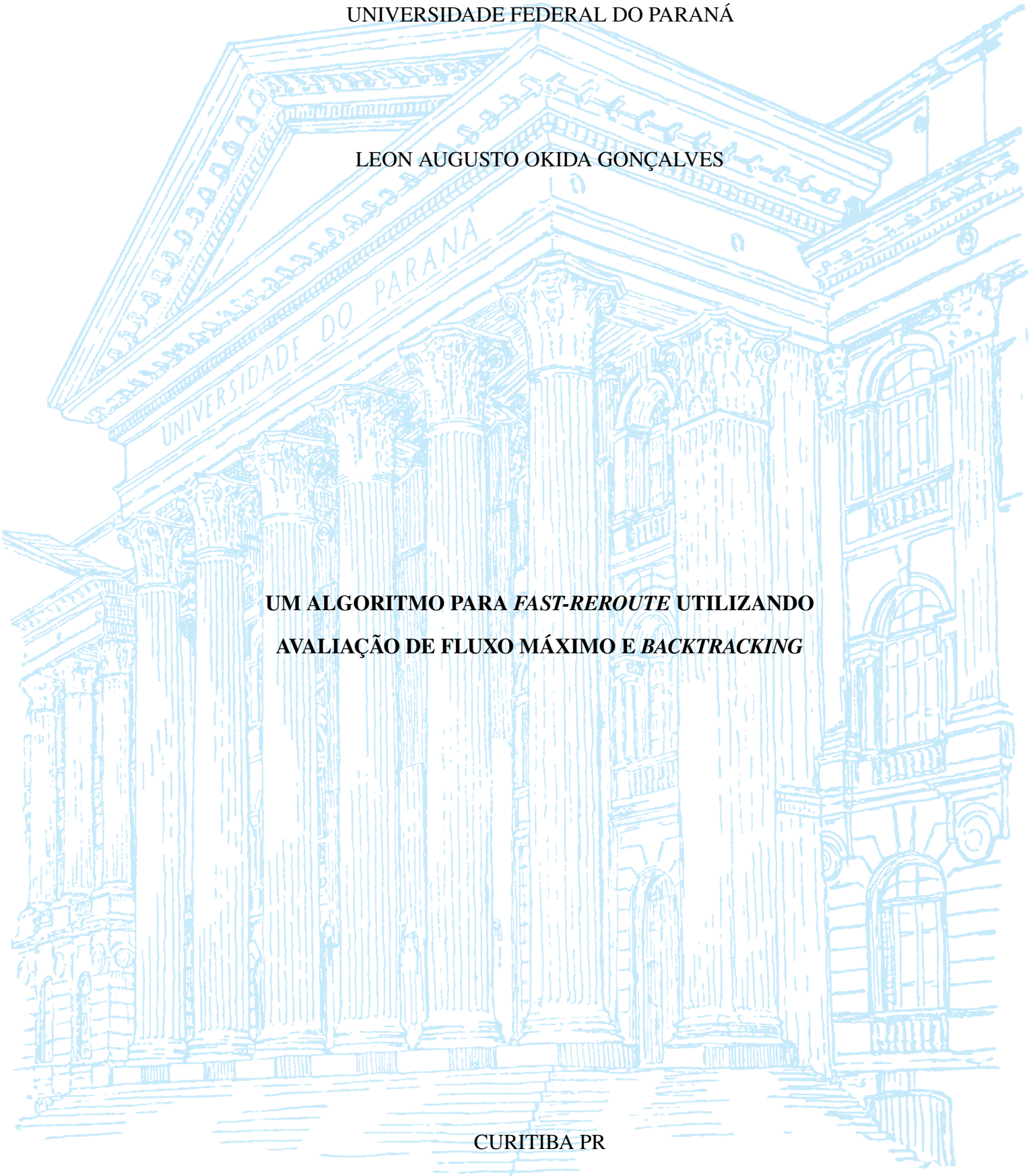
UNIVERSIDADE FEDERAL DO PARANÁ

LEON AUGUSTO OKIDA GONÇALVES



UM ALGORITMO PARA *FAST-REROUTE* UTILIZANDO AVALIAÇÃO DE FLUXO MÁXIMO E *BACKTRACKING*

CURITIBA PR

2024

LEON AUGUSTO OKIDA GONÇALVES

UM ALGORITMO PARA *FAST-REROUTE* UTILIZANDO AVALIAÇÃO DE FLUXO MÁXIMO E *BACKTRACKING*

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Computação*.

Orientador: Elias Procópio Duarte Jr.

CURITIBA PR

2024

AGRADECIMENTOS

Agradeço ao professor Elias pelo convite para fazer este trabalho. Foi uma experiência gratificante. Este trabalho não seria possível sem a sua orientação.

Agradeço também a meus pais, minhas irmãs, meus avós, meus tios, meus primos, a demais familiares e a todos os meus amigos pelo apoio e suporte que me deram durante todos esses anos. Sem esse apoio, este trabalho também não seria possível.

Por fim, gostaria de agradecer a todas as pessoas com quem convivi durante esses anos de curso. Recordo com afeto as experiências que vivi no PET Computação e no C3SL, onde, além de obter bastante aprendizado, fiz amizades muito especiais que também tiveram um papel na realização deste trabalho.

RESUMO

Este trabalho apresenta o algoritmo *MaxFlowRouting* para o roteamento tolerante a falhas. O algoritmo utiliza a estratégia de *Fast-ReRoute* (FRR) para lidar com falhas na rede. Um roteador mantém uma rota primária e rotas alternativas para cada destino. Quando um roteador que utiliza FRR detecta uma falha que impeça o uso da rota primária, ele imediatamente emprega uma rota alternativa para contornar o ponto em que ocorre a falha. O algoritmo *MaxFlowRouting* utiliza a avaliação de fluxo máximo para computar rotas que possuem mais opções de rotas alternativas que possam ser ativadas em caso de falha. Este trabalho também introduz uma estratégia para FRR com *backtracking*. Se um roteador que executa esse algoritmo não for capaz de encaminhar um pacote com sucesso tanto pela rota primária quanto pelas rotas alternativas, ele pode retorná-lo ao roteador anterior, permitindo que este busque outra rota. Resultados experimentais obtidos com simulação confirmam que o algoritmo *MaxFlowRouting* produz rotas com mais opções de rotas alternativas que o algoritmo de Dijkstra. No melhor caso, o algoritmo proposto produz rotas com até 1,54 vezes mais rotas alternativas por vértice e com soma de graus dos vértices até 1,43 vezes maior. Nesse caso, as rotas produzidas pelo algoritmo proposto têm tamanho médio apenas 1,06 vezes maior do que as rotas produzidas pelo algoritmo de Dijkstra. O roteamento computado pelo *MaxFlowRouting* também faz, no melhor caso, até 62,5% menos *backtracks* quando ocorrem falhas na rede do que o roteamento com o algoritmo de Dijkstra.

Palavras-chave: Roteamento. Tolerância a Falhas. Fast-ReRoute (FRR). Internet. Avaliação de Fluxo Máximo.

LISTA DE FIGURAS

1.1 Exemplo de escolha do próximo *hop* baseada em avaliação do fluxo máximo. . . 11

2.1 Um exemplo de grafo.. 13
2.2 Configuração inicial do algoritmo de Dijkstra: a origem s é inicializada com rótulo de valor 0 e os demais vértices possuem rótulo com valor infinito. 14
2.3 Primeira iteração do algoritmo de Dijkstra: a origem s é escolhida e seus vizinhos a e e têm seus rótulos atualizados pela soma do rótulo de s com o peso da aresta que os conecta. O vértice escolhido está em vermelho e os seus vizinhos estão em amarelo. 14
2.4 Segunda iteração do algoritmo de Dijkstra: o vértice a é escolhido e seus vizinhos b e c têm seus rótulos atualizados pela soma do rótulo de a com o peso da aresta que os conecta. O vértice escolhido está em vermelho e os seus vizinhos estão em amarelo. O vértice em azul (s) já foi marcado como visitado pelo algoritmo anteriormente.. 15
2.5 Terceira iteração do algoritmo de Dijkstra: o vértice e é escolhido e seu vizinho f tem seu rótulo atualizado pela soma do rótulo de e com o peso da aresta que os conecta. O vértice escolhido está em vermelho e o seu vizinho está em amarelo. Os vértices em azul (s e a) já foram visitados pelo algoritmo anteriormente. . . . 15
2.6 Última iteração do algoritmo de Dijkstra: o vértice t é escolhido. Como ele é o vértice de destino, encerra-se a execução do algoritmo, computando que o tamanho do caminho mínimo entre s e t é 3. Os vértices em azul já foram visitados pelo algoritmo anteriormente. 15

3.1 Configuração inicial do algoritmo de Ford-Fulkerson: todas as arestas recebem fluxo igual a 0. 18
3.2 Primeira iteração do algoritmo de Ford-Fulkerson: escolhido o caminho (s, a, t). As arestas do caminho (em vermelho) têm seu fluxo aumentado de acordo com o valor de gargalo, que é 1. 18
3.3 Segunda iteração do algoritmo de Ford-Fulkerson: escolhido o caminho (s, c, t). As arestas do caminho (em vermelho) têm seu fluxo aumentado de acordo com o valor de gargalo, que é 1. 19
3.4 O caminho (s, b, a, t) não é aumentante, pois a aresta (a, t) (em roxo) já está em sua capacidade máxima. O fluxo das arestas do caminho não aumenta. 19
3.5 Configuração inicial do algoritmo Push-Relabel: o vértice de origem s recebe rótulo igual ao número de vértices (5). Os demais vértices recebem rótulo igual a 0. As arestas adjacentes a s recebem fluxo igual a suas capacidades máximas. As demais arestas recebem fluxo igual a 0. O rótulo de cada vértice é denotado por r e o fluxo em excesso é denotado por ex. 20

3.6 Primeira iteração do algoritmo Push-Relabel: o vértice a é escolhido, aumenta seu rótulo para 1 e empurra seu fluxo para t. O rótulo de cada vértice é denotado por r e o fluxo em excesso é denotado por ex. 20
3.7 Segunda iteração do algoritmo Push-Relabel: o vértice b é escolhido, aumenta seu rótulo para 2 e empurra seu fluxo para a. O rótulo de cada vértice é denotado por r e o fluxo em excesso é denotado por ex. 20
3.8 Terceira iteração do algoritmo Push-Relabel: o vértice c é escolhido, aumenta seu rótulo para 1 e empurra seu fluxo para t. O rótulo de cada vértice é denotado por r e o fluxo em excesso é denotado por ex. 21
3.9 Última iteração do algoritmo Push-Relabel: o vértice a é escolhido e tenta empurrar seu fluxo para t, o que não é possível, pois (a,t) já está com sua capacidade máxima. Assim, a empurra o fluxo de volta para s. O rótulo de cada vértice é denotado por r e o fluxo em excesso é denotado por ex. 21

4.1 Exemplo de roteamento com *Fast-ReRoute*. 23
4.2 Segunda parte do exemplo de roteamento com *Fast-ReRoute*. 23
4.3 Terceira parte do exemplo de roteamento com *Fast-ReRoute*. 24

5.1 Casos de estudo do algoritmo *MaxFlowRouting*.. 28

6.1 Comparação da quantidade média de rotas alternativas por vértice, em grafos com conectividade $C = 0,1$ (esquerda) e $C = 0,7$ (direita). Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1. 30
6.2 Comparação da média da soma dos graus dos vértices das rotas produzidas, em grafos com conectividade $C = 0,1$ (esquerda) e $C = 0,7$ (direita). Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1. 32
6.3 Comparação do tamanho médio das rotas produzidas, em grafos com conectividade $C = 0,1$ (esquerda) e $C = 0,7$ (direita). Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1. 32
6.4 Comparação da quantidade média de rotas alternativas por vértice. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1. 33
6.5 Comparação da média da soma dos graus dos vértices das rotas produzidas. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1. 34
6.6 Comparação do tamanho médio das rotas produzidas. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1. 34
6.7 Comparação da quantidade média de rotas alternativas por vértice. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1. 36

6.8 Comparação da média da soma dos graus dos vértices das rotas produzidas. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1. 36

6.9 Comparação do tamanho médio das rotas produzidas. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1. 37

6.10 Comparação da quantidade média de rotas alternativas por vértice. Da esquerda para a direita: Internet2, Géant, RNP, Wide. Roxo: Dijkstra. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1. 37

6.11 Comparação da média da soma dos graus dos vértices das rotas produzidas. Da esquerda para a direita: Internet2, Géant, RNP, Wide. Roxo: Dijkstra. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1. 39

6.12 Comparação do tamanho médio das rotas produzidas. Da esquerda para a direita: Internet2, Géant, RNP, Wide. Roxo: Dijkstra. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1. 39

6.13 Comparação da ocorrência média de *backtracks* nos algoritmos de roteamento. Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1. 41

6.14 Comparação do tamanho médio das rotas produzidas nos cenários sem a ocorrência de falha (esquerda) e com a ocorrência de uma falha (direita). Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1. 41

6.15 Comparação da ocorrência média de *backtracks* nos algoritmos de roteamento. Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1. 43

6.16 Comparação do tamanho médio das rotas produzidas nos cenários sem a ocorrência de falha (esquerda) e com a ocorrência de uma falha (direita). Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1. 43

6.17 Comparação da ocorrência média de *backtracks* nos algoritmos de roteamento. Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1. 45

6.18 Comparação do tamanho médio das rotas produzidas nos cenários sem a ocorrência de falha (esquerda) e com a ocorrência de uma falha (direita). Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1. 45

6.19 Comparação da ocorrência média de *backtracks* nos algoritmos de roteamento. Da esquerda para a direita: Internet2, Géant, RNP e Wide. Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1. 46

6.20 Comparação do tamanho médio das rotas produzidas nos cenários sem a ocorrência de falha (esquerda) e com a ocorrência de uma falha (direita). Da esquerda para a direita: Internet2, Géant, RNP e Wide. Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.. 46

LISTA DE TABELAS

6.1 Resultados do Primeiro Experimento com Grafos Aleatórios de Conectividade Variável. 31
6.2 Resultados do Primeiro Experimento com Grafos de Conexão Preferencial. . . . 33
6.3 Resultados do Primeiro Experimento com Grafos de Pequeno Mundo. 35
6.4 Resultados do Primeiro Experimento com Grafos Representando Topologias Reais da Internet. 38
6.5 Resultados do Segundo Experimento com Grafos Aleatórios de Conectividade Variável. 40
6.6 Resultados do Segundo Experimento com Grafos de Conexão Preferencial. . . . 42
6.7 Resultados do Segundo Experimento com Grafos de Pequeno Mundo. 44
6.8 Resultados do Segundo Experimento com Grafos Representando Topologias Reais da Internet. 44

LISTA DE ACRÔNIMOS

FRR	*Fast-ReRoute*
IP	*Internet Protocol*
MPLS	*Multiprotocol Label Switching*
OSPF	*Open Shortest Path First*
BGP	*Border Gateway Protocol*
AS	Sistema Autônomo
FM	Critério de Fluxo Máximo
CM	Critério de Caminho Mínimo

SUMÁRIO

1 INTRODUÇÃO 11
2 O ALGORITMO DE DIJKSTRA 13
2.1 CAMINHOS MÍNIMOS 13
2.2 O FUNCIONAMENTO DO ALGORITMO DE DIJKSTRA 14
3 O PROBLEMA DO FLUXO MÁXIMO 17
3.1 O ALGORITMO DE FORD-FULKERSON 17
3.2 O ALGORITMO *PUSH-RELABEL* 19
4 O ROTEAMENTO TOLERANTE A FALHAS NA INTERNET COM *FAST-REROUTE* 22
5 O ALGORITMO *MAXFLOWROUTING* 25
5.1 ESPECIFICAÇÃO DO ALGORITMO *MAXFLOWROUTING* 25
5.2 ESTUDOS DE CASO 26
5.3 PROVA DE CORRETUDE DO ALGORITMO DE FRR COM *BACKTRACKING* 28
6 RESULTADOS EXPERIMENTAIS 29
6.1 COMPARAÇÃO DAS ROTAS PRODUZIDAS PELOS ALGORITMOS 30
6.1.1 Grafos Aleatórios de Conectividade Variável. 30
6.1.2 Grafos de Conexão Preferencial. 33
6.1.3 Grafos de Pequeno Mundo 35
6.1.4 Topologias Reais da Internet 36
6.2 COMPARAÇÃO DO COMPORTAMENTO DOS ALGORITMOS COM OCORRÊNCIA DE FALHAS 38
6.2.1 Grafos Aleatórios de Conectividade Variável. 39
6.2.2 Grafos de Conexão Preferencial. 41
6.2.3 Grafos de Pequeno Mundo 42
6.2.4 Topologias Reais da Internet 43
7 CONCLUSÃO 47
REFERÊNCIAS 48

1 INTRODUÇÃO

Organizações e indivíduos estão cada vez mais dependentes das redes de computadores, particularmente da Internet. Em algumas instâncias, a Internet pode apresentar bastante instabilidade (Duarte Jr et al., 2010). Para aumentar a robustez da rede nesses casos, deve-se adotar estratégias de roteamento com tolerância a falhas (Cohen et al., 2011a; Cohen e Duarte Jr, 2001b,a). Isso contribui para que a rede se recupere rapidamente da ocorrência de uma falha em sua infraestrutura. O *Fast-ReRoute* (FRR) (Shand e Bryant, 2010) é uma estratégia proativa de reparo em caso de falhas, em que uma rota alternativa é empregada quando a rota primária falha. O FRR foi adotado em diversos protocolos da Internet, como o IP (Shand e Bryant, 2010), o MPLS (Pan et al., 2005; Duarte Jr et al., 2012) e o OSPF (Filsfils et al., 2012). A estratégia alternativa ao FRR para reparação de falhas é reativa e consiste em esperar as tabelas de roteamento reconvergirem para a nova topologia da rede. Isso causa um período de disrupção no tráfego da rede, em que pacotes já enviados ao destino através da rede anteriormente à falha serão descartados, causando, consequentemente, prejuízos para serviços distribuídos e aplicações.

O FRR computa rotas alternativas que são armazenadas na tabela de roteamento junto da rota principal. Assim, quando um roteador detecta uma falha, a rota alternativa pode ser empregada imediatamente. O período de disrupção é reduzido para o tempo que o roteador leva para detectar a falha e ativar a rota alternativa. Entretanto, a efetividade do FRR depende da existência de rotas alternativas que possam contornar o ponto em que ocorreu a falha.

Neste trabalho, é proposto o algoritmo *MaxFlowRouting*, baseado em avaliação do fluxo máximo (Harris e Ross, 1955; Ford e Fulkerson, 1956; Cormen et al., 2022; Schroeder et al., 2004; Schroeder e Duarte Jr, 2007; Okida et al., 2024), para a seleção de rotas robustas. A avaliação de fluxo máximo encontra, implicitamente, rotas com maior conectividade possível considerando a topologia da rede. Isso acontece porque o fluxo máximo é igual à quantidade de caminhos disjuntos em arestas. Consequentemente, é maior o número de rotas alternativas que podem ser utilizadas na ocorrência de uma falha.

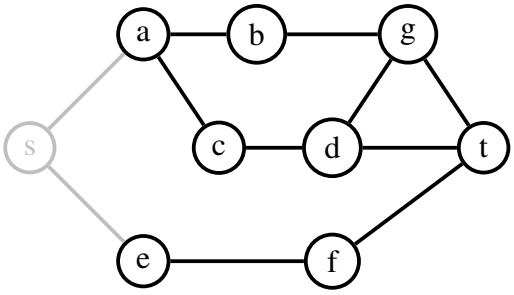


Figura 1.1: Exemplo de escolha do próximo *hop* baseada em avaliação do fluxo máximo.

O algoritmo *MaxFlowRouting* é executado por um roteador e constrói uma tabela de roteamento com múltiplas alternativas para o próximo *hop* para cada destino. A Figura 1.1 ilustra como um roteador executando o algoritmo *MaxFlowRouting* escolhe o próximo *hop*. Na figura, a origem s precisa enviar uma mensagem ao destino t. O algoritmo está sendo executado pelo vértice s. Existem duas arestas adjacentes a ele: (s,a) e (s,e). Isso significa que o próximo *hop* será para a ou para e. Para fazer essa escolha, o algoritmo inicialmente gera um subgrafo removendo o vértice que está executando o algoritmo. Consequentemente, as arestas adjacentes a esse vértice são removidas também. Na figura, o subgrafo é formado pelas linhas e vértices escuros.

Para avaliar a opção pela aresta (s, a), o algoritmo computa os critérios de caminho mínimo e de fluxo máximo entre a e t no subgrafo. Ele obtém, nesse caso, o caminho mínimo de tamanho 3 e o fluxo máximo de valor 2. Fazendo a mesma avaliação para a opção pela aresta (s, e), obtém-se o caminho mínimo de tamanho 2 e o fluxo máximo de valor 1. Isso significa que existem dois caminhos disjuntos entre a e t, enquanto apenas um entre e e t. O algoritmo utiliza pesos para definir a influência de cada critério na decisão do próximo *hop*. Como o foco do algoritmo é tolerância a falhas, o critério de fluxo máximo deve ter um peso maior. No exemplo, o vértice a tem preferência maior como o próximo *hop* por ter um valor de fluxo máximo do que o vértice e.

No trabalho, também é descrita uma estratégia de FRR baseada em *backtracking* que pode ser utilizada em qualquer tabela de roteamento que possua múltiplas alternativas de próximo *hop* por destino. Se a rota alternativa também falha, o pacote pode retornar para o roteador anterior para que ele tente outra rota. Assim, o roteamento tem maiores chances de funcionar, mesmo que as tabelas de roteamento não reflitam a topologia real da rede. Basta que nenhum roteador visitado anteriormente falhe e que exista pelo menos uma rota funcional entre a origem e o destino. Essa estratégia de FRR pode utilizar rotas produzidas tanto pelo algoritmo *MaxFlowRouting* como por outros algortimos de roteamento, como o algoritmo de Dijkstra (Dijkstra, 1959).

Foram feitos experimentos com simulações para comparar o algoritmo *MaxFlowRouting* com o algoritmo de Dijkstra. Os dois algoritmos foram executados em diversas topologias, como grafos aleatórios (grafos de Erdos-Renyi (Erdos e Rényi, 1960), de Barábasi-Albert (Albert e Barabási, 2002) e de Watts-Strogatz (Watts e Strogatz, 1998)) e também em representações de *backbones* importantes da Internet, como a RNP do Brasil (RNP, 2024), a Internet2 dos Estados Unidos da América (Internet2, 2024), a Géant da Europa (GÉANT, 2024) e a Wide do Japão (WIDE Project, 2024). Resultados confirmam que o algoritmo proposto produz rotas com mais opções de alternativas do que o algoritmo de Dijkstra. No melhor caso, o *MaxFlowRouting* computou rotas com até 1,54 vezes mais rotas alternativas por vértice e com grau de conectividade 1,43 vezes maior. Nesse caso, as rotas produzidas pelo algoritmo proposto foram apenas 1,06 vezes maiores do que as rotas produzidas pelo algoritmo de Dijkstra, que produz rotas com o menor tamanho possível. O roteamento computado pelo *MaxFlowRouting* também faz, no melhor caso, até 62,5% menos *backtracks* na ocorrência de falhas na rede do que o roteamento com o algoritmo de Dijkstra.

O restante deste trabalho está organizado como segue. O Capítulo 2 descreve brevemente o algoritmo de Dijkstra, que é aplicado no protocolo OSPF (*Open Shortest Path First*), amplamente utilizado na Internet. O Capítulo 3 apresenta o problema do fluxo máximo e algoritmos utilizados para resolvê-lo. O Capítulo 4 aborda o roteamento na Internet e a estratégia de FRR para reparação de falhas. O Capítulo 5 apresenta o algoritmo *MaxFlowRouting* e o algoritmo de FRR propostos neste trabalho. O Capítulo 6 apresenta os resultados experimentais da comparação do *MaxFlowRouting* com o algoritmo de Dijkstra obtidos com simulação. Por fim, o Capítulo 7 conclui o trabalho.

2 O ALGORITMO DE DIJKSTRA

Este capítulo apresenta de maneira resumida o algoritmo de Dijkstra (Dijkstra, 1959), que resolve o problema do caminho mínimo entre dois vértices de um grafo. Na Internet, o algoritmo de Dijkstra é utilizado como algoritmo de roteamento do protocolo OSPF (Moy, 1998). Mais à frente nesta monografia, o algoritmo de Dijkstra será comparado com a estratégia de roteamento proposta. Na seção 2.1, é definido o que é um caminho mínimo. Na seção 2.2, são apresentados o funcionamento do algoritmo de Dijkstra e um exemplo de execução.

2.1 CAMINHOS MÍNIMOS

Esta seção apresenta definições básicas, incluindo a de caminho mínimo. Para isso, são necessárias definições anteriores, como a de passeio e a de caminho em um grafo.

Considere um *grafo* $G = (V, E)$, onde V é o conjunto de vértices e E é o conjunto de arestas. Um *passeio* no grafo G é uma sequência de vértices $P = (v_1, ..., v_2)$, onde cada par de vértices consecutivos a e b em P é conectado por uma aresta (a, b) em G. Um *caminho* é um passeio em que não há repetição de arestas. Um caminho que não possui repetição de vértices é chamado de simples. Como exemplo, no grafo da Figura 2.1, o passeio (a, e, d, h, t, b, a, e) não é um caminho, pois há a repetição da aresta (a, e). O passeio (c, d, h, t, d, f) é também um caminho, pois não há repetição de arestas. Ele não é um caminho simples pois há repetição do vértice d. Já o passeio (a, e, d, c, b, t) é um caminho simples, pois não há repetição de vértices ou arestas.

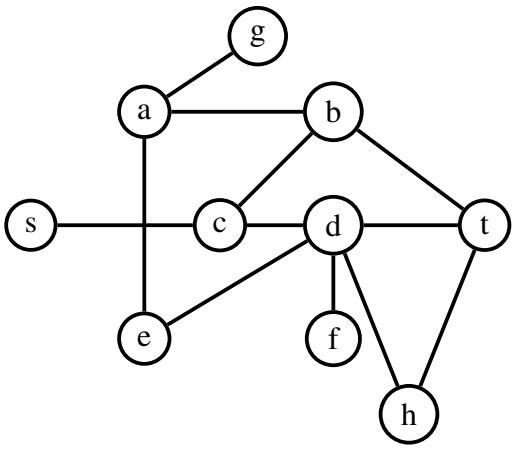


Figura 2.1: Um exemplo de grafo.

No problema do caminho mínimo, assume-se que o grafo G possui pesos em suas arestas. Existe uma função $w((x, y))$ de pesos que associa um valor para cada aresta (x, y) de G. Um caminho mínimo entre dois vértices u e v de G é o caminho C, tal que, para todos os outros caminhos C' em G entre u e v, temos que $\sum_{(u,v)\in C} w(u, v) \leq \sum_{(u,v)\in C'} w(u, v)$. É o caminho entre os vértices u e v que possui a menor soma de pesos possível.

Neste trabalho, considera-se que o peso de todas as arestas é igual a 1. Essa escolha é para que a distância entre dois vértices seja calculada pelo número de vértices percorridos entre eles. Isso é análogo ao número de *hops* em uma rota entre dois roteadores no contexto da Internet. Um *hop* é a passagem de uma mensagem de um roteador para o próximo na rota.

2.2 O FUNCIONAMENTO DO ALGORITMO DE DIJKSTRA

O algoritmo de Dijkstra permite determinar o menor caminho entre um vértice origem $u \in V$ e um vértice destino $v \in V$ no grafo G. O algoritmo constrói, iterativamente, os menores caminhos entre o vértice u e os demais vértices do grafo, até atingir o vértice v.

No algoritmo, inicialmente cada vértice recebe como rótulo um valor de distância inicial, sendo 0 para o vértice de origem u e infinita para os demais. Esse valor de distância é referente à distância do vértice com relação à origem. A cada iteração do algoritmo, é escolhido o vértice k de menor distância com relação à origem que não tenha sido visitado ainda. Utiliza-se uma estrutura de fila de prioridades para armazenar os vértices, escolhendo o de menor rótulo.

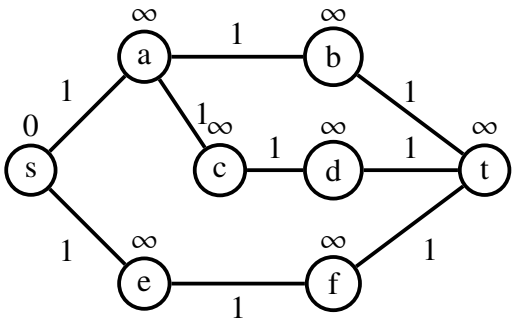


Figura 2.2: Configuração inicial do algoritmo de Dijkstra: a origem s é inicializada com rótulo de valor 0 e os demais vértices possuem rótulo com valor infinito.

Na primeira iteração, é escolhida a própria origem u. Para cada vizinho i de k, calcula-se a distância dele para o vértice de origem, somando o rótulo de distância de k com o peso da aresta (k, i). Se o valor da nova distância calculada for menor que o valor de distância atual de i, ele é atualizado. O vértice escolhido k é então marcado como já visitado, para não ser escolhido novamente em outras iterações. Após isso, as iterações se seguem, escolhendo o vértice de menor distância com relação à origem que não tenha sido visitado ainda, até que o vértice de destino v seja visitado. Assim, se obtém o tamanho do menor caminho entre os vértices u e v. Para encontrar o menor caminho em si, basta armazenar, para cada vértice t, o vértice p que atualizou o valor da menor distância de t em sua iteração. Esse vértice p será o predecessor de t no caminho mais curto.

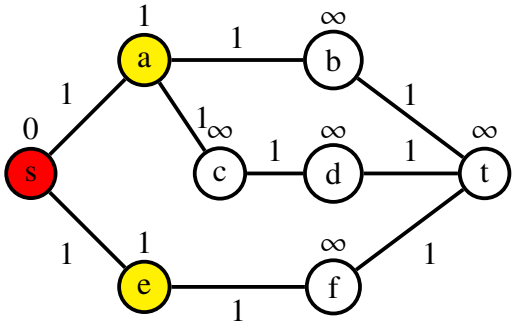


Figura 2.3: Primeira iteração do algoritmo de Dijkstra: a origem s é escolhida e seus vizinhos a e e têm seus rótulos atualizados pela soma do rótulo de s com o peso da aresta que os conecta. O vértice escolhido está em vermelho e os seus vizinhos estão em amarelo.

Por exemplo, o algoritmo de Dijkstra pode ser utilizado para encontrar o menor caminho entre o vértice s e o vértice t no grafo da Figura 2.2. Inicialmente, s recebe o rótulo de distância

de valor 0 e os demais vértices recebem o rótulo de valor infinito. É escolhido o vértice com o menor rótulo, sendo escolhido o s. Nessa etapa, os vizinhos de s, a e e, são atualizados com suas respectivas distâncias para a origem, como mostrado na Figura 2.3. Ambos recebem o valor 1, pois é o peso de cada aresta do grafo. Após isso, s é marcado como já visitado e escolhe-se um dos vértices com menor rótulo de distância, ou seja, a ou e.

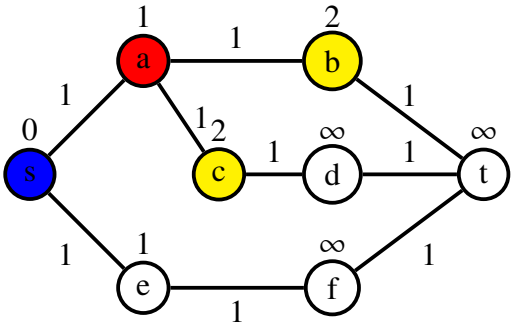


Figura 2.4: Segunda iteração do algoritmo de Dijkstra: o vértice a é escolhido e seus vizinhos b e c têm seus rótulos atualizados pela soma do rótulo de a com o peso da aresta que os conecta. O vértice escolhido está em vermelho e os seus vizinhos estão em amarelo. O vértice em azul (s) já foi marcado como visitado pelo algoritmo anteriormente.

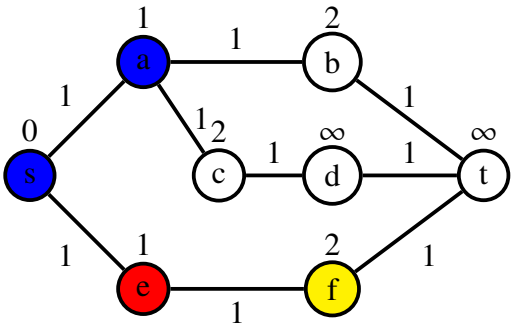


Figura 2.5: Terceira iteração do algoritmo de Dijkstra: o vértice e é escolhido e seu vizinho f tem seu rótulo atualizado pela soma do rótulo de e com o peso da aresta que os conecta. O vértice escolhido está em vermelho e o seu vizinho está em amarelo. Os vértices em azul (s e a) já foram visitados pelo algoritmo anteriormente.

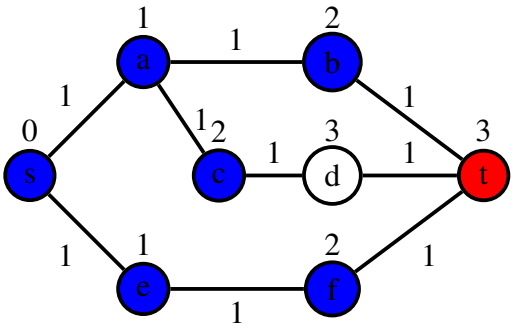


Figura 2.6: Última iteração do algoritmo de Dijkstra: o vértice t é escolhido. Como ele é o vértice de destino, encerra-se a execução do algoritmo, computando que o tamanho do caminho mínimo entre s e t é 3. Os vértices em azul já foram visitados pelo algoritmo anteriormente.

Escolhendo arbitrariamente o vértice a, os vizinhos b e c são atualizados com distância igual a 2, calculada pela soma da distância de a até a origem e a distância de a até b e c, como mostrado na Figura 2.4.

O vértice a é marcado como visitado e escolhe-se o vértice e, como mostrado na Figura 2.5. Os passos descritos se repetem até que o vértice t seja visitado, como mostrado na Figura 2.6. Assim, encerra-se a execução do algoritmo, obtendo o tamanho do menor caminho entre s e t, que é 3.

O algoritmo de Dijkstra possui complexidade de tempo de $O(|V|^2)$ (Cormen et al., 2022) no pior caso e é o algoritmo padrão utilizado pelo protocolo da Internet OSPF (Moy, 1998). O OSPF é um protocolo de roteamento interno que é utilizado para a construção das tabelas de roteamento de um sistema autônomo.

3 O PROBLEMA DO FLUXO MÁXIMO

Este capítulo descreve o problema da avaliação do fluxo máximo (Harris e Ross, 1955; Ford e Fulkerson, 1956; Cormen et al., 2022; Schroeder et al., 2004), bem como algoritmos para resolvê-lo. A ideia de fluxo em um grafo representa um sistema em que um recurso, produzido por uma fonte, percorre uma rede até um destino. A quantidade desse recurso produzida pela fonte é a mesma que é consumida pelo destino. Entretanto, a rede tem uma capacidade limitada de conduzir esse fluxo, que não pode ser ultrapassada. Isso limita a quantidade do recurso que pode ser gerada pela fonte. O problema da avaliação do fluxo máximo consiste em determinar o maior fluxo possível entre um vértice origem e um vértice destino em um grafo. A avaliação de fluxo máximo é utilizada no algoritmo de roteamento proposto neste trabalho.

Um sistema de condução de fluxo pode ser representado por um grafo $G = (V, E)$, em que V é o conjunto de vértices e E é o conjunto de arestas. Dois vértices u e $v \in V$ representam, respectivamente, a origem e o destino do fluxo. Cada aresta do grafo possui uma capacidade máxima de conduzir fluxo que não pode ser excedida. O fluxo que entra em um vértice não pode exceder o fluxo que sai dele, com exceção dos vértices de origem e de destino. O problema do fluxo máximo consiste em determinar a maior quantidade de fluxo que a origem pode produzir e que pode percorrer o grafo até o destino, respeitando a capacidade das arestas.

Neste trabalho, considera-se que a capacidade de fluxo de todas as arestas é 1. Assim, a avaliação de fluxo máximo encontra, implicitamente, rotas com mais conectividade e que possuem mais opções de caminhos alternativos em caso de falha. Isso ocorre nesse caso porque, o fluxo máximo entre dois vértices indica o número de caminhos disjuntos em arestas entre eles. Consequentemente, obtém-se um número maior de caminhos alternativos entre os vértices.

O fluxo máximo é equivalente ao problema do corte mínimo (Cohen et al., 2011b, 2012, 2017; Maske et al., 2020). O corte mínimo m entre dois vértices u e v de um grafo G é o conjunto de arestas C, tal que a remoção de todas as arestas de C em G remove todos os caminhos possíveis entre u e v, desconectando-os. A cardinalidade do corte C, denotada por $|C|$ é o número de arestas em C. Um corte C é chamado de corte mínimo se, para cada corte C' entre o mesmo par de vértices u e v no grafo G, $|C| \leq |C'|$. Para cada par de vértices no grafo G, o fluxo máximo e o corte mínimo possuem a mesma cardinalidade e podem ser computados pelos mesmos algoritmo.

O algoritmo de Ford-Fulkerson (Ford e Fulkerson, 1956) foi o primeiro algoritmo proposto para determinar o fluxo máximo. A Seção 3.1 contém uma descrição desse algoritmo e um exemplo de execução. Na Seção 3.2, é abordado o algoritmo *Push-Relabel* (Goldberg e Tarjan, 1988), que é mais recente que o algoritmo de Ford-Fulkerson e é utilizado na proposta deste trabalho. Além disso, é apresentado um exemplo de aplicação do algoritmo.

3.1 O ALGORITMO DE FORD-FULKERSON

O algoritmo de Ford-Fulkerson (Ford e Fulkerson, 1956) resolve o problema do fluxo máximo aumentando, gradativamente, o fluxo no grafo até que o limite máximo seja atingido. O fluxo é aumentado ao encontrar, iterativamente, caminhos da origem ao destino em que todas as arestas ainda possuem capacidade livre para conduzir fluxo. Esses caminhos são chamados de aumentantes. Quando um caminho aumentante é encontrado, determina-se qual é a aresta no caminho com a menor capacidade disponível. A capacidade dessa aresta é o gargalo do caminho

aumentante. Então, todas as arestas do caminho aumentante têm o seu fluxo aumentado de acordo com o gargalo encontrado.

No primeiro passo do algoritmo, todas as arestas são inicializadas com fluxo igual a 0. Iterativamente, são selecionados caminhos aumentantes entre a origem e o destino. O algoritmo não especifica como encontrar os caminhos aumentantes. Para cada aresta do caminho aumentante, calcula-se sua capacidade livre, subtraindo o fluxo atual de sua capacidade máxima. Como mencionado acima, o menor valor de capacidade livre calculado entre todas as arestas é o valor de gargalo do caminho aumentante. O valor de gargalo determina como o fluxo de cada aresta do caminho aumentante é aumentado. Isso faz com que o fluxo no grafo aumente sem exceder a capacidade de nenhuma aresta. As iterações se repetem até que não existam mais caminhos aumentantes no grafo entre a origem e o destino. O fluxo resultante ao final da execução do algoritmo é igual ao fluxo máximo.

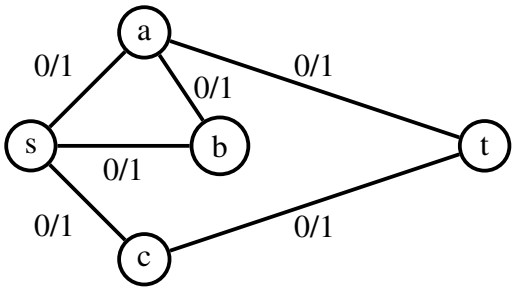


Figura 3.1: Configuração inicial do algoritmo de Ford-Fulkerson: todas as arestas recebem fluxo igual a 0.

Como exemplo de aplicação, o algoritmo de Ford-Fulkerson pode ser utilizado para determinar o fluxo máximo entre o vértice de origem s e o vértice de destino t no grafo da Figura 3.1. Inicialmente, todas as arestas recebem fluxo igual a 0, sendo que o valor máximo possível em todas as arestas é igual a 1.

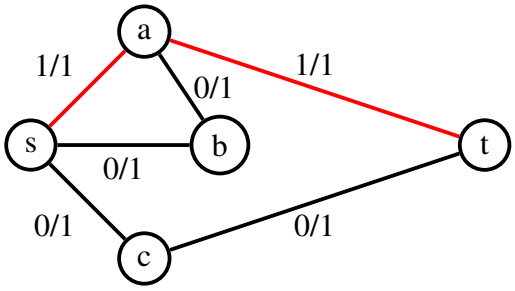


Figura 3.2: Primeira iteração do algoritmo de Ford-Fulkerson: escolhido o caminho (s, a, t). As arestas do caminho (em vermelho) têm seu fluxo aumentado de acordo com o valor de gargalo, que é 1.

Feita a configuração inicial do algoritmo, é escolhido arbitrariamente o caminho aumentante (s, a, t). As arestas do caminho têm seu fluxo aumentado para o valor de gargalo, que é 1, como mostrado na Figura 3.2.

Na próxima iteração, é escolhido arbitrariamente o caminho aumentante (s, c, t). As arestas desse caminho aumentante também têm seu fluxo aumentado para o valor de gargalo, que é 1, como ilustrado na Figura 3.3.

Após isso, verifica-se se há mais caminhos aumentantes que possam ser escolhidos. O caminho ainda não explorado (s, b, a, t) possui a aresta (a, t), que já está com o fluxo igual a sua capacidade máxima. Portanto, o caminho (s, b, a, t) não é aumentante, como ilustrado na Figura 3.4. Assim, o algoritmo encerra a execução, computando que o fluxo máximo entre s e t é igual a 2.

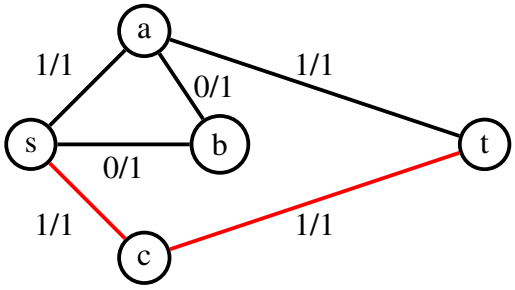


Figura 3.3: Segunda iteração do algoritmo de Ford-Fulkerson: escolhido o caminho (s, c, t). As arestas do caminho (em vermelho) têm seu fluxo aumentado de acordo com o valor de gargalo, que é 1.

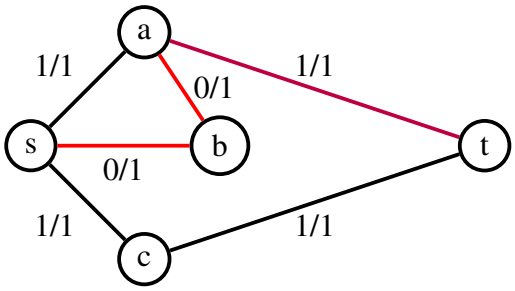


Figura 3.4: O caminho (s, b, a, t) não é aumentante, pois a aresta (a, t) (em roxo) já está em sua capacidade máxima. O fluxo das arestas do caminho não aumenta.

Como o algoritmo de Ford-Fulkerson não define a estratégia de escolha dos caminhos aumentantes, a complexidade de tempo do algoritmo depende da implementação.

3.2 O ALGORITMO *PUSH-RELABEL*

O algoritmo *Push-Relabel* (Goldberg e Tarjan, 1988) resolve o problema do fluxo máximo utilizando o conceito de *preflow*. O *preflow* permite, temporariamente, que um vértice tenha fluxo em excesso, com um fluxo de entrada maior que o fluxo de saída. Iterativamente, os vértices empurram o fluxo em excesso que possuem, por meio da operação *push*, para seus vizinhos. Para controlar esse processo, os vértices possuem rótulos. Um vértice só pode empurrar seu fluxo em excesso para um vizinho que possui um rótulo de valor menor. Caso um vértice não consiga empurrar o fluxo para nenhum vizinho e ainda possua fluxo em excesso, ele pode aumentar o seu rótulo por meio da operação *relabel*.

No algoritmo, inicialmente o vértice de origem é rotulado com um valor igual ao número de vértices do grafo. Os demais vértices recebem rótulo igual a 0. Na configuração inicial, o fluxo nas arestas adjacentes ao vértice de origem é aumentado de acordo com a capacidade máxima dessas arestas, resultando em fluxo em excesso nos vértices vizinhos da origem. Após isso, o algoritmo iterativamente seleciona um vértice com fluxo em excesso para realizar a operação de *push*. Essa operação transfere o fluxo excedente do vértice para um vizinho com rótulo menor, respeitando a capacidade máxima da aresta. Caso o vértice não possua vizinhos com rótulos menores para a operação de *push*, o algoritmo faz uso da operação *relabel*. Ela aumenta o rótulo do vértice para 1 a mais que o menor rótulo de seus vizinhos, possibilitando que ele faça uso da operação *push*. Quando um vértice não consegue empurrar seu fluxo em direção ao destino pela falta de capacidade disponível nas arestas adjacentes, é necessário que ele diminua o fluxo que recebe. Para isso, ele empurra o fluxo em excesso de volta, em direção à origem. O algoritmo realiza essas operações até que não haja mais fluxo em excesso em nenhum dos vértices, com exceção da origem e do destino. Quando isso acontece, obtém-se o fluxo máximo no grafo.

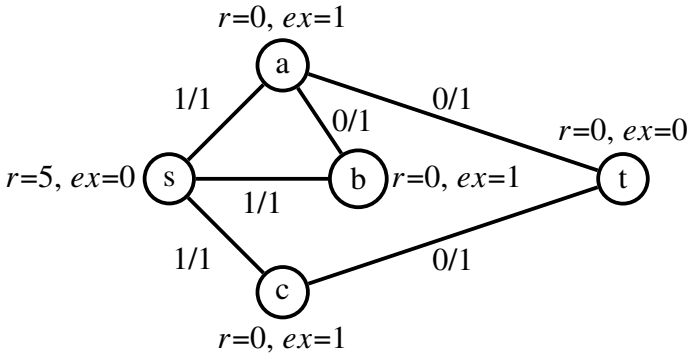


Figura 3.5: Configuração inicial do algoritmo Push-Relabel: o vértice de origem s recebe rótulo igual ao número de vértices (5). Os demais vértices recebem rótulo igual a 0. As arestas adjacentes a s recebem fluxo igual a suas capacidades máximas. As demais arestas recebem fluxo igual a 0. O rótulo de cada vértice é denotado por r e o fluxo em excesso é denotado por ex.

Como exemplo de execução, o algoritmo *Push-Relabel* pode ser utilizado para determinar o fluxo máximo entre os vértices s e t no grafo da Figura 3.5. Inicialmente, o vértice s recebe o rótulo $r = 5$ (o número de vértices do grafo) e os demais recebem rótulo $r = 0$. As arestas adjacentes a s, que são (s, a), (s, b) e (s, c), recebem fluxo igual a sua capacidade máxima. Assim, a, b e c estão com fluxo em excesso $ex = 1$, como ilustrado na Figura 3.5.

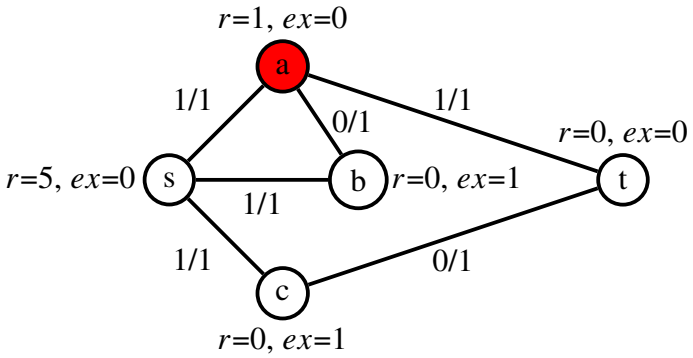


Figura 3.6: Primeira iteração do algoritmo Push-Relabel: o vértice a é escolhido, aumenta seu rótulo para 1 e empurra seu fluxo para t. O rótulo de cada vértice é denotado por r e o fluxo em excesso é denotado por ex.

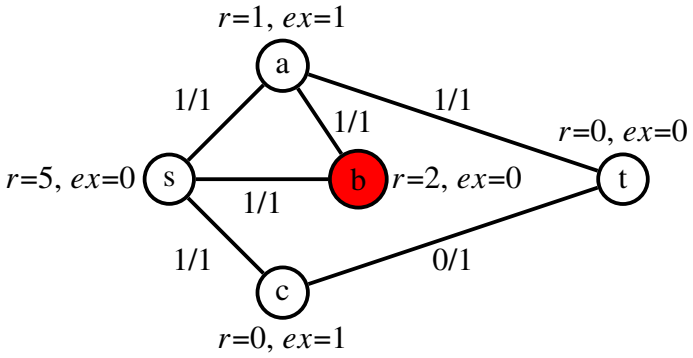


Figura 3.7: Segunda iteração do algoritmo Push-Relabel: o vértice b é escolhido, aumenta seu rótulo para 2 e empurra seu fluxo para a. O rótulo de cada vértice é denotado por r e o fluxo em excesso é denotado por ex.

Iniciam-se, então, as iterações do algoritmo. O vértice a é selecionado, pois possui fluxo em excesso, que é igual a 1. Ele tenta realizar a operação de *push* para o seu vizinho t.

Entretanto, isso não é possível, pois a não tem um rótulo maior que t. Assim, ele realiza a operação *relabel*, aumentando seu rótulo para 1 (1 a mais que o menor rótulo de seus vizinhos). Desse modo, ele consegue empurrar seu fluxo em excesso para t, como na Figura 3.6.

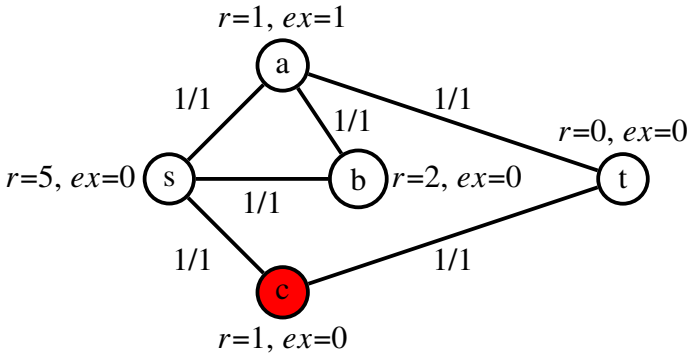


Figura 3.8: Terceira iteração do algoritmo Push-Relabel: o vértice c é escolhido, aumenta seu rótulo para 1 e empurra seu fluxo para t. O rótulo de cada vértice é denotado por r e o fluxo em excesso é denotado por ex.

Após isso, é escolhido o vértice b, que também possui fluxo em excesso igual a 1. Ele tenta empurrar seu fluxo em excesso para o vizinho a. Como ele possui rótulo menor que a, ele também deve fazer uso da operação *relabel* para isso, ficando com rótulo igual a 2. Como mostrado na Figura 3.7, quando b realiza a operação de *push*, a volta a ter fluxo em excesso.

Após essa iteração, c é selecionado. Ao fazer uso da operação *relabel*, ele consegue empurrar seu fluxo em excesso para t, como mostrado na Figura 3.8.

O algoritmo ainda não é encerrado, pois a ainda possui fluxo em excesso. Ele tenta empurrar seu fluxo em excesso para t, mas isso não é possível, pois a aresta (a,t) já está em sua capacidade máxima. Assim, ele empurra o fluxo em excesso de volta para s, reduzindo seu fluxo de entrada, como na Figura 3.9. Como não há mais vértices com fluxo em excesso, o algoritmo se encerra. Conclui-se que o fluxo máximo entre s e t é 2.

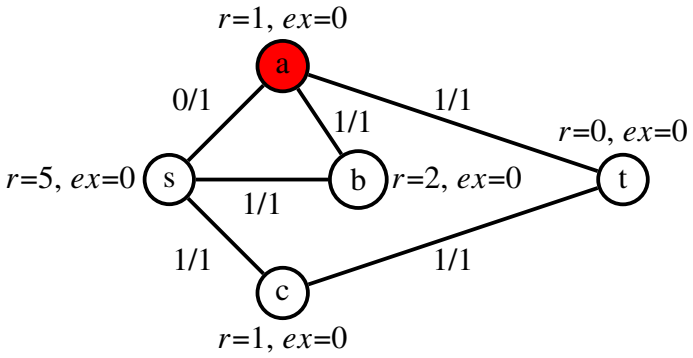


Figura 3.9: Última iteração do algoritmo Push-Relabel: o vértice a é escolhido e tenta empurrar seu fluxo para t, o que não é possível, pois (a,t) já está com sua capacidade máxima. Assim, a empurra o fluxo de volta para s. O rótulo de cada vértice é denotado por r e o fluxo em excesso é denotado por ex.

O algoritmo *Push-Relabel* possui complexidade de tempo de $O(|V|^2|E|)$ (Goldberg e Tarjan, 1988).

4 O ROTEAMENTO TOLERANTE A FALHAS NA INTERNET COM *FAST-REROUTE*

O roteamento na Internet permite que uma máquina qualquer de uma rede envie um pacote que chega até qualquer destino final (Medhi, 2010). Para isso, o pacote passa por intermediários, chamados de *roteadores*. Ao receber um pacote, os roteadores verificam o endereço do destino final e, após uma busca local, encaminham o pacote para o próximo roteador, para que ele avance até chegar ao destino final. Desta forma, o roteamento é o processo que define para qual roteador um pacote é encaminhado, com base em seu destino.

Para que o roteamento na Internet seja viável, é necessário que as máquinas possam ser localizadas. Para isso, são atribuídos endereços IP (*Internet Protocol*) a elas. Um endereço IP é composto por duas partes: o *netid* e o *hostid*. O *hostid* identifica o *host*, que é a máquina ou dispositivo que se comunica na Internet. Um *host* é o ponto inicial e também o ponto final de uma comunicação na Internet. O *netid* identifica a rede em que o *host* está conectado. A Internet é uma rede virtual que interliga várias redes, cada uma contendo uma coleção de *hosts*.

Cada roteador decide, independentemente, o próximo roteador para qual ele deve encaminhar o pacote. O próximo roteador é também chamado de próximo *hop* e deve ser ligado ao roteador atual por meio de um enlace. A decisão de qual roteador deve ser o próximo *hop* é tomada pelo roteador atual com base na *tabela de roteamento*. Na tabela de roteamento, há uma entrada para cada *netid* conhecido, que é associada a um roteador específico que deve ser selecionado para o próximo *hop*.

Na Internet, o protocolo IP (Postel, 1981; Deering e Hinden, 2017) é responsável pelo encaminhamento de pacotes. O preenchimento das tabelas de roteamento de cada roteador é responsabilidade dos protocolos de roteamento. Dois dos principais protocolos de roteamento da Internet são o BGP (*Border Gateway Protocol*) (Rekhter et al., 2006) e o OSPF (*Open Shortest Path First*) (Moy, 1998). O BGP é responsável por definir rotas entre redes mantidas por organizações diferentes. Uma rede mantida por uma organização é chamada de Sistema Autônomo (AS). Roteadores que enviam mensagens entre ASes diferentes são chamados de Roteadores de Borda ou Roteadores Externos. O protocolo OSPF define rotas entre roteadores internos de um mesmo Sistema Autônomo. Esses roteadores são chamados de Roteadores Internos.

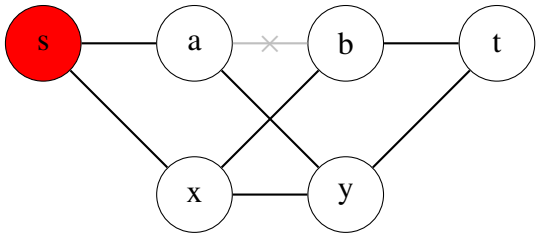
O protocolo BGP utiliza o algoritmo de Bellman-Ford (Bellman, 1958) para construir as tabelas de roteamento dos Roteadores de Borda. Assim como o algoritmo de Dijkstra, descrito no Capítulo 2, o algoritmo de Bellman-Ford constrói caminhos de tamanho mínimo entre vértices, mas sua grande vantagem é que a topologia da rede não precisa ser conhecida por seus roteadores. Em rodadas, os roteadores trocam informações sobre suas tabelas de roteamento com seus vizinhos, de modo que, iterativamente, elas são expandidas e atualizadas. Entretanto, o algoritmo apresenta longa latência de convergência (Duarte Jr et al., 2004). Ele é utilizado no BGP porque é impossível que um roteador conheça a topologia completa da Internet.

O protocolo OSPF utiliza o algoritmo de Dijkstra, apresentado no Capítulo 2, para a construção das tabelas de roteamento dos Roteadores Internos. O algoritmo de Dijkstra é mais rápido que o algoritmo de Bellman-Ford. Para que ele funcione, é necessário que a topologia seja conhecida pelos roteadores. Isso é possível internamente em um Sistema Autônomo. Além disso, existem algoritmos para descoberta e manutenção da topologia de redes dinâmicas (Nassu et al., 2007).

A topologia da rede de um Sistema Autônomo pode ser alterada, porém, pela ocorrência de falhas na rede. Falhas podem ocorrer em roteadores ou enlaces, tornando-os indisponíveis

temporariamente (Shand e Bryant, 2010). Isso faz com que o roteamento não funcione nos casos em que a rota computada pelo algoritmo de roteamento atravessa uma parte da rede que apresenta falhas. Após uma falha ser detectada, o próprio algoritmo de roteamento ou algoritmos de diagnóstico disseminam informações sobre o estado de roteadores e enlaces (Duarte Jr e Weber, 2003; Duarte Jr et al., 2023). Em seguida, os roteadores aplicam alguma estratégia para lidar com a falha. A estratégia comum para reparar esse problema é reativa, consistindo em executar o algoritmo de roteamento novamente com a nova topologia, reconvergindo as tabelas de roteamento dos roteadores da rede. Enquanto esse processo de reconvergência das tabelas de roteamento não termina, existe um período de disrupção do tráfego na rede que pode ser demorado. Além disso, pacotes que chegaram aos seus destinos anteriormente à ocorrência da falha podem ser descartados. Essa situação causa prejuízos para serviços distribuídos e demais aplicações que utilizam a rede.

Uma estratégia alternativa e mais veloz para lidar com a ocorrência de falhas na rede é o *Fast-ReRoute* (FRR). Diversos protocolos adotam o *Fast-ReRoute*, como o IP (Shand e Bryant, 2010), o MPLS (Pan et al., 2005) e o OSPF (Filsfils et al., 2012). O FRR reduz o tempo de disrupção do tráfego na rede ao fazer com que cada roteador repare o problema localmente. Cada roteador pré-computa rotas alternativas além da rota principal determinada pelo algoritmo de Dijkstra. Essas rotas devem contornar o próximo *hop* da rota principal. Ao detectar a ocorrência de uma falha no próximo *hop*, o roteador ativa a rota alternativa, conseguindo repassar o pacote sem a necessidade de informar os demais roteadores da falha. Assim, o tempo de disrupção é reduzido para o tempo do roteador detectar a falha e ativar a rota alternativa.



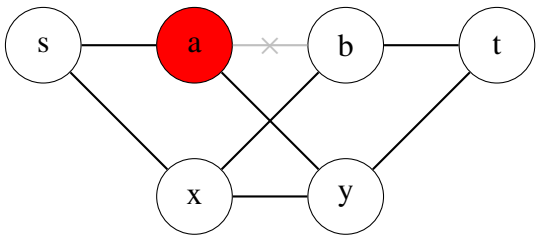
(a) O roteador s (em vermelho) roteia um pacote para t.

Destino	Próximo Hop	Alternativa
a	a	x
b	a	x
x	x	a
y	x	a
t	a	x

(b) A tabela de roteamento do roteador s com rotas alternativas computadas para FRR.

Figura 4.1: Exemplo de roteamento com *Fast-ReRoute*.

Como exemplo de aplicação do *Fast-ReRoute*, considere a situação mostrada na Figura 4.1. Um pacote deve ser roteado pelo roteador s com destino ao roteador t. Para isso, s consulta sua tabela de roteamento, mostrada na Figura 4.1(b), e verifica que, para o destino t, o próximo *hop* deve ser o roteador a. Portanto, ele encaminha o pacote para a.



(a) O roteador a (em vermelho) roteia um pacote para t. Ocorre uma falha no enlace entre os roteadores a e b.

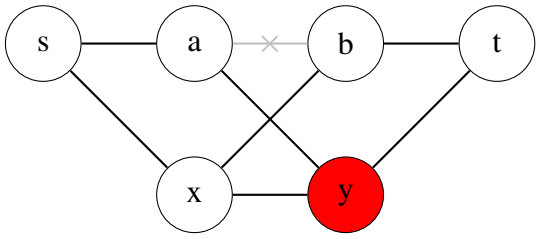
Destino	Próximo Hop	Alternativa
s	s	y
b	b	y
x	s	y
y	y	b
t	b	y

(b) A tabela de roteamento do roteador a com rotas alternativas computadas para FRR.

Figura 4.2: Segunda parte do exemplo de roteamento com *Fast-ReRoute*.

Como mostrado na Figura 4.2, o roteador a recebe o pacote encaminhado por s. Então, a também consulta sua tabela de roteamento para decidir o próximo *hop* da rota para o destino t.

Ao consultar sua tabela de roteamento, mostrada na Figura 4.2(b), ele identifica que o próximo *hop* da rota principal é o roteador *b*. Entretanto, *a* detecta uma falha no enlace que o conecta a *b*. Diante disso, a estratégia de FRR é empregada, e *a* verifica em sua tabela de roteamento que existe uma rota alternativa para *t* através do roteador *y*. Assim, ele encaminha o pacote para *y*.



(a) O roteador *y* (em vermelho) roteia um pacote para *t*.

Destino	Próximo Hop	Alternativa
a	a	x
b	a	t
x	x	a
s	x	a
t	t	a

(b) A tabela de roteamento do roteador *y* com rotas alternativas computadas para FRR

Figura 4.3: Terceira parte do exemplo de roteamento com *Fast-ReRoute*.

Como ilustrado na Figura 4.3, o roteador *y* recebe o pacote encaminhado por *a*. Ao consultar sua tabela de roteamento, como mostra a Figura 4.3(b), *y* identifica que o próximo *hop* deve ser o próprio roteador *t*. Então, *y* envia o pacote para *t*, chegando ao destino final da rota.

5 O ALGORITMO *MAXFLOWROUTING*

Este capítulo apresenta o algoritmo *MaxFlowRouting* para roteamento tolerante a falhas. Ele faz uso de um algoritmo de FRR com *backtracking*, definido a seguir. Após a descrição dos algoritmos de roteamento e de FRR propostos, são apresentados estudos de caso e a prova de que o algoritmo de FRR consegue rotear com sucesso um pacote para o destino se nenhum roteador visitado anteriormente falha e se existe pelo menos uma rota funcional na topologia.

5.1 ESPECIFICAÇÃO DO ALGORITMO *MAXFLOWROUTING*

Um roteador que executa o algoritmo *MaxFlowRouting* mantém o grafo $G = (V, E)$ como uma representação local da topologia da rede. Não é necessário que o grafo G reflita exatamente a topologia real da rede. Quando um roteador detecta uma mudança na topologia, ele deve disseminar essa informação para os demais roteadores. A tabela de roteamento e o grafo G são atualizados pelo roteador quando este recebe uma informação de mudança na topologia. Cada roteador executa o Algoritmo 1 (*MaxFlowRouting*) para gerar a tabela de roteamento para todos os destinos possíveis na rede. O algoritmo *MaxFlowRouting* utiliza a avaliação de fluxo máximo e o algoritmo de Dijkstra para decidir qual deve ser o próximo *hop* da rota.

Um roteador seleciona o próximo *hop* ordenando todos os seus vizinhos com base em uma função avaliadora $\Gamma(G', i, d)$. A função é executada por cada roteador s para cada vizinho i considerando um destino d. Essa função utiliza o grafo $G' = (V', E')$ onde $V' = V - s$ e $E' = E - (s, j), \forall j | (s, j) \in E$. A função $\Gamma(G', i, d)$ retorna um valor numérico para cada vértice i adjacente a s considerando o destino d, de forma que valores altos são considerados melhores que valores baixos, como descrito a seguir.

A função $\Gamma(G', i, d)$ é computada levando em conta o fluxo máximo e o tamanho da rota, sendo calculada pela expressão $\Gamma(G', i, d) = w_1 c_1(G', i, d) + w_2 c_2(G', i, d)$. Essa expressão leva em conta os critérios de valor do fluxo máximo entre i e d no grafo G' (critério c_1) e de tamanho da rota de tamanho mínimo entre i e d no grafo G' (critério c_2). Cada critério é multiplicado por um peso (respectivamente w_1 e w_2), que determina como o critério influencia o resultado da função $\Gamma(G', i, d)$. Então, dado um vértice s, todos os seus vizinhos i que possuem caminho para o destino d são ordenados de modo decrescente de acordo com o resultado da função $\Gamma(G', i, d)$ como candidatos a próximo *hop*. Os vértices i de maior valor de $\Gamma(G', i, d)$ são escolhidos primeiro para rotear um pacote de s para o destino d. Assim, o peso correspondente ao critério de fluxo máximo deve ser positivo, pois o objetivo é maximizá-lo, enquanto o peso correspondente ao critério de caminho mínimo deve ser negativo, pois deseja-se minimizá-lo.

O Algoritmo 1 (*MaxFlowRouting*) é executado por cada roteador s, que inicialmente se remove do grafo G da topologia conhecida, gerando o grafo G'. Após isso, para cada destino possível d e para cada vizinho i, s computa o fluxo máximo e o tamanho do menor caminho entre i e d. Feito isso, a função $\Gamma(G', i, d)$ é computada utilizando pesos para os critérios de fluxo máximo e distância. Para cada destino (entrada na tabela de roteamento), existe uma lista de *CandidatosAProximoHop*, que é inicializada vazia. Um vizinho i é incluído em *CandidatosAProximoHop* se possui um caminho para o destino d em G'. Finalmente, os vizinhos em *CandidatosAProximoHop* são ordenados de acordo com a função avaliadora $\Gamma(G', i, d)$.

Algoritmo 1 Especificação do algoritmo *MaxFlowRouting*: Geração da tabela de roteamento do roteador s

```
1: G' ← G − {s}
2: for all destino d em G' do
3:    CandidatosAProximoHop ← lista vazia
4:    for all vizinho i adjacente a s em G do
5:       if não há caminho entre i a d em G' then
6:          ignora i
7:       else
8:          adiciona i a CandidatosAProximoHop
9:          computa Γ(G', i, d)
10:      end if
11:   end for
12:   if CandidatosAProximoHop possui mais de um elemento then
13:      ordena CandidatosAProximoHop de acordo com Γ(G', i, d)
14:   end if
15:   TabelaDeRoteamento[d] ← CandidatosAProximoHop
16: end for
```

A tabela de roteamento gerada pelo algoritmo *MaxFlowRouting* é utilizada como base para o roteamento de pacotes. O algoritmo 2 de FRR utiliza a tabela de roteamento para selecionar o próximo *hop* dado o destino do pacote.

O Algoritmo 2 (*Fast-ReRoute* com *Backtracking*) é executado pelo roteador s para encaminhar um pacote enviado pela origem o para o destino d. Primeiro, o roteador s checa se ele é vizinho do roteador d e, caso seja, envia o pacote diretamente a d. Cada pacote pct carrega consigo a lista sequencial dos vértices pelos quais passou, chamada de $pct.NodosVisitados$. O roteador s adiciona a si mesmo a $pct.NodosVisitados$ caso não esteja na lista. Caso o roteador já esteja na lista, um ciclo é detectado, o que ocorre quando o pacote é enviado por *backtracking* por um vizinho que não tinha rota disponível para o destino. Após isso, o roteador s seleciona o próximo *hop* para encaminhar o pacote. O vizinho de melhor classificação na entrada da tabela de roteamento para o destino d que não esteja em $pct.NodosVisitados$ é selecionado para ser o próximo *hop*. Caso não exista um vizinho disponível para isso, o roteador s retorna o pacote para o roteador r de onde recebeu o pacote, isto é, faz *backtracking*. Caso não exista r antes de s, então s é própria a origem o do pacote e não existe rota disponível entre o e d.

5.2 ESTUDOS DE CASO

A Figura 5.1 apresenta dois exemplos de aplicação do algoritmo *MaxFlowRouting* e do algoritmo proposto de *Fast-ReRoute* com *Backtracking*. A parte (a) da figura apresenta a situação em que o enlace (b, t) falha durante o roteamento de um pacote do nodo s para o nodo t. Inicialmente, s envia o pacote para a, que é a única opção possível para o próximo *hop*. Então, a recebe o pacote e consulta sua tabela de roteamento para decidir para qual roteador deve encaminhá-lo. Na tabela de roteamento de a, o nodo b é indicado como a opção preferencial de próximo *hop*, pois possui o maior valor atribuído pela função avaliadora Γ. Assim, a encaminha o pacote para b. Então, o nodo b tenta enviar o pacote diretamente para t, mas não consegue devido à falha no enlace (b, t). Como não possui mais opções de próximo *hop*, b retorna o pacote para a. Após receber o pacote novamente, a verifica então o nodo com o maior valor retornado pela função avaliadora Γ que ainda não foi visitado. Assim, a encaminha o pacote

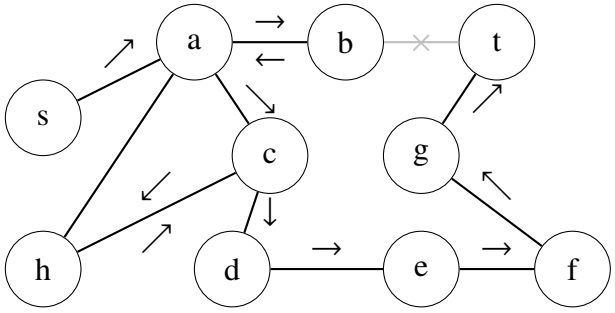
Algoritmo 2 Especificação do algoritmo *Fast-ReRoute* com *Backtracking* no roteador s.

```
1:  if destino d é adjacente a s em G then
2:      encaminha pacote pct para d
3:  else if pct.NodosVisitados não contém o roteador s then
4:      adiciona s a pct.NodosVisitados
5:      if TabelaDeRoteamento[d] contém roteadores que não estão em pct.NodosVisitados
        then
6:          escolhe o próximo hop i de acordo com TabelaDeRoteamento[d], tal que i ∉
            pct.NodosVisitados
7:          envia pacote pct para i
8:      else
9:          // Não é possível encaminhar o pacote, deve fazer backtracking
10:         if pct.NodosVisitados contém pelo menos um roteador r antes de s then
11:             envia pct de volta para r
12:         else
13:             // Não é possível fazer backtracking, pois está na origem o
14:             retorna Erro: Não há rota disponível entre o e d
15:         end if
16:     end if
17: end if
```

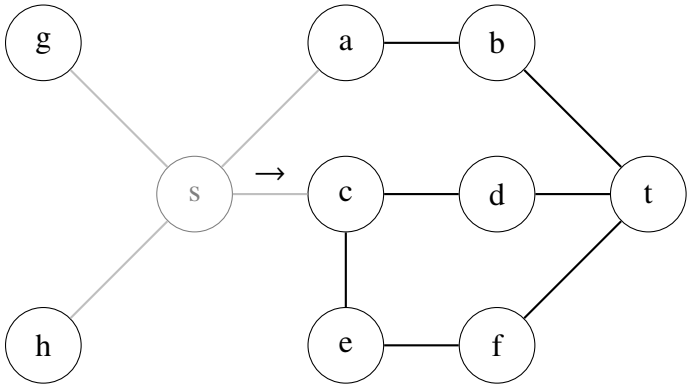
para o nodo c. A rota percorrida até este momento é (s, a, c), com o nodo b também marcado como visitado, mas sem fazer parte da rota utilizada para chegar ao destino.

A tabela de roteamento de c apresenta as opções a, h e d ordenadas pela função avaliadora Γ. A opção prioritária é a, mas esta já está marcada como visitada. Portanto, o nodo h é escolhido. O nodo h possui um valor retornado por Γ maior do que o nodo d porque possui um tamanho menor de caminho mínimo até t, desconsiderando a falha no enlace (b, t), que é desconhecida por c. O pacote é então encaminhado para h. Entretanto, as opções de roteamento a partir de h já estão marcadas como visitadas. Por isso, o pacote é retornado para c, que escolhe então o nodo d. Em seguida, o pacote é encaminhado para d e percorre o caminho (d, e, f, g, t) para chegar até o destino final.

A Figura 5.1(b) mostra um segundo caso de estudo, que ilustra como um nodo seleciona um de seus vizinhos para o roteamento. O nodo s roteia um pacote para o nodo t. Na topologia original, representada pelo grafo G, o nodo s possui quatro opções para o próximo *hop*: os nodos a, c, g e h. Para avaliar as opções com a função Γ, é gerado o grafo G' retirando s e todas as suas arestas adjacentes de G. Sobram, assim, apenas dois vértices em G' que possuem pelo menos um caminho para t: a e c. A função Γ retorna um valor maior para o nodo c, pois este possui um valor maior para o critério de fluxo máximo e o mesmo valor para o critério de tamanho do caminho mínimo comparado ao nodo a. Então, o nodo s escolhe o nodo c para encaminhar o pacote. O pacote então percorre a rota (s, c, d, t) para chegar até o destino. Note que a rota que passa por c possui mais opções de rotas alternativas para serem empregadas em caso de falha do que a única rota possível que passa por a.



(a) O enlace (b, t) falha quando b tenta enviar o pacote para t. O algoritmo de FRR com *backtracking* é utilizado para rotear, com sucesso, o pacote para t.



(b) O nodo s computa os critérios de fluxo máximo e caminho mínimo para escolher o próximo *hop*. O nodo escolhido para o próximo *hop* é c.

Figura 5.1: Casos de estudo do algoritmo *MaxFlowRouting*.

5.3 PROVA DE CORRETUDE DO ALGORITMO DE FRR COM *BACKTRACKING*

Esta seção apresenta a prova de que o algoritmo de FRR com *backtracking* consegue rotear um pacote do nodo s para o nodo t pela rede representada pelo grafo G se nenhum nodo visitado anteriormente falha e se existe pelo menos uma rota funcional da origem até o destino.

Teorema 5.1. Considere o grafo $G = (V, E)$ representando a topologia da rede e dois vértices selecionados arbitrariamente $s, t \in V$, representando a origem e o destino de um pacote p. O algoritmo de FRR com *backtracking* consegue rotear o pacote com sucesso entre s e t se nenhum vértice já visitado falha e se existe pelo menos uma rota funcional entre s e t em G.

Prova. O nodo executando o algoritmo de FRR com *backtracking* tenta enviar o pacote através do nodo vizinho de melhor avaliação na entrada da tabela de roteamento correspondente ao destino t. Este nodo, por sua vez, também faz isso. Esse processo se repete até que o pacote chegue ao destino. Caso um nodo j receba o pacote e detecte que não possui uma rota funcional para o destino, ele pode retornar o pacote ao nodo i anterior de onde recebeu o pacote. O nodo i tenta enviar o pacote através do próximo nodo k de melhor avaliação na entrada da tabela de roteamento correspondente ao destino. O processo se repete: se não há rota funcional entre k e t, o pacote será novamente retornado para i. Após i tentar todas as suas opções de próximo *hop* sem sucesso, ele retorna para o nodo anterior de onde recebeu o pacote. Se nenhum nodo já visitado anteriormente falhar, esse processo pode acontecer, sucessivamente, até o pacote ser retornado para a origem s, que também tentará todas as suas alternativas de próximo *hop*. Assim, se existe uma rota funcional entre a origem e o destino na rede, ela será encontrada após um tempo e utilizada para rotear o pacote com sucesso para o destino t. □

6 RESULTADOS EXPERIMENTAIS

Este capítulo apresenta os resultados experimentais obtidos com simulação comparando o algoritmo *MaxFlowRouting* e o algoritmo de Dijkstra. Foram realizados dois experimentos. O primeiro deles compara as rotas produzidas pelo *MaxFlowRouting* com as rotas produzidas pelo algoritmo de Dijkstra. O segundo experimento compara o comportamento dos algoritmos com a ocorrência de uma falha. Os experimentos foram realizados com programas desenvolvidos com a linguagem *Python* e a biblioteca *NetworkX* (Hagberg et al., 2008). Nos dois experimentos, o algoritmo *Push-Relabel* é utilizado para a avaliação do fluxo máximo. O algoritmo *MaxFlowRouting* é executado nos experimentos considerando 3 pares diferentes de pesos para os critérios de Fluxo Máximo (FM) e Caminho Mínimo (CM). Os pares escolhidos foram: FM = 2 e CM = -5, que faz com que o algoritmo produza rotas de tamanhos menores; FM = 5 e CM = -5, que equilibra a influência dos dois critérios; e FM = 5 e CM = -1, que favorece a produção de rotas com maior conectividade.

Nos dois experimentos, os algoritmos de roteamento foram executados em diversas topologias. Foram utilizados 3 tipos de grafos aleatórios. O primeiro deles é o Grafo Aleatório de Conectividade Variável de Erdos-Renyi (Erdos e Rényi, 1960). Nesses grafos de tamanho N, o número de arestas adjacentes a um vértice é determinada por um parâmetro de conectividade $0 < C \leq 1$. O parâmetro C reflete a possibilidade de dois vértices serem conectados por uma aresta. Portanto, quanto maior o valor de C, maior a conectividade do grafo. Quando $C = 1$, a conectividade é de 100%, isto é, o grafo é completo. De forma análoga, quando $C = 0,5$, a probabilidade de existir uma aresta entre quaisquer dois vértices é de 50%. O mesmo padrão se aplica para os demais valores possíveis de C.

Os Grafos de Conexão Preferencial de Barabási-Albert (Albert e Barabási, 2002) são grafos de tamanho N gerados, iterativamente, ao adicionar cada vértice com um número fixo m de arestas. Cada aresta conecta o novo vértice a vértices já existentes, com preferência por aqueles com graus maiores, ou seja, os que possuem um número maior de vizinhos.

Por fim, os Grafos de Pequeno Mundo de Watts-Strogatz (Watts e Strogatz, 1998) são grafos de N vértices gerados a partir de uma topologia em forma de anel, em que cada vértice é conectado a seus k vizinhos mais próximos. Então, o grafo é construído ao substituir, com probabilidade p, cada aresta (u, v) entre vértices vizinhos u e v por uma aresta (u, t), em que t é um outro vértice do grafo selecionado aleatoriamente.

Os algoritmos foram também executados em grafos representando topologias reais da Internet. Foram selecionadas topologias importantes dos EUA, da Europa, do Brasil e do Japão. A Internet2 (Internet2, 2024) é uma rede com pontos de acesso em diversas localidades dos Estados Unidos da América. A rede Géant (GÉANT, 2024) é um *backbone* que conecta diversas redes de instituições educacionais e de pesquisa pelo continente europeu. A rede Ipê da RNP (RNP, 2024) conecta diversos órgãos e instituições acadêmicas do Brasil. A rede Wide (WIDE Project, 2024) é um *backbone* que conecta diversas empresas, universidades e instituições de pesquisa no Japão.

O restante do capítulo está organizado como segue. A Seção 6.1 apresenta os resultados do primeiro experimento, que compara as rotas produzidas pelos algoritmos. A Seção 6.2 apresenta os resultados obtidos com o segundo experimento, que compara como os algoritmos se comportam com a ocorrência de uma falha.

6.1 COMPARAÇÃO DAS ROTAS PRODUZIDAS PELOS ALGORITMOS

No primeiro experimento, as rotas produzidas pelo *MaxFlowRouting* e pelo algoritmo de Dijkstra são comparadas levando em consideração 3 métricas. A primeira métrica é o tamanho médio das rotas. A segunda métrica é a soma dos graus de todos os vértices da rota. A ideia é que um vértice com um grau maior tem mais possibilidades de possuir uma rota alternativa para o destino. A terceira métrica é o número médio de rotas alternativas disponíveis por vértice de cada rota. Uma rota alternativa é uma rota disjunta da rota original produzida pelo algoritmo de roteamento. A métrica é computada como segue. Inicialmente, as arestas percorridas pela rota original são removidas do grafo. Em seguida, para cada vértice da rota, com exceção da origem e do destino, verifica-se se cada um de seus vizinhos possui um caminho para o destino. Para cada vizinho com caminho funcional para o destino, incrementa-se o contador de rotas alternativas associado ao vértice. Ao final, somam-se todos os contadores e divide-se o resultado pelo tamanho da rota, excluindo a origem e o destino. Assim, obtém-se o número médio de rotas alternativas por vértice da rota.

A Subseção 6.1.1 apresenta os resultados experimentais obtidos executando os algoritmos nos Grafos Aleatórios de Conectividade Variável. A Subseção 6.1.2 apresenta os resultados para a aplicação dos algoritmos em Grafos de Conexão Preferencial. A Subseção 6.1.3 apresenta os resultados obtidos considerando os Grafos de Pequeno Mundo. Por fim, a Subseção 6.1.4 apresenta os resultados experimentais obtidos com a execução dos algoritmos nas topologias reais da Internet apresentadas anteriormente.

6.1.1 Grafos Aleatórios de Conectividade Variável

No experimento, foram utilizados Grafos Aleatórios de Conectividade Variável de Erdos-Renyi (Erdos e Rényi, 1960), que são grafos de N vértices e com a probabilidade C de existir uma aresta ligando quaisquer dois vértices. Os algoritmos foram executados em grafos de tamanho $N = 100, 150$ e 200 e com níveis de conectividade $C = 0,1, 0,3, 0,5$ e $0,7$. Foram computadas rotas entre todos os vértices do grafo que não possuem aresta entre si. As métricas foram calculadas somente considerando casos em que as rotas produzidas por cada algoritmo diferem. Os resultados estão apresentados na Tabela 6.1.

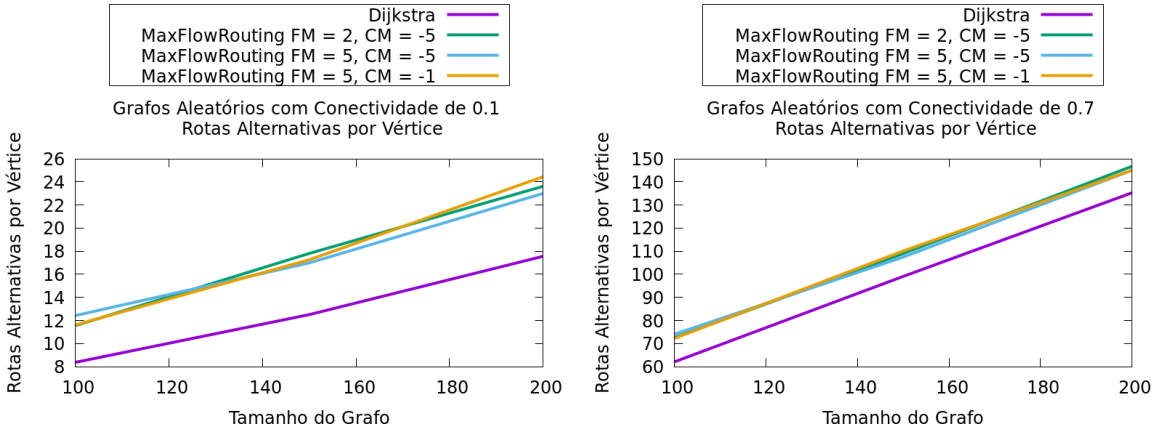


Figura 6.1: Comparação da quantidade média de rotas alternativas por vértice, em grafos com conectividade $C = 0,1$ (esquerda) e $C = 0,7$ (direita). Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.

A Figura 6.1 mostra com clareza que rotas computadas pelo *MaxFlowRouting* possuem mais opções de rotas alternativas por vértice do que as rotas produzidas pelo algoritmo de

Tabela 6.1: Resultados do Primeiro Experimento com Grafos Aleatórios de Conectividade Variável.

N	*C*	Algoritmo	Tam. Médio	Soma de Graus	Rotas Alternativas
100	0,1	Dijkstra	3,49	37,32	8,38
100	0,1	*MaxFlowRouting* com 2,-5	3,95	48,09	11,56
100	0,1	*MaxFlowRouting* com 5,-1	4,85	60,57	11,62
100	0,1	*MaxFlowRouting* com 5,-5	4,27	54,70	12,43
100	0,3	Dijkstra	3,00	90,74	26,28
100	0,3	*MaxFlowRouting* com 2,-5	3,03	98,60	33,67
100	0,3	*MaxFlowRouting* com 5,-1	4,14	106,73	34,59
100	0,3	*MaxFlowRouting* com 5,-5	3,13	102,55	33,54
100	0,5	Dijkstra	3,00	147,61	44,71
100	0,5	*MaxFlowRouting* com 2,-5	3,00	155,85	54,45
100	0,5	*MaxFlowRouting* com 5,-1	3,04	157,87	53,77
100	0,5	*MaxFlowRouting* com 5,-5	3,01	163,87	56,80
100	0,7	Dijkstra	3,00	203,59	62,13
100	0,7	*MaxFlowRouting* com 2,-5	3,00	213,45	72,62
100	0,7	*MaxFlowRouting* com 5,-1	3,00	214,21	72,28
100	0,7	*MaxFlowRouting* com 5,-5	3,01	216,81	74,08
150	0,1	Dijkstra	3,38	51,22	12,51
150	0,1	*MaxFlowRouting* com 2,-5	3,85	68,59	17,81
150	0,1	*MaxFlowRouting* com 5,-1	4,24	69,67	17,26
150	0,1	*MaxFlowRouting* com 5,-5	4,37	76,01	16,99
150	0,3	Dijkstra	3,00	135,64	40,51
150	0,3	*MaxFlowRouting* com 2,-5	3,02	148,99	51,57
150	0,3	*MaxFlowRouting* com 5,-1	3,07	149,19	50,76
150	0,3	*MaxFlowRouting* com 5,-5	3,09	150,45	50,44
150	0,5	Dijkstra	3,00	224,96	71,10
150	0,5	*MaxFlowRouting* com 2,-5	3,01	234,37	80,28
150	0,5	*MaxFlowRouting* com 5,-1	3,02	230,44	78,18
150	0,5	*MaxFlowRouting* com 5,-5	3,02	241,53	81,98
150	0,7	Dijkstra	3,00	311,04	99,10
150	0,7	*MaxFlowRouting* com 2,-5	3,00	318,70	109,01
150	0,7	*MaxFlowRouting* com 5,-1	3,01	326,50	110,13
150	0,7	*MaxFlowRouting* com 5,-5	3,00	318,62	107,48
200	0,1	Dijkstra	3,23	66,57	17,56
200	0,1	*MaxFlowRouting* com 2,-5	3,78	88,55	23,62
200	0,1	*MaxFlowRouting* com 5,-1	4,25	103,68	24,45
200	0,1	*MaxFlowRouting* com 5,-5	4,32	99,99	23,00
200	0,3	Dijkstra	3,00	177,46	53,69
200	0,3	*MaxFlowRouting* com 2,-5	3,09	196,16	65,21
200	0,3	*MaxFlowRouting* com 5,-1	3,07	194,70	66,17
200	0,3	*MaxFlowRouting* com 5,-5	3,03	194,03	67,53
200	0,5	Dijkstra	3,00	300,88	95,26
200	0,5	*MaxFlowRouting* com 2,-5	3,00	310,89	105,78
200	0,5	*MaxFlowRouting* com 5,-1	3,01	315,93	107,21
200	0,5	*MaxFlowRouting* com 5,-5	3,01	313,88	106,80
200	0,7	Dijkstra	3,00	419,51	135,30
200	0,7	*MaxFlowRouting* com 2,-5	3,00	432,45	146,78
200	0,7	*MaxFlowRouting* com 5,-1	3,01	428,44	145,06
200	0,7	*MaxFlowRouting* com 5,-5	3,00	429,13	145,04

Dijkstra. Observa-se que número de rotas alternativas por vértice cresce conforme o tamanho do grafo aumenta. No entanto, a diferença entre os resultados dos algoritmos diminui conforme a conectividade do grafo aumenta. Esse resultado é esperado. Inclusive, em um grafo completo, os resultados dos dois algoritmos seriam idênticos. É notável, também, que os diferentes pares de pesos não geraram variações significativas no funcionamento do *MaxFlowRouting*.

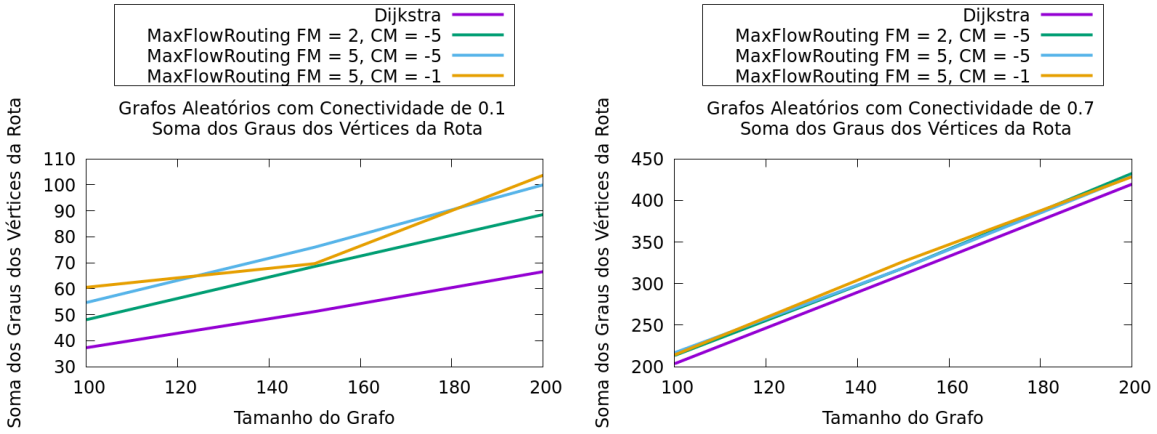


Figura 6.2: Comparação da média da soma dos graus dos vértices das rotas produzidas, em grafos com conectividade $C = 0,1$ (esquerda) e $C = 0,7$ (direita). Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.

Considerando a métrica de média da soma dos graus dos vértices da rota, a Figura 6.2 mostra que o algoritmo *MaxFlowRouting* produz rotas com maior conectividade em comparação ao algoritmo de Dijkstra. No entanto, à medida que a conectividade do grafo aumenta, a diferença entre a conectividade das rotas geradas pelos dois algoritmos diminui. A variação nos resultados das rotas produzidas pelo *MaxFlowRouting* com os diferentes pares de pesos também é reduzida com o aumento da conectividade do grafo. Além disso, observa-se que a média da soma dos graus dos vértices da rota aumenta em grafos maiores.

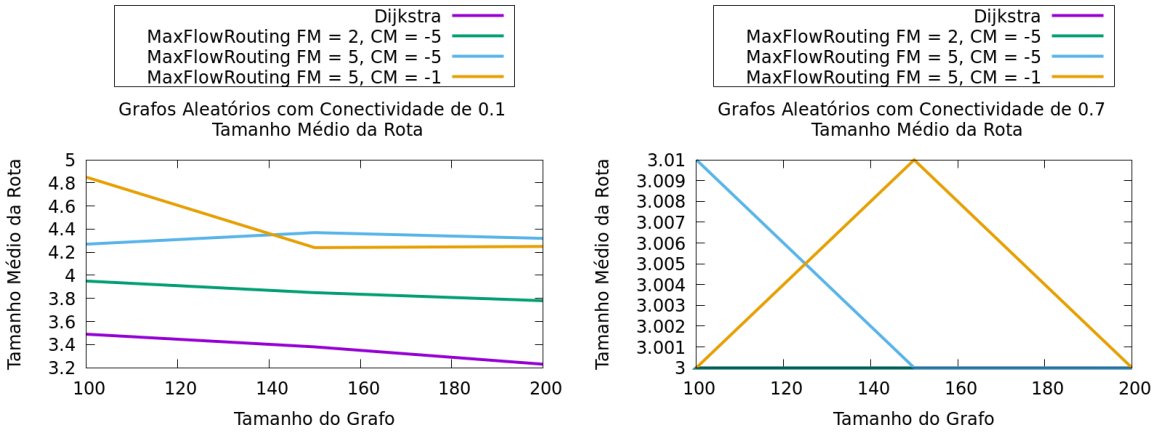


Figura 6.3: Comparação do tamanho médio das rotas produzidas, em grafos com conectividade $C = 0,1$ (esquerda) e $C = 0,7$ (direita). Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.

O tamanho das rotas produzidas pelo *MaxFlowRouting* é levemente maior do que o tamanho das rotas computadas pelo algoritmo de Dijkstra, como mostrado na Figura 6.3. A diferença nos tamanhos das rotas produzidas pelos diferentes algoritmos diminui em grafos de maior conectividade. No grafo de conectividade 0,1, o *MaxFlowRouting* com pesos FM = 2 e

Tabela 6.2: Resultados do Primeiro Experimento com Grafos de Conexão Preferencial.

N	Algoritmo	Tam. Médio	Soma de Graus	Rotas Alternativas
100	Dijkstra	4,02	39,11	11,28
100	*MaxFlowRouting* com 2,-5	4,21	44,66	13,19
100	*MaxFlowRouting* com 5,-5	4,27	49,61	15,18
100	*MaxFlowRouting* com 5,-1	4,14	53,26	17,88
150	Dijkstra	4,15	48,67	14,90
150	*MaxFlowRouting* com 2,-5	4,26	55,27	17,64
150	*MaxFlowRouting* com 5,-5	4,60	61,86	17,59
150	*MaxFlowRouting* com 5,-1	4,38	69,67	22,94
200	Dijkstra	4,24	53,06	16,16
200	*MaxFlowRouting* com 2,-5	4,39	63,53	20,26
200	*MaxFlowRouting* com 5,-5	4,60	69,65	20,74
200	*MaxFlowRouting* com 5,-1	4,97	72,33	20,05

CM = -5 foi de um tamanho médio de rota de $3,95$ em $N = 100$ para $3,85$ em $N = 150$ e $3,78$ em $N = 200$, enquanto o algoritmo de Dijkstra foi de $3,54$ em $N = 100$, para $3,39$ em $N = 150$ e para $3,25$ em $N = 200$. No grafo de conectividade 0,7, os dois algoritmos produziram rotas com tamanho médio igual a 3, para todos os tamanhos de grafo (N).

6.1.2 Grafos de Conexão Preferencial

Os Grafos de Conexão Preferencial de Barabási-Albert (Albert e Barabási, 2002) são grafos de N vértices, gerados de forma iterativa pela adição de vértices com m arestas, que se conectam preferencialmente a vértices de maior grau. No experimento, foram utilizados grafos de tamanho $N = 100, 150$ e 200 com vértices com $m = 3$ arestas. Foram computadas rotas entre todos os vértices do grafo que não são vizinhos entre si. As métricas foram calculadas somente considerando casos em que as rotas produzidas por cada algoritmo são diferentes. Os resultados obtidos são apresentados na Tabela 6.2.

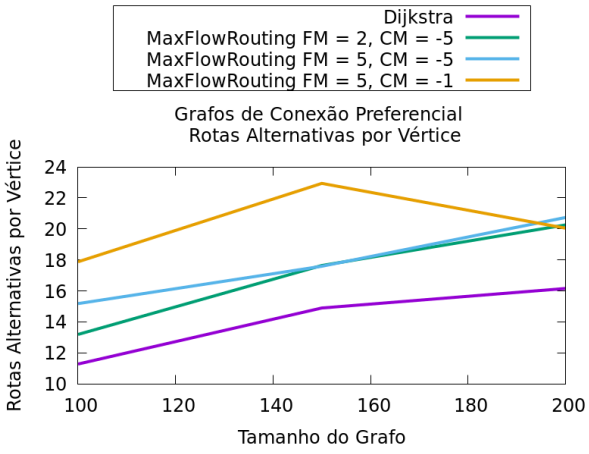


Figura 6.4: Comparação da quantidade média de rotas alternativas por vértice. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.

A Figura 6.4 mostra que o algoritmo *MaxFlowRouting* produz rotas com uma quantidade média de rotas alternativas por vértice maior do que o algoritmo de Dijkstra. Entretanto, o par de pesos FM = 5 e CM = -1 fez com que o *MaxFlowRouting* produzisse resultados com alta variabilidade conforme N varia, partindo de 17,88 rotas alternativas por vértice em grafos de tamanho $N = 100$, indo para 22,94 em grafos de tamanho $N = 150$, mas caindo para 20,05 em grafos de tamanho $N = 200$. Nos grafos de tamanho 150, o algoritmo com esse par de pesos produziu rotas com 1,54 vezes mais rotas alternativas por vértice do que as rotas computadas pelo algoritmo de Dijkstra.

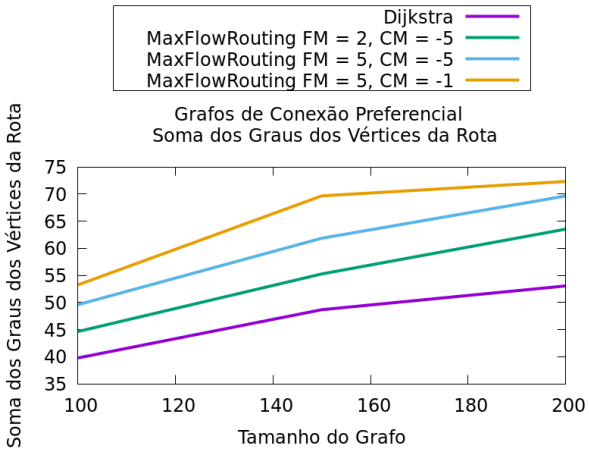


Figura 6.5: Comparação da média da soma dos graus dos vértices das rotas produzidas. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.

A Figura 6.5 mostra que, em média, a soma dos graus dos vértices das rotas produzidas pelo algoritmo *MaxFlowRouting* é maior do que a das rotas produzidas pelo algoritmo de Dijkstra. Em todos os algoritmos, o valor da soma cresceu conforme N aumentou. O algoritmo *MaxFlowRouting* com o par de pesos FM=5 e CM=-1 produziu as rotas com a maior soma de graus dos vértices. Nos grafos de tamanho 150, esse algoritmo produziu rotas que, em média, possuem uma soma de graus dos vértices 1,43 vezes maior do que as rotas produzidas pelo algoritmo de Dijkstra.

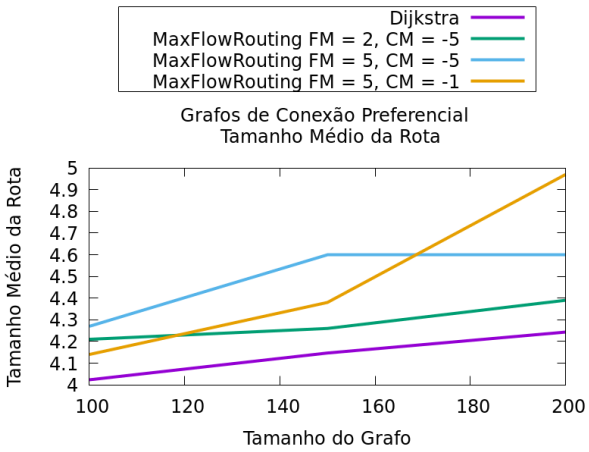


Figura 6.6: Comparação do tamanho médio das rotas produzidas. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.

Tabela 6.3: Resultados do Primeiro Experimento com Grafos de Pequeno Mundo.

N	Algoritmo	Tam. Médio	Soma de Graus	Rotas Alternativas
100	Dijkstra	5,17	22,61	2,48
100	*MaxFlowRouting* com 2,-5	5,61	26,41	3,06
100	*MaxFlowRouting* com 5,-5	6,07	28,50	2,96
100	*MaxFlowRouting* com 5,-1	10,76	54,91	3,37
150	Dijkstra	5,54	24,26	2,49
150	*MaxFlowRouting* com 2,-5	5,96	27,11	2,80
150	*MaxFlowRouting* com 5,-5	6,47	31,77	3,22
150	*MaxFlowRouting* com 5,-1	13,98	70,97	3,28
200	Dijkstra	5,84	25,36	2,43
200	*MaxFlowRouting* com 2,-5	6,16	28,57	2,91
200	*MaxFlowRouting* com 5,-5	7,08	34,08	3,05
200	*MaxFlowRouting* com 5,-1	15,46	74,51	2,94

A Figura 6.6 mostra que o algoritmo *MaxFlowRouting* com os pesos FM = 2 e CM = -5 produz rotas de tamanho médio pouco maior do que as rotas produzidas pelo algoritmo de Dijkstra. Já o *MaxFlowRouting* com os pesos FM = 5 e CM = -1 apresentou um aumento significativo no tamanho das rotas conforme o tamanho N do grafo aumentou. Nos grafos de tamanho 150, porém, o tamanho das rotas produzidas por esse algoritmo foi apenas 1,06 vezes maior do que as rotas computadas pelo algoritmo de Dijkstra.

6.1.3 Grafos de Pequeno Mundo

Os Grafos de Pequeno Mundo de Watts-Strogatz (Watts e Strogatz, 1998) são grafos de N vértices, gerados a partir de uma topologia em forma de anel em que cada vértice é conectado a seus k vizinhos mais próximos. Com uma probabilidade p, cada aresta (u, v) entre vizinhos u e v é substituída por uma aresta (u, t), em que t é um outro vértice do grafo, selecionado aleatoriamente. No experimento, foram utilizados grafos de tamanho $N = 100, 150$ e 200, com vértices com $k = 4$ vizinhos e probabilidade $p = 0,4$ das arestas serem substituídas. Foram computadas rotas entre todos os vértices do grafo que não possuem aresta entre si. As métricas foram calculadas considerando casos em que as rotas produzidas por cada algoritmo diferem. Os resultados obtidos com o experimento estão apresentados na Tabela 6.3.

A Figura 6.7 mostra que o número médio de rotas alternativas por vértice das rotas produzidas pelo *MaxFlowRouting* ultrapassa o valor obtido pelas rotas computadas pelo algoritmo de Dijkstra. Entretanto, é possível notar uma alta variabilidade nos valores conforme N varia. As rotas computadas pelo algoritmo *MaxFlowRouting* com pesos FM=5 e CM=-1 apresentaram uma média de 3,37 rotas alternativas por vértice em $N = 100$, caindo para 3,28 em $N = 150$ e ainda mais para 2,94 em $N = 200$.

A Figura 6.8 mostra que a maior média de soma de graus dos vértices ocorre em rotas computadas pelo algoritmo *MaxFlowRouting* com o par de pesos FM = 5 e CM = -1. Isso é esperado, pois esse par de pesos favorece a conectividade nas rotas. Com outros pares de pesos, o *MaxFlowRouting* também mostrou uma vantagem comparado ao algoritmo de Dijkstra, considerando essa métrica.

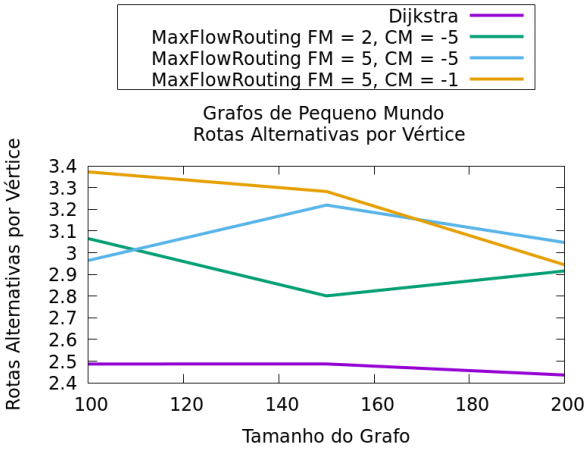


Figura 6.7: Comparação da quantidade média de rotas alternativas por vértice. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.

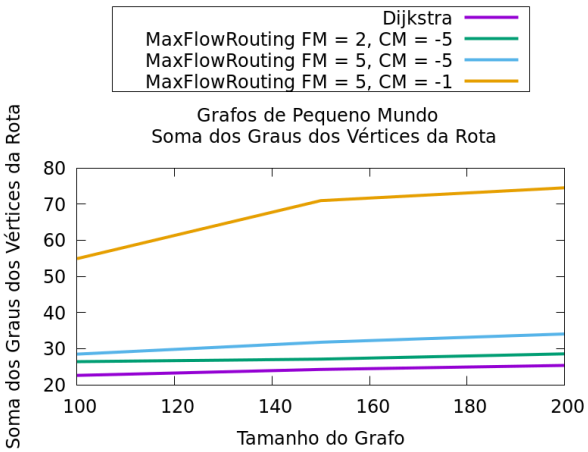


Figura 6.8: Comparação da média da soma dos graus dos vértices das rotas produzidas. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.

A Figura 6.9 mostra que as rotas computadas pelo *MaxFlowRouting* com pesos FM = 2 e CM = -5 possuem um tamanho médio menor comparadas às rotas produzidas pelo algoritmo com os outros pares de pesos (FM = 5 e CM = -5 e FM = 5 e CM = -1). No melhor caso, as rotas computadas pelo *MaxFlowRouting* foram um pouco maiores do que as rotas produzidas pelo algoritmo de Dijkstra, que computa rotas com o menor tamanho possível.

6.1.4 Topologias Reais da Internet

Os experimentos foram executados em grafos representando as seguintes topologias reais da Internet: Internet2 (Internet2, 2024), com $N = 54$ vértices; Géant (GÉANT, 2024), com $N = 44$ vértices; RNP (RNP, 2024), com $N = 28$ vértices; e Wide (WIDE Project, 2024), com $N = 14$ vértices. Foram computadas rotas entre todos os vértices do grafo que não possuem aresta entre si. As métricas foram calculadas somente considerando casos em que as rotas produzidas por cada algoritmo diferem. Os resultados experimentais estão apresentados na Tabela 6.4.

A Figura 6.10 mostra que o algoritmo *MaxFlowRouting* produz rotas com mais rotas alternativas por vértice do que o algoritmo de Dijkstra com os 3 pares de pesos. É possível ver

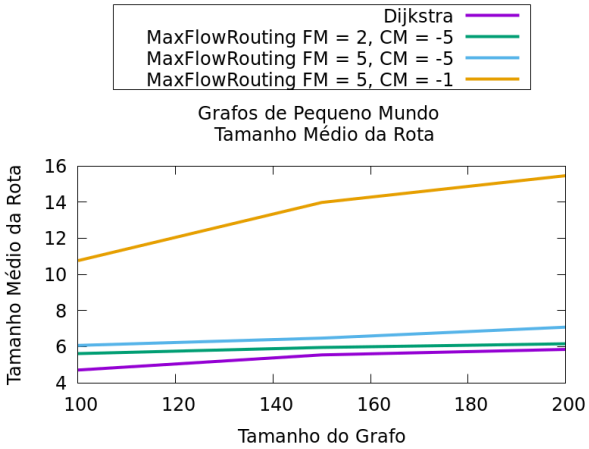


Figura 6.9: Comparação do tamanho médio das rotas produzidas. Roxo: Dijkstra. Verde: *MaxFlowRouting* FM = 2, CM = -5. Azul: *MaxFlowRouting* FM = 5, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -1.

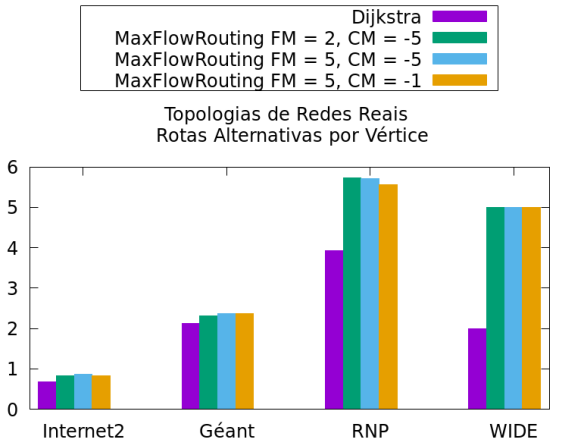


Figura 6.10: Comparação da quantidade média de rotas alternativas por vértice. Da esquerda para a direita: Internet2, Géant, RNP, Wide. Roxo: Dijkstra. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.

também que a variação nos resultados obtidos pelo *MaxFlowRouting* com os diferentes pares de pesos é pequena. A quantidade de rotas alternativas por vértice varia de uma topologia para outra.

Como apresentado na Figura 6.11, o algoritmo *MaxFlowRouting* computa rotas com a maior média de soma de graus dos vértices com o par de pesos FM = 5 e CM = -1. O algoritmo *MaxFlowRouting* com outros pares de pesos também ultrapassa o algoritmo de Dijkstra nesse quesito, com destaque para a topologia Wide.

A Figura 6.12 mostra que as rotas computadas pelo algoritmo *MaxFlowRouting* com o par de pesos FM = 2 e CM = -5 são menores quando comparadas às rotas produzidas pelo algoritmo com os outros pares de pesos em quase todas as topologias. As rotas produzidas com esse par de pesos são pouco maiores do que as rotas computadas pelo algoritmo de Dijkstra, que produz as rotas com o menor tamanho possível. Na topologia Internet2, o par de pesos FM = 5 e CM = -5 fez com que o algoritmo tivesse o melhor desempenho nesse quesito comparado aos outros pares de pesos.

Tabela 6.4: Resultados do Primeiro Experimento com Grafos Representando Topologias Reais da Internet.

Topologia	Algoritmo	Tam. Médio	Soma de Graus	Rotas Alternativas
Internet2	Dijkstra	9,32	24,74	0,68
Internet2	*MaxFlowRouting* com 2,-5	9,72	26,76	0,83
Internet2	*MaxFlowRouting* com 5,-5	9,49	26,38	0,87
Internet2	*MaxFlowRouting* com 5,-1	10,42	28,70	0,83
Géant	Dijkstra	6,37	23,94	2,14
Géant	*MaxFlowRouting* com 2,-5	6,46	25,16	2,32
Géant	*MaxFlowRouting* com 5,-5	6,77	26,42	2,36
Géant	*MaxFlowRouting* com 5,-1	7,61	30,11	2,37
RNP	Dijkstra	4,39	21,58	3,93
RNP	*MaxFlowRouting* com 2,-5	4,44	25,79	5,73
RNP	*MaxFlowRouting* com 5,-5	4,42	25,54	5,71
RNP	*MaxFlowRouting* com 5,-1	4,74	27,02	5,57
WIDE	Dijkstra	3,44	12,22	2,00
WIDE	*MaxFlowRouting* com 2,-5	3,66	17,66	5,00
WIDE	*MaxFlowRouting* com 5,-5	3,83	18,83	5,00
WIDE	*MaxFlowRouting* com 5,-1	3,83	18,83	5,00

6.2 COMPARAÇÃO DO COMPORTAMENTO DOS ALGORITMOS COM OCORRÊNCIA DE FALHAS

No segundo experimento, o comportamento dos algoritmos *MaxFlowRouting* e do algoritmo de Dijkstra com e sem o uso de FRR é comparado quando ocorre uma falha em algum vértice da rota calculada. Na ocorrência de uma falha, o algoritmo de Dijkstra sem FRR retorna até a origem e computa uma nova rota considerando a topologia atualizada com a falha. O algoritmo *MaxFlowRouting* e o algoritmo de Dijkstra com FRR reparam a falha localmente empregando uma rota alternativa. O algoritmo *MaxFlowRouting* foi executado com os seguintes pares de pesos para os critérios de Fluxo Máximo (FM) e Caminho Mínimo (CM): FM=2 e CM=-5, FM=5 e CM=-5 e FM=5 e CM=-1, os mesmos utilizados nos experimentos descritos na Seção 6.1.

No experimento, inicialmente a rota é computada sem a ocorrência de falhas. Após isso, escolhe-se um vértice aleatório (com exceção da origem ou do destino) para falhar, e cada algoritmo recalcula a rota considerando a falha no vértice escolhido. Foram utilizadas 3 métricas para comparar o comportamento dos algoritmos. A primeira delas é o tamanho médio da rota sem a ocorrência da falha. A segunda é o tamanho médio da rota com a ocorrência da falha. A terceira métrica é a quantidade média de *backtracks* que ocorrem. Essa métrica é computada com o uso de um contador no algoritmo de roteamento, que é inicializado com 0. O contador é incrementado quando um vértice esgota todas as suas possibilidades de roteamento e precisa retornar para um vértice anterior.

A Subseção 6.2.1 apresenta os resultados experimentais obtidos com a execução dos algoritmos de roteamento nos Grafos Aleatórios de Conectividade Variável. A Subseção 6.2.2 apresenta os resultados obtidos em Grafos de Conexão Preferencial. A Subseção 6.2.3 apresenta os resultados considerando os Grafos de Pequeno Mundo. Por fim, a Subseção 6.2.4 apresenta os resultados experimentais obtidos com a execução dos algoritmos nos grafos representando as topologias reais da Internet que foram apresentadas anteriormente.

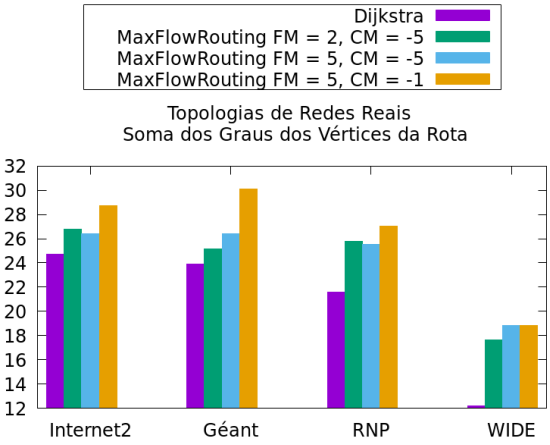


Figura 6.11: Comparação da média da soma dos graus dos vértices das rotas produzidas. Da esquerda para a direita: Internet2, Géant, RNP, Wide. Roxo: Dijkstra. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.

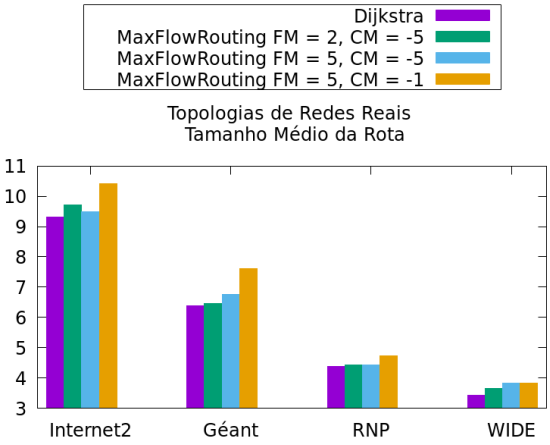


Figura 6.12: Comparação do tamanho médio das rotas produzidas. Da esquerda para a direita: Internet2, Géant, RNP, Wide. Roxo: Dijkstra. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.

6.2.1 Grafos Aleatórios de Conectividade Variável

No segundo experimento, foram utilizados Grafos Aleatórios de Conectividade Variável de Erdos-Renyi (Erdos e Rényi, 1960) com N vértices e com probabilidade C de existir uma aresta ligando quaisquer dois vértices. Os algoritmos foram executados em grafos de tamanho $N = 100, 150$ e 200 e com níveis de conectividade $C = 0,05$ e $0,50$. Para cada valor de N e C, foram gerados 6 grafos diferentes. Em cada um deles, cada algoritmo de roteamento foi executado para computar rotas entre 6 pares de vértices não-adjacentes selecionados aleatoriamente. Os resultados obtidos com o experimento estão apresentados na Tabela 6.5.

Como mostrado na Figura 6.13, no grafo com conectividade $C = 0,05$, houve uma ocorrência maior de *backtracks* na execução do algoritmo de Dijkstra sem o uso do FRR. À medida que o tamanho dos grafos cresceu, a quantidade de *backtracks* em todos os algoritmos de roteamento diminuiu.

A Figura 6.14 mostra que a ocorrência de uma falha faz com que as rotas produzidas por todos os algoritmos sejam maiores em comparação com o cenário sem a ocorrência de falhas. O algoritmo de Dijkstra produz as rotas com o menor tamanho possível nos dois casos. O algoritmo que produziu as rotas de maior tamanho médio foi o *MaxFlowRouting* com os pesos

Tabela 6.5: Resultados do Segundo Experimento com Grafos Aleatórios de Conectividade Variável.

N	*C*	Algoritmo	*Backtracks*	Tam. Médio Sem Falha	Tam. Médio com Falha
100	0,05	Dijkstra sem FRR	0,64	4,14	4,89
100	0,05	Dijkstra com FRR	0,03	4,14	5,53
100	0,05	*MaxFlowRouting* com 2,-5	0,00	4,14	5,64
100	0,05	*MaxFlowRouting* com 5,-5	0,00	4,44	6,30
100	0,05	*MaxFlowRouting* com 5,-1	0,06	5,86	8,28
100	0,50	Dijkstra sem FRR	0,00	3,00	3,00
100	0,50	Dijkstra com FRR	0,00	3,00	3,00
100	0,50	*MaxFlowRouting* com 2,-5	0,00	3,00	3,03
100	0,50	*MaxFlowRouting* com 5,-5	0,00	3,03	3,03
100	0,50	*MaxFlowRouting* com 5,-1	0,00	3,03	3,03
150	0,05	Dijkstra sem FRR	0,61	3,86	4,44
150	0,05	Dijkstra com FRR	0,03	3,86	4,53
150	0,05	*MaxFlowRouting* com 2,-5	0,00	3,97	4,89
150	0,05	*MaxFlowRouting* com 5,-5	0,00	4,28	5,14
150	0,05	*MaxFlowRouting* com 5,-1	0,00	4,33	5,69
150	0,50	Dijkstra sem FRR	0,00	3,00	3,00
150	0,50	Dijkstra com FRR	0,00	3,00	3,00
150	0,50	*MaxFlowRouting* com 2,-5	0,00	3,00	3,00
150	0,50	*MaxFlowRouting* com 5,-5	0,00	3,00	3,00
150	0,50	*MaxFlowRouting* com 5,-1	0,00	3,00	3,00
200	0,05	Dijkstra sem FRR	0,28	3,64	4,14
200	0,05	Dijkstra com FRR	0,00	3,64	4,33
200	0,05	*MaxFlowRouting* com 2,-5	0,00	4,00	4,83
200	0,05	*MaxFlowRouting* com 5,-5	0,00	5,00	6,03
200	0,05	*MaxFlowRouting* com 5,-1	0,00	5,64	7,14
200	0,50	Dijkstra sem FRR	0,00	3,00	3,00
200	0,50	Dijkstra com FRR	0,00	3,00	3,00
200	0,50	*MaxFlowRouting* com 2,-5	0,00	3,00	3,00
200	0,50	*MaxFlowRouting* com 5,-5	0,00	3,00	3,00
200	0,50	*MaxFlowRouting* com 5,-1	0,00	3,00	3,03

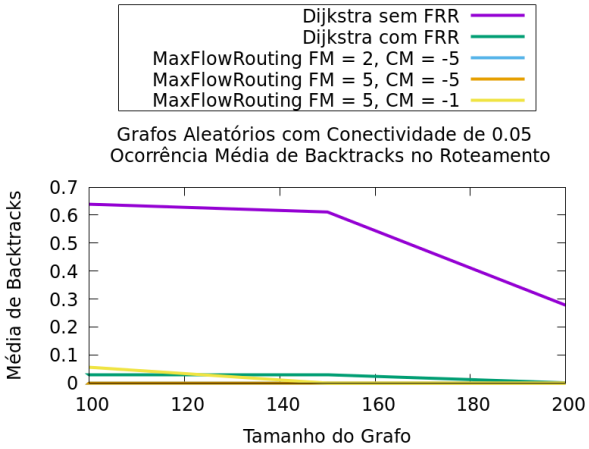


Figura 6.13: Comparação da ocorrência média de *backtracks* nos algoritmos de roteamento. Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.

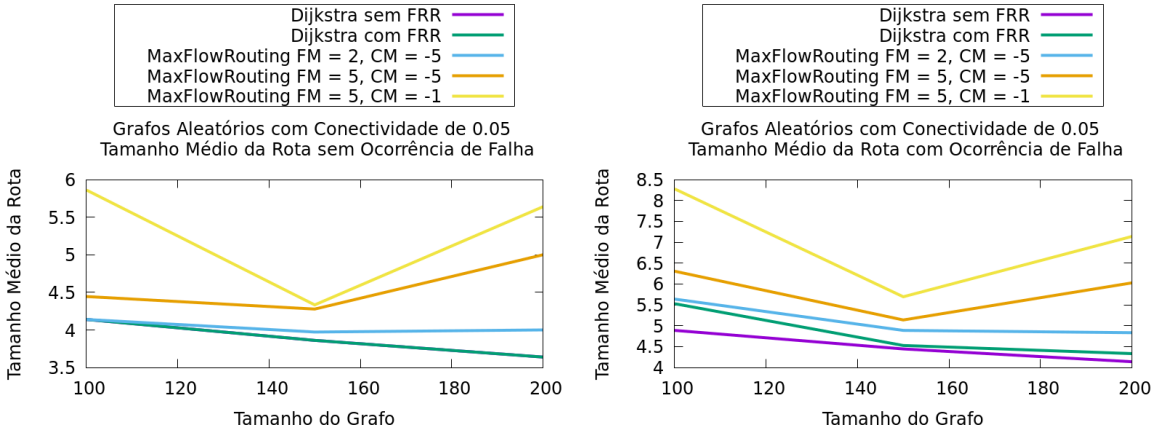


Figura 6.14: Comparação do tamanho médio das rotas produzidas nos cenários sem a ocorrência de falha (esquerda) e com a ocorrência de uma falha (direita). Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.

FM = 5 e CM = -1. Isso é esperado, pois esse par de pesos favorece a conectividade das rotas em detrimento da eficiência em encontrar rotas de tamanho menor.

6.2.2 Grafos de Conexão Preferencial

No segundo experimento, foram utilizados Grafos de Conexão Preferencial de Barábasi-Albert (Albert e Barabási, 2002) de tamanho N, cujos vértices possuem m arestas. O experimento foi executado com grafos de tamanho $N = 100, 150$ e 200 cujos vértices possuem $m = 3$ arestas. Para cada valor de N, foram gerados 6 grafos diferentes. Para cada um deles, cada algoritmo de roteamento computou rotas entre 6 pares de vértices não-vizinhos selecionados aleatoriamente. Os resultados experimentais obtidos são apresentados na Tabela 6.6.

A Figura 6.15 mostra que em média, o algoritmo de Dijkstra sem FRR precisou fazer mais *backtracks* do que os outros algoritmos de roteamento. Considerando essa métrica, houve pouca diferença nos resultados produzidos pelos algoritmos que utilizam FRR.

Considerando as métricas de tamanho médio das rotas nos cenários sem e com ocorrência de falha, a Figura 6.16 mostra que o algoritmo *MaxFlowRouting* com pesos FM = 2 e CM = -5

Tabela 6.6: Resultados do Segundo Experimento com Grafos de Conexão Preferencial.

N	Algoritmo	*Backtracks*	Tam. Médio Sem Falha	Tam. Médio com Falha
100	Dijkstra sem FRR	0,42	3,75	4,22
100	Dijkstra com FRR	0,00	3,75	4,44
100	*MaxFlowRouting* com 2,-5	0,00	3,75	4,44
100	*MaxFlowRouting* com 5,-5	0,03	3,75	5,22
100	*MaxFlowRouting* com 5,-1	0,03	3,75	5,67
150	Dijkstra sem FRR	0,39	3,78	4,22
150	Dijkstra com FRR	0,03	3,78	4,58
150	*MaxFlowRouting* com 2,-5	0,03	3,92	4,83
150	*MaxFlowRouting* com 5,-5	0,03	4,56	4,97
150	*MaxFlowRouting* com 5,-1	0,03	4,64	5,61
200	Dijkstra sem FRR	0,58	4,14	4,53
200	Dijkstra com FRR	0,00	4,14	4,92
200	*MaxFlowRouting* com 2,-5	0,00	4,19	4,94
200	*MaxFlowRouting* com 5,-5	0,00	4,25	5,39
200	*MaxFlowRouting* com 5,-1	0,00	4,25	5,22

produziu rotas com tamanho médio pouco maior do que o algoritmo de Dijkstra com FRR. O algoritmo de Dijkstra sem FRR produziu as rotas de menor tamanho médio nos dois cenários.

6.2.3 Grafos de Pequeno Mundo

No segundo experimento, foram utilizados Grafos de Pequeno Mundo de Watts-Strogatz (Watts e Strogatz, 1998) de tamanho N, cujos vértices possuem k vizinhos iniciais. Cada vizinho inicial de um vértice é substituído com probabilidade p por outro vértice do grafo selecionado aleatoriamente. Os experimentos foram executados com grafos de tamanho $N = 100, 150$ e 200 cujos vértices possuem $k = 4$ arestas e com uma probabilidade de substituição de arestas $p = 0,4$. Para cada valor de N, 6 grafos diferentes foram gerados. Em cada um deles, cada algoritmo de roteamento produziu rotas entre 6 pares de vértices não-adjacentes selecionados aleatoriamente. Os resultados do experimento realizado são apresentados na Tabela 6.7.

Como mostrado na Figura 6.17, é notável que o algoritmo *MaxFlowRouting* com pesos FM=5 e CM=-1 apresentou alta variabilidade considerando a métrica de ocorrência média de *backtracks* no roteamento. Nos grafos de tamanho 150, o algoritmo teve que fazer, em média, 3,39 *backtracks* por roteamento, comparado a 0,00 nos grafos de tamanho 100 e a 0,06 nos grafos de tamanho 200. Nos demais casos, o algoritmo de Dijkstra sem o uso de FRR teve uma média maior de ocorrência de *backtracks* do que os algoritmos que utilizam FRR. Além disso, nos grafos de tamanho 100, o *MaxFlowRouting* fez 62,5% menos *backtracks* do que o algoritmo de Dijkstra com FRR.

A Figura 6.18 mostra que o algoritmo *MaxFlowRouting* com pesos FM=5 e CM=-1 produziu rotas com tamanho médio muito maior do que as rotas produzidas pelos demais algoritmos. Além disso, o *MaxFlowRouting* com o par de pesos FM=2 e CM=-5 produziu rotas de tamanho médio com pouca diferença em comparação às rotas produzidas pelo algoritmo de Dijkstra com FRR.

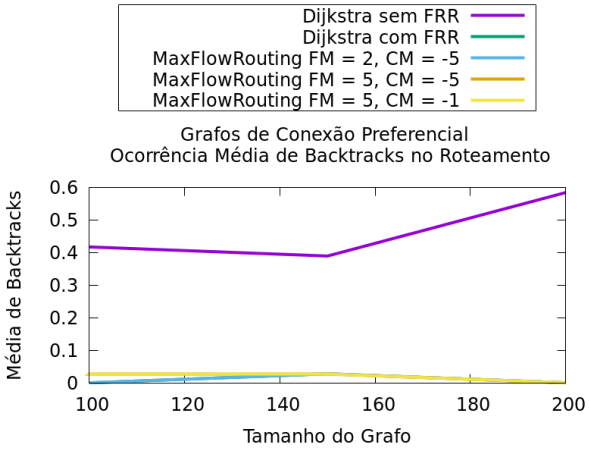


Figura 6.15: Comparação da ocorrência média de *backtracks* nos algoritmos de roteamento. Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.

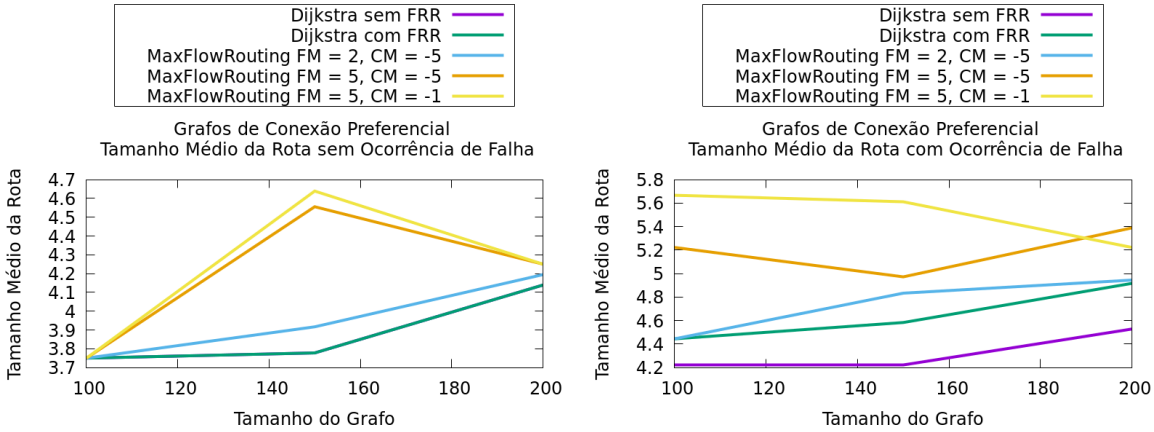


Figura 6.16: Comparação do tamanho médio das rotas produzidas nos cenários sem a ocorrência de falha (esquerda) e com a ocorrência de uma falha (direita). Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.

6.2.4 Topologias Reais da Internet

O segundo experimento também foi executado em grafos representando as seguintes topologias reais da Internet: Internet2 (Internet2, 2024), com $N = 54$ vértices; Géant (GÉANT, 2024), com $N = 44$ vértices; RNP (RNP, 2024), com $N = 28$ vértices; e Wide (WIDE Project, 2024), com $N = 14$ vértices. Em cada topologia, cada algoritmo de roteamento computou rotas entre 30 pares de vértices que não possuem aresta entre si. Os resultados experimentais estão apresentados na Tabela 6.8.

A Figura 6.19 mostra que houve bastante variabilidade na ocorrência de *backtracks* em cada algoritmo de roteamento nas diferentes topologias. O algoritmo *MaxFlowRouting* com pesos FM=5 e CM=-1 apresentou valores relativamente altos nas topologias Internet2 e RNP, enquanto apresentou o menor número médio de *backtracks* na topologia Wide dentre os algoritmos que utilizam FRR.

Na Figura 6.20, é notável que o algoritmo *MaxFlowRouting* com pesos FM=5 e CM=-1 produziu rotas com o maior tamanho médio nos cenários sem e com a ocorrência de falha. O *MaxFlowRouting* com os outros pares de pesos produziu rotas com pouca diferença de tamanho com relação ao algoritmo de Dijkstra com FRR. O algoritmo de Dijkstra sem FRR produziu

Tabela 6.7: Resultados do Segundo Experimento com Grafos de Pequeno Mundo.

N	Algoritmo	*Backtracks*	Tam. Médio Sem Falha	Tam. Médio com Falha
100	Dijkstra sem FRR	0,97	4,81	5,81
100	Dijkstra com FRR	0,08	4,81	6,83
100	*MaxFlowRouting* com 2,-5	0,03	4,81	7,17
100	*MaxFlowRouting* com 5,-5	0,08	5,08	8,14
100	*MaxFlowRouting* com 5,-1	0,00	8,31	12,22
150	Dijkstra sem FRR	0,92	5,14	5,97
150	Dijkstra com FRR	0,03	5,14	7,39
150	*MaxFlowRouting* com 2,-5	0,03	5,14	6,72
150	*MaxFlowRouting* com 5,-5	0,03	5,47	8,33
150	*MaxFlowRouting* com 5,-1	3,39	10,11	16,61
200	Dijkstra sem FRR	0,97	5,19	6,17
200	Dijkstra com FRR	0,03	5,19	8,06
200	*MaxFlowRouting* com 2,-5	0,03	5,19	8,39
200	*MaxFlowRouting* com 5,-5	0,03	5,47	8,08
200	*MaxFlowRouting* com 5,-1	0,06	8,97	18,44

Tabela 6.8: Resultados do Segundo Experimento com Grafos Representando Topologias Reais da Internet.

Topologia	Algoritmo	*Backtracks*	Tam. Médio Sem Falha	Tam. Médio com Falha
Internet2	Dijkstra sem FRR	1,70	6,63	9,50
Internet2	Dijkstra com FRR	1,10	6,63	10,80
Internet2	*MaxFlowRouting* com 2,-5	1,70	6,63	11,37
Internet2	*MaxFlowRouting* com 5,-5	1,13	6,63	10,53
Internet2	*MaxFlowRouting* com 5,-1	3,80	8,13	14,03
Géant	Dijkstra sem FRR	1,07	4,97	6,47
Géant	Dijkstra com FRR	1,13	4,97	7,53
Géant	*MaxFlowRouting* com 2,-5	0,13	4,97	7,33
Géant	*MaxFlowRouting* com 5,-5	1,50	5,17	8,40
Géant	*MaxFlowRouting* com 5,-1	1,53	5,97	8,43
RNP	Dijkstra sem FRR	0,27	4,00	5,43
RNP	Dijkstra com FRR	0,87	4,00	6,17
RNP	*MaxFlowRouting* com 2,-5	0,80	4,00	6,23
RNP	*MaxFlowRouting* com 5,-5	1,77	4,00	6,30
RNP	*MaxFlowRouting* com 5,-1	3,50	4,23	7,30
Wide	Dijkstra sem FRR	0,60	3,47	2,37
Wide	Dijkstra com FRR	2,10	3,47	2,67
Wide	*MaxFlowRouting* com 2,-5	2,33	3,47	2,20
Wide	*MaxFlowRouting* com 5,-5	2,70	3,50	2,13
Wide	*MaxFlowRouting* com 5,-1	1,47	3,50	2,63

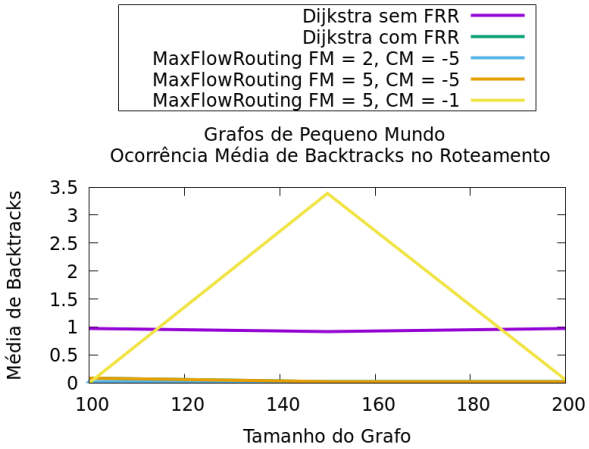


Figura 6.17: Comparação da ocorrência média de *backtracks* nos algoritmos de roteamento. Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.

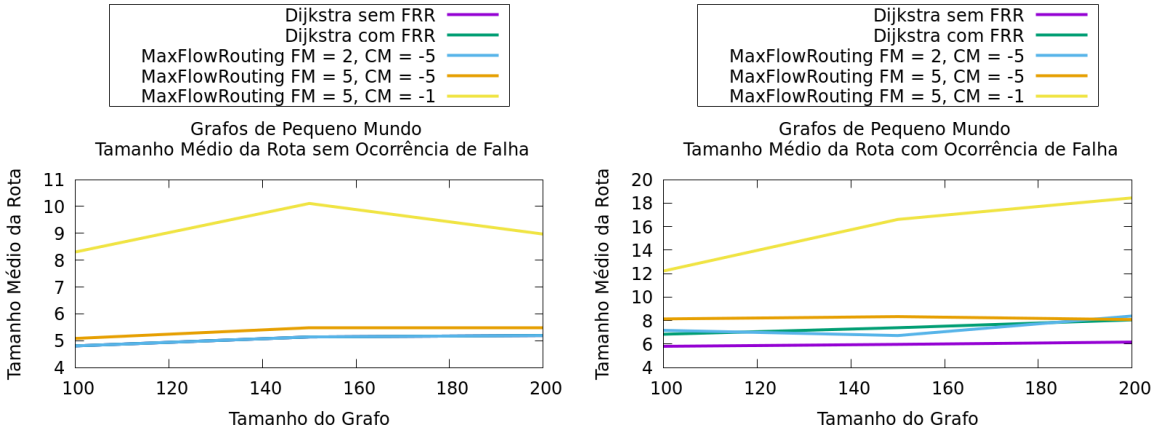


Figura 6.18: Comparação do tamanho médio das rotas produzidas nos cenários sem a ocorrência de falha (esquerda) e com a ocorrência de uma falha (direita). Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.

as rotas de menor tamanho nos cenários sem e com a ocorrência de falha. Na topologia Wide, a ocorrência de falha tornou o roteamento impossível em alguns casos e, por isso, a média do tamanho das rotas com falha é menor do que a média do tamanho sem falha.

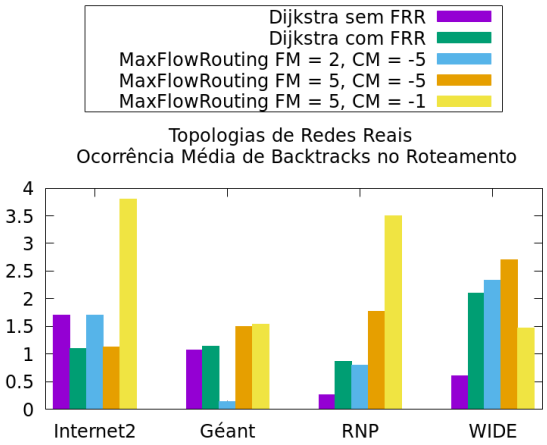


Figura 6.19: Comparação da ocorrência média de *backtracks* nos algoritmos de roteamento. Da esquerda para a direita: Internet2, Géant, RNP e Wide. Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.

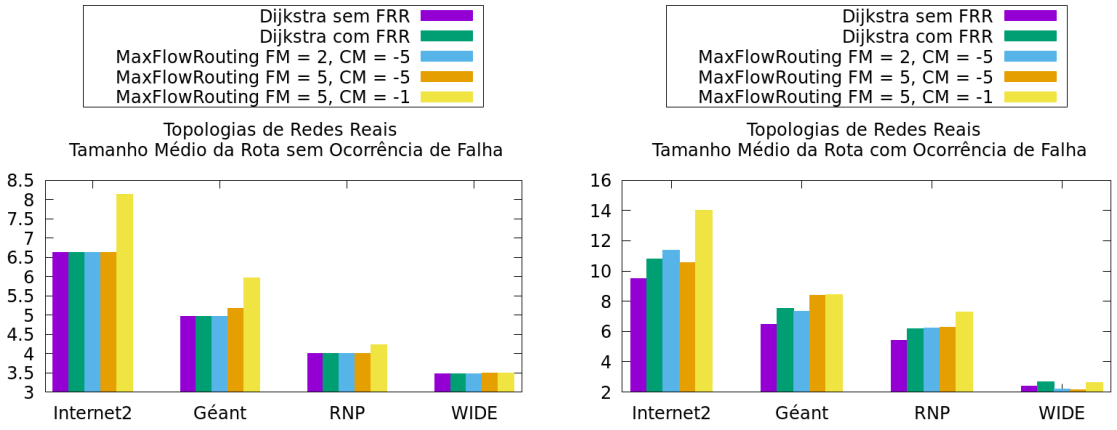


Figura 6.20: Comparação do tamanho médio das rotas produzidas nos cenários sem a ocorrência de falha (esquerda) e com a ocorrência de uma falha (direita). Da esquerda para a direita: Internet2, Géant, RNP e Wide. Roxo: Dijkstra sem FRR. Verde: Dijkstra com FRR. Azul: *MaxFlowRouting* FM = 2, CM = -5. Marrom: *MaxFlowRouting* FM = 5, CM = -5. Amarelo: *MaxFlowRouting* FM = 5, CM = -1.

7 CONCLUSÃO

Este trabalho apresentou o algoritmo *MaxFlowRouting* para roteamento tolerante a falhas. O algoritmo apresentado atua no contexto do *Fast-ReRoute*, que é uma estratégia proativa de reparação de falhas, em que uma rota alternativa é empregada por um roteador quando este detecta uma falha na rota primária. O *MaxFlowRouting* é um algoritmo de roteamento que utiliza a avaliação de fluxo máximo para encontrar rotas com uma quantidade maior de rotas alternativas. Isso acontece porque o fluxo máximo é diretamente proporcional à quantidade de caminhos disjuntos. No trabalho, foi também apresentado um algoritmo de *Fast-ReRoute* baseado em *backtracking*. No algoritmo, um roteador tenta encaminhar um pacote através da rota primária. Se essa rota falha, ele tenta encaminhar o pacote através da rota alternativa. Se essa também falha, o roteador pode retornar o pacote para o roteador anterior, permitindo que ele busque outra rota.

Foram feitos experimentos com simulação para comparar as rotas computadas pelo algoritmo *MaxFlowRouting* com rotas produzidas pelo algoritmo de Dijkstra. Os experimentos foram feitos tanto em grafos aleatórios, como Grafos Aleatórios de Conectividade Variável, Grafos de Conexão Preferencial e Grafos de Pequeno Mundo, bem como em topologias reais da Internet, como a RNP, a Internet2, a Géant e a Wide. Os resultados dos experimentos mostram que o *MaxFlowRouting* produz rotas com mais opções de alternativas do que o algoritmo de Dijkstra. No melhor caso, o algoritmo proposto computou rotas com até 1,54 vezes mais rotas alternativas por vértice e com grau de conectividade 1,43 vezes maior. Nesse caso, as rotas produzidas pelo algoritmo proposto tiveram tamanho médio apenas 1,06 vezes maior do que as rotas produzidas pelo algoritmo de Dijkstra, que produz rotas com o menor tamanho possível. O roteamento computado pelo *MaxFlowRouting* também faz, no melhor caso, até 62,5% menos *backtracks* na ocorrência de falhas na rede do que o roteamento com o algoritmo de Dijkstra.

Trabalhos futuros incluem propor um protocolo que utilize o *MaxFlowRouting*. Além disso, é importante encontrar melhorias para o algoritmo que reduzam a alta variabilidade no desempenho que foi observada nos experimentos.

REFERÊNCIAS

Albert, R. e Barabási, A. (2002). Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47.

Bellman, R. (1958). On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90.

Cohen, J. e Duarte Jr, E. (2001a). Fault-tolerant routing of network management messages in the internet. Em *LANOMS*, páginas 87–98.

Cohen, J. e Duarte Jr, E. (2001b). Fault-tolerant routing of tcp/ip pdu's on general topology backbones. Em *The 3rd IEEE International Workshop on the Design of Reliable Communication Networks (DRCN'2001)*, páginas 303–309.

Cohen, J., Duarte Jr, E. e Schroeder, J. (2011a). Connectivity criteria for ranking network nodes. Em *Complex Networks: Second International Workshop, CompleNet 2010, Rio de Janeiro, Brazil, October 13-15, 2010, Revised Selected Papers*, páginas 35–45. Springer.

Cohen, J., Rodrigues, L. e Duarte Jr, E. (2012). A parallel implementation of gomory-hu's cut tree algorithm. Em *2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*, páginas 124–131. IEEE.

Cohen, J., Rodrigues, L. e Duarte Jr, E. (2017). Parallel cut tree algorithms. *Journal of Parallel and Distributed Computing*, 109:1–14.

Cohen, J., Rodrigues, L., Silva, F., Carmo, R., Guedes, A. L. e Duarte, E. (2011b). Parallel implementations of gusfield's cut tree algorithm. Em *Algorithms and Architectures for Parallel Processing: 11th International Conference, ICA3PP, Melbourne, Australia, October 24-26, 2011, Proceedings, Part I 11*, páginas 258–269. Springer.

Cormen, T., Leiserson, C., Rivest, R. e Stein, C. (2022). *Introduction to Algorithms, fourth edition*. MIT Press.

Deering, S. e Hinden, B. (2017). Internet Protocol, Version 6 (IPv6) Specification. RFC 8200.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

Duarte Jr, E., Garrett, T., Bona, L., Carmo, R. e Züge, A. (2010). Finding stable cliques of PlanetLab nodes. Em *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, páginas 317–322. IEEE.

Duarte Jr, E., Granville, L. Z., Pirmez, L., de Souza, J., Andrade, R., Tarouco, L., Correia, R. e Lages, A. (2012). GigaManP2P: an overlay network for distributed QoS management and resilient routing. *International Journal of Network Management*, 22(1):50–64.

Duarte Jr, E., Rodrigues, L., Camargo, E. e Turchetti, R. (2023). The missing piece: a distributed system-level diagnosis model for the implementation of unreliable failure detectors. *Computing*, 105(12):2821–2845.

Duarte Jr, E., Santini, R. e Cohen, J. (2004). Delivering packets during the routing convergence latency interval through highly connected detours. Em *International Conference on Dependable Systems and Networks (DSN), 2004*, páginas 495–504. IEEE.

Duarte Jr, E. e Weber, A. (2003). A distributed network connectivity algorithm. Em *The Sixth International Symposium on Autonomous Decentralized Systems, 2003. ISADS 2003.*, páginas 285–292. IEEE.

Erdos, P. e Rényi, A. (1960). On the evolution of random graphs. *Publ. math. inst. hung. acad. sci*, 5(1):17–60.

Filsfils, C., Francois, P., Shand, M., Decraene, B., Uttaro, J., Leymann, N. e Horneffer, M. (2012). Loop-free alternate (LFA) applicability in service provider (SP) networks. RFC 6571.

Ford, L. R. e Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404.

Goldberg, A. V. e Tarjan, R. E. (1988). A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940.

GÉANT (2024). GÉANT network. `https://network.geant.org/`. Acessado em 05/11/2024.

Hagberg, A., Schult, D. e Swart, P. (2008). Exploring network structure, dynamics, and function using networkx. Em Varoquaux, G., Vaught, T. e Millman, J., editores, *Proceedings of the 7th Python in Science Conference*, páginas 11 – 15, Pasadena, CA USA.

Harris, T. E. e Ross, F. S. (1955). Fundamentals of a method for evaluating rail net capacities. Relatório técnico, Project RAND, Santa Monica, CA, USA.

Internet2 (2024). Layer 1 service. `https://internet2.edu/services/layer-1/`. Acessado em 05/11/2024.

Maske, C., Cohen, J. e Duarte Jr, E. (2020). Speeding up the gomory-hu parallel cut tree algorithm with efficient graph contractions. *Algorithmica*, 82(6):1601–1615.

Medhi, D. (2010). *Network Routing: Algorithms, Protocols, and Architectures*. ISSN. Elsevier Science.

Moy, J. (1998). OSPF Version 2. RFC 2328.

Nassu, B., Nanya, T. e Duarte Jr, E. (2007). Topology discovery in dynamic and decentralized networks with mobile agents and swarm intelligence. Em *Seventh International Conference on Intelligent Systems Design and Applications (ISDA 2007)*, páginas 685–690. IEEE.

Okida, L., Maverson, E. e Duarte Jr, E. (2024). Fast reroute with highly connected routes based on maximum flow evaluation.

Pan, P., Swallow, G. e Atlas, A. (2005). Fast reroute extensions to RSVP-TE for LSP tunnels. RFC 4090.

Postel, J. (1981). Internet Protocol. RFC 791.

Rekhter, Y., Hares, S. e Li, T. (2006). A Border Gateway Protocol 4 (BGP-4). RFC 4271.

RNP (2024). Ipê network. https://www.rnp.br/en/ipe-network. Acessado em 05/11/2024.

Schroeder, J. e Duarte Jr, E. (2007). Fault-tolerant dynamic routing based on maximum flow evaluation. Em *Latin-American Symposium on Dependable Computing*, páginas 7–24. Springer.

Schroeder, J., Guedes, A. e Duarte Jr, E. (2004). Computing the minimum cut and maximum flow of undirected graphs. *Technical report, Federal University of Paraná*.

Shand, M. e Bryant, S. (2010). IP fast reroute framework. RFC 5714.

Watts, D. J. e Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442.

WIDE Project (2024). WIDE internet official site. https://two.wide.ad.jp/. Acessado em 05/11/2024.