COPPE/UFRJ

ALGORITMOS DESINFORMADOS PARA ROTEAMENTOS EM REDES

Thatiana Fernandes de Oliveira

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Márcia Rosana Cerioli
Claudson Ferreira Bornstein

Rio de Janeiro
Março de 2009

ALGORITMOS DESINFORMADOS PARA ROTEAMENTOS EM REDES

Thatiana Fernandes de Oliveira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Márcia Rosana Cerioli, D.Sc.

Prof. Claudson Ferreira Bornstein, Ph.D.

Prof. Cristina Gomes Fernandes, Ph.D.

Prof. Daniel Ratton Figueiredo, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2009

Oliveira, Thatiana Fernandes de

Algoritmos desinformados para roteamentos em redes/Thatiana Fernandes de Oliveira. – Rio de Janeiro: UFRJ/COPPE, 2009.

IX, 120 p. 29, 7cm.

Orientadores: Márcia Rosana Cerioli

Claudson Ferreira Bornstein

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2009.

Referências Bibliográficas: p. 116 – 120.

1. Algoritmos aproximativos. 2. Roteamento em redes. 3. Congestionamento em redes. I. Cerioli, Márcia Rosana *et al.*. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

A Deus, por tudo que representa em minha vida, e por ter me permitido chegar até aqui.

A André, pela paciência, carinho, dedicação e compreensão. Por ter abdicado, sem reclamar, de tantas horas ao meu lado para que eu pudesse terminar a dissertação e por ter me incentivado a continuar o mestrado quando eu mesma já havia entregue os pontos. Palavras nunca serão suficientes para expressar tudo que sinto...

À minha mãe, que mesmo não compreendendo o porquê de eu guardar tantos papéis "com desenhos feitos de bolinhas e tracinhos e com cálculos esquisitos", que mesmo não entendendo o porquê de eu não ter seguido o curso natural de emprego imediatamente após o término da graduação, ainda assim me incentivava a obter o título de mestre.

A todos os meus amigos. Em especial, a Claudio Leandro, por ter me acompanhado desempenhando com louvor seu papel de melhor amigo; a Lissandra, Ingrid e Evelin, pela eterna amizade, por serem a herança de um dos melhores tempos da minha vida, pelas saídas, pelas dúvidas de todo almoço (onde vamos almoçar?) de todos estes anos; a Eduardo Garcia, por me lembrar que a vida sempre precisa ser levada com humor e conseguir me fazer rir mesmo nas horas mais difíceis. Vocês moram no meu ♡!

Aos meus companheiros de mestrado, em especial os da mesma linha: Adriana, Cris, Danilo, Fabiano, Maíse, Rafael, Raquel, Daniel... e todos aqueles que dividiam o LAC conosco. Aos colegas de outras linhas que em algum momento participaram de minha trajetória no mestrado, seja por ter feito alguma disciplina em comum ou pela simples amizade feita pelos corredores da COPPE. Em especial, a Rodrigo Granja, por partilhar dos mesmos "problemas de mestrando" e entendê-los tão bem.

Aos meus amigos de graduação: Bruno, Morgado, Leonardo, Zé Omar, Tuninho

e Ronaldo, pelos anos juntos no Fundão, e pelos anos após estes; pelos almoços, churrascos, aniversários, chopps, karts e boliches juntos.

Aos meus colegas de trabalho: Celso, Felipe, Gustavo, Luiz, Marylanne, Ovídio, Rafael, Roberto, Waleska e Wesley. Pelas barras seguradas enquanto eu precisava correr com a dissertação, por cobrir (e encobrir!) minhas saídas ao Fundão; pela amizade, companheirismo e torcida nesta reta final do mestrado.

Aos professores da banca, por terem aceitado o convite e terem dado tanta atenção ao texto. Aos meus orientadores, Márcia e Claudson, por terem aceitado orientar este trabalho. Aos demais professores da COPPE, pelos conhecimentos agregados.

Por fim, agradeço aos criadores do COPPETEX, pelo template da dissertação e à CAPES, pela bolsa de mestrado.

> "Andei. Por caminhos difíceis, eu sei. Mas olhando o chão sob meus pés, vejo a vida correr. E, assim, cada passo que der, tentarei fazer o melhor que puder. Aprendi. Não tanto quanto quis, mas vi que, conhecendo o universo ao meu redor, aprendo a me conhecer melhor, e assim escutarei o tempo, que me ensinará a tomar a decisão certa em cada momento. E partirei, em busca de novos ideais. Mas sei que hoje se encontram meu passado, futuro e presente. Hoje sinto em mim a emoção da despedida. Hoje é um ponto de chegada e, ao mesmo tempo, ponto de partida..."
>
> (Fernando Sabino)

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ALGORITMOS DESINFORMADOS PARA ROTEAMENTOS EM REDES

Thatiana Fernandes de Oliveira

Março/2009

Orientadores: Márcia Rosana Cerioli
Claudson Ferreira Bornstein

Programa: Engenharia de Sistemas e Computação

O problema do roteamento desinformado consiste em selecionar caminhos para rotear *commodities* sem considerar a carga da rede. Neste cenário, a escolha dos caminhos depende somente da origem e destino de cada *commodity* e da topologia da rede.

Este trabalho considera o problema do roteamento desinformado com o objetivo de minimizar o congestionamento em redes direcionadas ou não, com capacidades nas arestas ou nos vértices. Nesta dissertação é feito um estudo mais detalhado de dois algoritmos aproximativos baseados na construção de uma árvore de decomposição, ambos para a versão do problema com redes não-direcionadas que possuem capacidades nas arestas. Os algoritmos e as provas de suas razões de competitividade são apresentados em notação unificada, fazendo com que os resultados sejam facilmente comparáveis.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

OBLIVIOUS ALGORITHMS FOR NETWORK ROUTING

Thatiana Fernandes de Oliveira

March/2009

Advisors: Márcia Rosana Cerioli

Claudson Ferreira Bornstein

Department: Systems Engineering and Computer Science

The oblivious routing problem consists of selecting paths to route commodities in a network, without ever considering the load of the network. More precisely, the choice of paths depends only on the source and destination of each commodity and on the network topology.

This work considers the oblivious routing problem that selects paths so as to minimize the congestion on the network links or nodes, on either directed or undirected networks. This dissertation reviews two approximation algorithms for the edge capacitated version of this problem which are based on the construction of decomposition trees. In an attempt to make the results easier to read and compare, the algorithms and the proofs for their approximation guarantees are presented in a unified manner.

Sumário

Capítulo 1

Introdução

O **problema de roteamento** em redes consiste em, dada uma rede e um conjunto de *commodities* (objetos que precisam ser roteados entre pontos da rede), escolher um conjunto de caminhos pelos quais elas serão enviadas de suas respectivas origens até seus destinos. Este problema e suas diversas variantes são conteúdos bastante conhecidos na área de ciência da computação, sendo amplamente estudados nas áreas de algoritmos e redes de telecomunicações [1].

A literatura apresenta várias versões deste problema. Cada uma delas implica certas características ao algoritmo de roteamento, restringindo, por exemplo, a forma como a escolha de caminhos pode ser feita ou o uso de certos dados da instância do problema. Pode-se citar como algumas das variantes mais conhecidas do problema de roteamento: roteamento fracionário ou inteiro, estático ou dinâmico, determinístico ou randomizado, distribuído ou centralizado, entre outras.

O tema desta dissertação é **roteamento desinformado**. Na variante do problema do roteamento considerada neste estudo, não se deseja considerar a carga da rede para a escolha de caminhos para o roteamento. Isto é equivalente a dizer que, ao escolher o caminho para rotear uma *commodity*, não é possível saber quais são as demais *commodities* que estão sendo roteadas na mesma rede, ou seja, a escolha de caminhos para rotear uma *commodity* não pode se basear nas rotas utilizadas por alguma outra. Neste problema, portanto, não é permitido que a escolha de rotas utilize informações sobre o conjunto de *commodities* que deverá ser roteado. O objetivo do problema é produzir um **esquema de roteamento**, o que consiste em atribuir, para cada par de vértices, uma distribuição de probabilidade para os

caminhos que os ligam. Este esquema é usado posteriormente para decidir o **roteamento** de cada *commodity*, tornando portanto a escolha de caminhos para roteá-las dependente unicamente de suas origens e destinos. Um algoritmo que produz desta forma um roteamento é um **algoritmo desinformado**.

Neste trabalho, o foco será sobre algoritmos desinformados que têm como meta minimizar o **congestionamento**, ou seja, minimizar a maior **carga relativa** – proporção da capacidade usada – das arestas (ou vértices) da rede. A forma de roteamento utilizada neste trabalho será **randomizada**, isto é, após calcular o esquema de roteamento, a escolha do caminho utilizado para rotear entre cada par de vértices é feita utilizando um sorteio no qual a probabilidade de cada caminho ser escolhido é dada pela distribuição de probabilidades do esquema de roteamento. Além disso, os algoritmos aqui abordados serão **aproximativos**. Isto significa que estaremos interessados em comparar o valor do congestionamento que o algoritmo fornece com o valor ótimo, para a mesma instância. A qualidade destes algoritmos será medida através da **razão de competitividade** do algoritmo, isto é, a maior razão entre o congestionamento obtido pelo algoritmo e o valor ótimo (calculado por um algoritmo que conhece todas as *commodities* e pode usar esta informação para a escolha de rotas), dentre todas as possíveis instâncias.

Com o avanço dos meios de telecomunicação e a crescente expansão da Internet, nestes últimos anos passou-se a dar mais atenção para o problema de conseguir roteamentos eficientes. Desde meados do século XX a obtenção de algoritmos de roteamento eficientes é amplamente pesquisada e, atualmente, buscam-se modelos matemáticos que se aproximem o máximo possível da realidade [1].

A modelagem matemática comumente utilizada para algoritmos de roteamento é através de **fluxos em redes**. A rede é modelada por um grafo que possui **capacidades** nas arestas (ou vértices) e a escolha dos caminhos usados para o roteamento é feita através de um fluxo ligando o ponto de origem ao destino de cada *commodity*.

Um dos problemas mais conhecidos neste contexto é o problema do **fluxo máximo em redes** [2]. Neste, é dada uma rede onde há uma fonte e um sumidouro, e deseja-se levar a maior quantidade possível de fluxo da origem (fonte) ao destino (sumidouro) sem violar as capacidades das arestas (ou vértices).

Uma generalização do problema do fluxo máximo é o problema do **multifluxo**,

no qual são dadas várias *commodities*, cada uma delas possuindo uma quantidade (chamada de **demanda**) que deverá ser levada do seu ponto de origem ao seu destino. O objetivo deste problema é satisfazer às demandas de todas as *commodities* de forma simultânea, não violando a capacidade das arestas (ou vértices) da rede.

O problema de otimização associado ao multifluxo é chamado de **multifluxo concorrente máximo**. Nele, o objetivo é maximizar $z \in \mathbb{R}$ para o qual existe um multifluxo que envia uma fração z da demanda de cada *commodity*. Note que a fração z é idêntica para todas as *commodities*, isto é, a porcentagem da demanda roteada é a mesma para todas elas.

Embora o problema de encontrar um multifluxo que obtenha o congestionamento ótimo (isto é, o menor congestionamento possível para uma certa instância do problema) tenha solução em tempo polinomial pela resolução de um programa linear [3], para obtê-la é preciso conhecer de antemão quais serão as demandas, o que na maioria das aplicações não é realista.

Para evitar este problema e conseguir um modelo mais próximo da realidade, chegamos ao problema do multifluxo **online** [4]. Neste cenário podemos considerar cada *commodity* como um conjunto de **pedidos de roteamentos** que chegam ao longo do tempo, cada um com uma demanda para ser atendida. Algoritmos que usam este modelo decidem como rotear as *commodities* à medida que estes pedidos de roteamento chegam, contornando assim a necessidade de conhecê-los de antemão.

Algoritmos *online* em geral seguem o modelo de roteamento chamado de **adaptativo**, o que significa que os caminhos escolhidos para rotear as *commodities* podem mudar em função de alguma alteração ocorrida durante a execução do algoritmo. Nesta abordagem, por exemplo, a chegada de um novo pedido de roteamento pode alterar as rotas previamente escolhidas entre qualquer outro par de vértices da rede. Como a modificação do roteamento é feita em virtude de eventos ocorridos ao longo da execução do algoritmo, visando adaptar o roteamento a cada novo cenário, estes algoritmos geralmente obtêm bom desempenho. No entanto, em alguns casos suas implementações são bastante complicadas, pois por vezes as mudanças de rotas envolvem muitas variáveis compartilhadas e trocas de mensagens de controle entre os nós da rede.

Todos os algoritmos mencionados acima são considerados **informados**, visto que

para escolherem as rotas estes utilizam informações do conjunto de *commodities* ou de informações sobre o estado da rede no momento em que o pedido de roteamento é feito.

No entanto, desejamos um algoritmo em que a escolha das rotas seja facilmente implementada, independente do estado da rede ou do conjunto de *commodities*. Neste caso, uma alternativa é usar uma abordagem **desinformada**, o que significa que, além do algoritmo não considerar de antemão o conjunto de *commodities* a serem roteadas, não utilizará informações sobre o estado atual da rede. As rotas escolhidas pelo algoritmo entre cada par de vértices, serão, portanto, dependentes unicamente de sua origem e destino.

Com estas propriedades, um algoritmo desinformado possui duas características importantes para um roteamento eficiente:

- simplicidade: as decisões de roteamento se tornam simples em virtude da existência de um conjunto de rotas fixas entre cada par de vértices. A cada pedido de roteamento entre um par de vértices, uma das rotas que os ligam é escolhida com certa probabilidade, que é fixa e definida pelo esquema de roteamento desinformado;

- localidade: uma vez definido o esquema de roteamento e sorteada uma rota com a probabilidade estipulada pelo esquema de roteamento desinformado, as decisões de roteamento em cada vértice da rede são feitas localmente por consultas a uma tabela estática.

Ademais, desejamos que este roteamento possua bom desempenho para alguma métrica pré-estabelecida. Neste trabalho estudamos a minimização do congestionamento, ou seja, a minimização da maior **carga relativa** — proporção da capacidade usada no roteamento — dos componentes da rede. O congestionamento da rede pode ser maior do que 1, já que o roteamento desinformado precisa rotear toda a demanda das *commodities*, não necessariamente respeitando as capacidades dos componentes da rede usados pelas rotas.

Como exemplos de objetivos diferentes estudados na literatura do problema do roteamento desinformado, tem-se, por exemplo, a minimização da latência média [5]. Nesta versão do problema, a utilização de uma aresta e gera uma latência r_e no

roteamento de uma *commodity* e o objetivo é escolher rotas de forma a minimizar a soma das latências geradas no roteamento de um conjunto de *commodities* de uma certa instância. Outra função objetivo bastante estudada é a maximização da quantidade de pedidos de roteamento atendidos em um roteamento viável (com penalidades por pedidos não-atendidos [6] ou sem penalidades [7, 8]). Neste caso, o roteamento desinformado deve respeitar as capacidades das arestas e cada pedido de roteamento pode ser atendido ou recusado. O objetivo é encontrar uma estratégia de roteamento desinformado que maximize a quantidade de pedidos de roteamento atendidos, porém sem conhecê-los de antemão.

Como em muitas situações reais a falta de conhecimento prévio sobre as *commodities* do problema é comum, algoritmos desinformados podem ser utilizados em diversas aplicações. Os usos mais representativos destes algoritmos são na modelagem de problemas das áreas de telecomunicações, transportes e redes de computadores. Algoritmos desinformados são comumente utilizados como modelos simplificados para roteamento *online* de circuitos virtuais [9]. Neste problema os pedidos de roteamento chegam na rede de forma *online*, cada um deles especificando sua origem, destino e uma quantidade de largura de banda necessária para atendê-lo, e o algoritmo de roteamento precisa escolher um caminho entre estes dois pontos. O objetivo de minimizar o congestionamento, neste caso, evita a criação de pontos em que a carga da rede é alta.

Do ponto de vista teórico, este assunto se torna especialmente interessante porque, mesmo com a dificuldade de não se conhecer as *commodities* de antemão, é possível obter um algoritmo de tempo polinomial para minimizar o valor do congestionamento. É um resultado surpreendente que a razão de competitividade destes algoritmos possa ser relativamente pequena.

Nesta dissertação serão abordados alguns dos principais trabalhos existentes na literatura sobre algoritmos desinformados randomizados que tenham por objetivo a minimização do congestionamento. Será dada atenção maior aos algoritmos existentes na literatura do problema que fornecem uma construção intuitiva do roteamento, usando ferramentas como cortes em grafos e decomposições hierárquicas. Os pré-requisitos para a leitura deste trabalho são conhecimentos básicos sobre fluxo em redes [10], programação linear [11] e teoria de grafos [10].

Este trabalho está organizado da seguinte forma: no próximo capítulo são dadas as definições básicas que serão usadas ao longo do texto, bem como a formalização e um histórico do problema do roteamento desinformado; no capítulo 3 são apresentadas as demonstrações dos limites inferiores para a razão de competitividade de algoritmos desinformados; o capítulo 4 trata dos algoritmos desinformados cujas redes de entrada são digrafos com capacidades nas arestas, digrafos com capacidades nos vértices e grafos com capacidades nos vértices; o capítulo 5 introduz o conceito de árvore de *decomposição* e o método de roteamento desinformado usando esta árvore, que serão usados nos algoritmos para redes compostas de grafos com capacidades nas arestas abordados nos capítulos 6 e 7; por fim, no capítulo 8 são feitas as considerações finais e alguns problemas em aberto são listados.

Capítulo 2

Definições

Este capítulo é dedicado a apresentar as definições que serão usadas ao longo do texto e introduzir formalmente o problema do roteamento desinformado, além de fazer um breve histórico sobre as soluções já apresentadas para este problema.

Antes de abordar o problema do roteamento desinformado, será abordado o problema do multifluxo. Este é um problema clássico na área de fluxos em redes que introduzirá a notação e definições usadas neste trabalho, além de servir como introdução à versão desinformada do roteamento, visto que a versão de otimização do problema do multifluxo e o roteamento desinformado estão fortemente relacionados. No caso em que ambos têm como objetivo minimizar o congestionamento, o problema do roteamento desinformado configura-se em uma variante do problema do multifluxo na qual não é possível usar informações sobre o conjunto de *commodities*. Além disso, para qualquer instância dada como entrada do problema de roteamento desinformado, o congestionamento ótimo é dado pela solução ótima do problema do multifluxo para esta mesma entrada.

Justamente pela restrição de não poder utilizar informações sobre o conjunto de *commodities*, um algoritmo que resolve o problema do multifluxo não pode ser aplicado para obter um roteamento desinformado, o que faz com que o estudo deste último seja feito em separado do primeiro.

Ao longo do texto será usado **rede** como um grafo (ou digrafo) conexo com pesos não-negativos (chamados de **capacidades**) nas arestas ou vértices. O número de vértices da rede será denotado por n e o número de arestas por m. Para redes com capacidades nas arestas, a capacidade de uma aresta e será denotada por $cap(e)$.

Analogamente, $cap(v)$ indicará a capacidade de um vértice v em redes com capacidades nos vértices.

2.1 Multifluxo

Nesta seção o problema do multifluxo será abordado. A fim de simplificar a notação, os problemas e definições serão para redes com capacidades nas arestas. Definições análogas para o caso de redes com capacidades nos vértices se encontram na subseção 2.1.1.

A entrada do problema do multifluxo consiste em uma tupla $\mathcal{R}(G, \mathcal{K})$, onde $G(V, E)$ é uma rede e $\mathcal{K}$ é um conjunto de *commodities* a serem roteadas. Cada *commodity* k consiste de um par de vértices $\langle s_k, t_k \rangle$, que são sua **origem** e seu **destino**, e um número não-negativo d_k, que é sua **demanda**. Esta demanda indica a quantidade desta *commodity* que deve ser transportada de s_k para t_k, como explicado abaixo. A rede G da entrada do problema pode ser um grafo direcionado ou não, com capacidades em arestas ou em vértices, modelando, assim, quatro problemas de multifluxo: rede direcionada com capacidades nas arestas, rede direcionada com capacidades nos vértices, rede não-direcionada com capacidades nas arestas, rede não-direcionada com capacidades nos vértices.

A demanda de cada *commodity* é considerada fixa e levada da sua origem ao seu destino através de caminhos entre estes dois vértices, definindo um **fluxo** na rede. No problema clássico de multifluxo, a demanda poderá ser dividida entre os caminhos, isto é, o fluxo poderá ser fracionado entre múltiplos caminhos da rede. Sendo $N(v)$ a vizinhança de um vértice $v \in V$, um **fluxo** f_k para uma *commodity* $k \in \mathcal{K}$ é definido por:

$$\begin{aligned} \sum_{u \in N(v)} f_k((u,v)) &= \sum_{w \in N(v)} f_k((v,w)) \quad \forall v \in V \setminus \{s_k, t_k\} \\ \sum_{v \in N(s_k)} f_k((s_k,v)) &= d_k \end{aligned}$$

com

$$0 \leq f_k(e) \leq cap(e) \qquad \forall e \in E,$$

onde para cada aresta $e = (u, v)$, $f_k((u, v))$ é a quantidade da *commodity* k que passa de u para v por e e $f_k((v, u))$ é a quantidade da *commodity* k que passa de v

para u por e e a quantidade de fluxo da *commodity* k que passa por e é dado por

$$f_k(e) = f_k((u,v)) + f_k((v,u)).$$

Esta notação valerá tanto para redes não-direcionadas quanto direcionadas, sendo que neste último caso, uma das parcelas da equação anterior é obrigatoriamente igual a zero, já que o fluxo deverá ser unidirecional na aresta.

A solução de um problema de multifluxo com entrada $\mathcal{R}(G,\mathcal{K})$ consiste de um fluxo para cada *commodity* e definimos um **multifluxo** como sendo a família dos fluxos $F = \{f_k \mid k \in \mathcal{K}\}$. Um multifluxo F é **viável** se a soma de todos os fluxos que passam em qualquer aresta não viola a capacidade da mesma, ou seja,

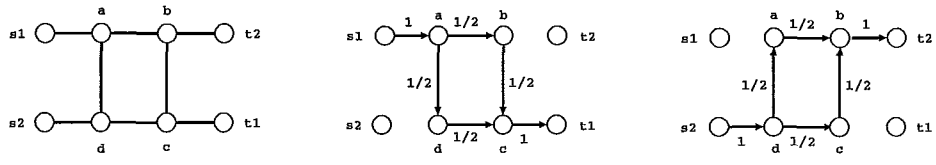
$$0 \leq \sum_{k \in \mathcal{K}} f_k(e) \leq cap(e) \qquad \forall e \in E.$$

Para medir a quantidade total de fluxo que passa por uma aresta (também chamada de carga), soma-se a contribuição de cada fluxo f_k. Esta quantidade é a **carga absoluta** de uma aresta $e = (u,v)$, sendo dada por:

$$Ld(e) = \sum_{k \in \mathcal{K}} f_k(e).$$

Para ilustrar o conceito de multifluxo viável, segue um exemplo:

Exemplo 2.1. *A figura 2.1 apresenta um exemplo de multifluxo viável. A figura 2.1(a) apresenta uma rede onde todas as arestas possuem capacidade igual a* 1. *As duas* commodities *c*1, *c*2, *ambas com demanda igual a 1, possuem pares origem/destino* $\langle s_1, t_1 \rangle$ *e* $\langle s_2, t_2 \rangle$, *respectivamente.*



(a) (b) (c)

Figura 2.1: Exemplo de um multifluxo viável.

As figuras 2.1(b) e 2.1(c) apresentam os fluxos de cada commodity *c*1 *e c*2, *respectivamente. Cada um dos caminhos entre* s_1 *e* t_1 — (s_1, a, b, c, t_1) *e* (s_1, a, d, c, t_1)

— *transporta* $\frac{1}{2}$ *da demanda da* commodity $c1$, *assim como cada um dos caminhos entre* s_2 *e* t_2 — (s_2, d, a, b, t_2) *e* (s_2, d, c, b, t_2) — *transporta* $\frac{1}{2}$ *da demanda da* commodity $c2$. *Temos um multifluxo viável pois* $Ld(e) \leq 1$, *para toda aresta* $e \in E$.

É interessante notar que este exemplo somente tem solução viável porque é permitido dividir uma *commodity* entre caminhos, isto é, cada caminho pode rotear uma fração de uma *commodity*. Caso houvesse a exigência de que a demanda das *commodities* fosse roteada apenas por um caminho, não seria possível obter um multifluxo que atendesse a todas as demandas sem violar a capacidade das arestas.

O problema de otimização relacionado ao problema do multifluxo é chamado de **problema do fluxo concorrente máximo** (***maximum concurrent flow problem*** ou **MCFP**) [12]. Ele consiste em encontrar a maior fração z, isto é, $0 \leq z \leq 1$, tal que existe um conjunto de fluxos $F = \{f_k \mid k \in \mathcal{K}\}$ que roteia uma proporção z de cada uma das demandas. A fração z é chamada de **fração de vazão** da solução considerada.

Note que se $z = 1$, é possível rotear toda a demanda de todas as *commodities* através de fluxo viável. Este seria o caso em que o problema do multifluxo apresentado anteriormente, que é um problema de decisão, encontraria uma solução. Já no caso em que $z < 1$ no problema de otimização, o problema de decisão apresentado anteriormente não encontraria solução, pois não é possível rotear toda a demanda das *commodities* através de um fluxo viável.

Na literatura sobre algoritmos desinformados, este problema também é conhecido como **problema do multifluxo concorrente** (***concurrent multicommodity flow problem*** ou **CMCF**), motivo pelo qual esta notação será usada ao longo do texto.

O CMCF é modelado por um programa linear e, portanto, pode ser resolvido em tempo polinomial:

$$\text{Maximizar } z \text{ sujeito a: } \begin{cases} \sum_{u \in N(v)} f_k((u,v)) = \sum_{w \in N(v)} f_k((v,w)) & \forall v \in V \setminus \{s_k, t_k\} \\ \sum_{v \in N(s_k)} f_k((s_k,v)) = z\, d_k & \forall k \in \mathcal{K} \\ Ld(e) \leq cap(e) & \forall e \in E \end{cases} \tag{2.1}$$

Pode-se também considerar o problema do multifluxo com outras funções ob-

jetivo. O congestionamento, que será o foco dos roteamentos desinformados deste trabalho, é uma delas. Para formalizá-lo matematicamente, será definido o conceito de carga relativa.

Dado um multifluxo F, a **carga relativa** de uma aresta é definida pela proporção de sua capacidade que é utilizada para rotear as *commodities*. A carga relativa de e é dada por:

$$RLd(e) = \frac{Ld(e)}{cap(e)}.$$

Note que a carga relativa da solução de um CMCF é sempre menor ou igual a 1, por se tratar de um fluxo viável.

O **congestionamento** da rede produzido por um multifluxo F é dado pela maior carga relativa. Assim, em uma rede G com capacidades nas arestas, o congestionamento é dado por:

$$congestionamento(G) = \max_{e \in E}\{RLd(e)\}.$$

O **problema da utilização mínima da capacidade** (***minimum capacity utilization problem*** ou **MCUP**) [12], cuja meta é minimizar o congestionamento, também pode ser modelado por um programa linear:

$$\text{Minimizar } u \text{ sujeito a: } \begin{cases} \sum_{u \in N(v)} f_k((u,v)) = \sum_{w \in N(v)} f_k((v,w)) & \forall v \in V \setminus \{s_k, t_k\} \\ \sum_{v \in N(s_k)} f_k((s_k,v)) = d_k & \forall k \in \mathcal{K} \\ Ld(e) \leq u\, cap(e) & \forall e \in E \end{cases} \tag{2.2}$$

Note que a solução de um MCUP não é, necessariamente, um fluxo viável. O objetivo do problema consiste em minimizar o valor u, congestionamento do problema, porém não há nenhuma restrição matemática que impeça u de se tornar maior do que 1.

Os problemas do CMCF e MCUP são de certa forma equivalentes pois suas soluções estão relacionadas da seguinte forma [12]:

Lema 2.2. *Seja $\mathcal{R}(G,\mathcal{K})$ uma instância do MCUP. A fração de vazão ótima z^* do CMCF para esta mesma instância é dada por*

$$z^* = \frac{1}{u^*},$$

onde u^* é o congestionamento ótimo do MCUP.

Demonstração. Seja $\widehat{F} = \{\widehat{f}_k \mid k \in \mathcal{K}\}$ uma solução do MCUP e u o congestionamento produzido por ela. Podemos definir $F = \{f_k \mid k \in \mathcal{K}\}$, uma solução viável do CMCF, fazendo:

$$f_k(e) = \frac{\widehat{f}_k(e)}{u} \qquad \forall e \in E \quad \forall k \in \mathcal{K}. \tag{2.3}$$

Para provar que F é uma solução viável do CMCF basta observar que a construção de F se baseia no fato de que, dividindo a quantidade de fluxo que passa em cada aresta de $\widehat{F}$ por u, constrói-se um fluxo que possui carga relativa menor ou igual a 1 em todos os caminhos.

Podemos expressar z, fração de vazão da solução do CMCF, em função de u. De fato, utilizando a equação (2.3) e comparando os programas lineares em (2.1) e (2.2), temos:

$$\begin{aligned} \frac{\sum_{v \in N(s_k)} \widehat{f}_k((s_k, v))}{u} &= z\, d_k & \forall k \in \mathcal{K} \\ z &= \frac{\sum_{v \in N(s_k)} \widehat{f}_k((s_k, v))}{u\, d_k} & \forall k \in \mathcal{K} \\ z &= \frac{1}{u}. \end{aligned}$$

A primeira igualdade é conseqüência direta de (2.3) substituída em (2.1), e a terceira vem da primeira equação de (2.2) para $\widehat{f}$.

Logo, se $\widehat{F}$ for uma solução ótima do MCUP e produzir um congestionamento u^*, a fração de vazão produzida por F, dada por:

$$z^* = \frac{1}{u^*},$$

também será ótima.

Por outro lado, tendo uma solução F do CMCF, , podemos analogamente transformá-la em uma solução $\widehat{F}$ do MCUP: ao dividir por z o fluxo que passa em cada aresta, vale que

$$\frac{\sum_{v \in N(s_k)} f_k((s_k, v))}{z} = d_k \qquad \forall k \in \mathcal{K},$$

isto é, o valor do fluxo de uma *commodity* k é igual à sua demanda. Por outro lado, dividindo o fluxo f que passa numa aresta e no CMCF, construímos um fluxo para o qual vale para qualquer aresta e:

$$Ld(e) = \frac{\frac{\sum_{k \in \mathcal{K}} f_k(e)}{cap(e)}}{z} = \frac{1}{z} \frac{\sum_{k \in \mathcal{K}} f_k(e)}{cap(e)}.$$

Usando $\frac{1}{z} = u$ e observando, como dito anteriormente, que este novo fluxo consegue rotear toda a demanda das *commodities*, percebemos que $\widehat{F}$ construída deste modo é uma solução para o problema MCUP. □

Pelo lema 2.2, o problema de maximizar a fração de vazão traz consigo outra conseqüência: o congestionamento é minimizado. Este fato será usado nos algoritmos desinformados dos capítulos seguintes: ao procurar um roteamento que maximize a fração de vazão, o fluxo encontrado automaticamente minimizará o congestionamento na rede.

2.1.1 Definições para redes com capacidades em vértices

As definições de carga absoluta, carga relativa e congestionamento para redes com capacidades nos vértices também serão usadas neste texto. Elas são análogas para o caso de redes com capacidades nas arestas e estão definidas a seguir.

A **carga absoluta** de um vértice v é dada por:

$$Ld(v) = \sum_{k \in \mathcal{K}} \sum_{u \in N(v)} f_k((u,v)) + \sum_{\substack{k \in \mathcal{K} \\ s_k = v}} \sum_{w \in N(v)} f_k((v,w)),$$

onde o primeiro termo representa a quantidade de fluxo que chega ao vértice v por arestas incidentes e o segundo termo representa a quantidade de fluxo originada em $s_k = v$, isto é, o fluxo das *commodities* das quais v é o vértice de origem. Portanto, o fluxo de uma *commodity* irá contribuir para a carga relativa de um vértice v toda vez que v for vértice intermediário em algum caminho pelo qual está sendo roteado uma porção do fluxo, ou se v é origem ou destino da *commodity* em questão.

A **carga relativa** de um vértice v é dada por:

$$RLd(v) = \frac{Ld(v)}{cap(v)}.$$

O **congestionamento** em uma rede G com capacidades nos vértices é dado por:

$$congestionamento(G) = \max_{v \in V}\{RLd(v)\}.$$

2.2 Roteamento desinformado

O problema do roteamento desinformado é uma variante do problema de roteamento que se caracteriza, essencialmente, por não utilizar informações sobre o

conjunto de *commodities* para escolher as rotas entre cada par de vértices. Para ilustrar um impedimento que surge neste contexto, torna-se proibido, por exemplo, evitar um caminho para rotear uma *commodity* k por já saber de antemão que o rotamento de uma *commodity* k' também o utiliza. Proibir o uso deste tipo de informação implica que cada *commodity* é roteada de forma independente das demais e que o caminho escolhido para roteá-la depende apenas de sua origem e destino.

Outra característica do problema do roteamento desinformado que trataremos aqui (minimizando o congestionamento) é que a solução não necessariamente respeitará a capacidade das arestas ou vértices da rede. Isto quer dizer que, em geral, as cargas relativas das arestas (ou vértices) e o congestionamento dados pelo algoritmo desinformado se tornarão maiores que 1.

Como o algoritmo não conhece as *commodities*, a solução de um problema de roteamento desinformado consiste em estabelecer um conjunto de possíveis rotas entre cada par de vértices. Para cada par $u, v \in V$ existirá uma distribuição de probabilidade para caminhos que ligam u a v, que é posteriormente utilizada para definir quais caminhos serão usados para o roteamento da demanda de uma *commodity* com origem em u e destino em v. Chamamos este conjunto de n^2 distribuições de probabilidade (uma para cada par de vértices) de esquema de roteamento desinformado.

Após construído o esquema, a escolha de rotas para atender um pedido de roteamento entre um par $\langle u, v \rangle$ será baseada nele. Há várias formas diferentes de usar o esquema de roteamento para esta escolha, descritas a seguir:

- Roteamento **determinístico**: Ao primeiro pedido de roteamento entre um par $\langle u, v \rangle$, este tipo de roteamento escolhe uma única rota para o envio de toda a demanda. Além disso, todos os demais pedidos de roteamento entre este mesmo par também usarão este mesmo caminho, ou seja, os caminhos são rotas fixas entre cada par de vértices.

- Roteamento **randomizado**: Esta variante de roteamento possui duas vertentes:

  - Fracionário: A demanda da *commodity* pode ser dividida entre caminhos, isto é, para um mesmo pedido de roteamento podem ser usadas várias rotas, cada uma levando uma porção da demanda. Neste caso, a distri-

buição de probabilidade será usada para indicar qual a fração de cada *commodity* cada um dos caminhos irá carregar.

- Inteiro: É preciso escolher um único caminho para atender a um pedido de roteamento. A probabilidade de um caminho ser escolhido para atender a um pedido é igual à sua probabilidade no esquema de roteamento. No entanto, a diferença para o caso determinístico é que a cada pedido de roteamento pode ser usado um caminho diferente.

O trabalho de Raghavan e Thompson [13] demonstra que os métodos fracionário e inteiro são de certa forma equivalentes, já que o congestionamento obtido no método inteiro tende com alta probabilidade ao congestionamento obtido no método frácionário. Os métodos tratados neste trabalho pressupõem a aplicação do roteamento randomizado, visto que para o roteamento inteiro as razões de competitividade existem limites inferiores maiores do que os resultados apresentados aqui. Além disso, os algoritmos apresentados nos próximos contemplarão somente a construção do esquema de roteamento, sabendo que sua posterior utilização para escolher as rotas torna-se trivial.

Para definir esta distribuição de probabilidade para caminhos que ligam dois vértices u, v, o algoritmo desinformado estabelece um roteamento supondo uma demanda da *commodity*. Esta demanda é escolhida por cada algoritmo de modos diferentes, geralmente remetendo a algum limite superior para o valor do fluxo roteado entre este par de vértices em um algoritmo que obtém o congestionamento ótimo. Posteriormente, o valor do fluxo usado para rotear esta demanda é normalizado para que a soma do valor do fluxo definido em todos os caminhos entre u e v seja igual a 1, sendo, portanto, uma distribuição de probabilidade para os caminhos. Note que estas demandas não dependem das *commodities* da entrada do problema, mantendo assim a escolha de rotas independente do conjunto $\mathcal{K}$.

Para rotear estas demandas entre cada par de vértices estabelecidas para o esquema de roteamento, será usado um problema CMCF com estas demandas entre cada par de vértices, resultando em um conjunto de fluxos $F = \{f_i \mid i \in \langle u, v \rangle\}$. O fluxo solução deste problema, como visto anteriormente, minimiza o congestionamento, caso toda a demanda seja mandada através dele (o que é equivalente a usar o problema MCUP). Para transformar este conjunto de fluxos em um esquema

de roteamento, isto é, transformar os fluxos em um conjunto de n^2 distribuições de probabilidade, segue um algoritmo para transformar um fluxo f_i entre um par de vértices u, v em uma distribuição de probabilidade para os caminhos que ligam estes dois vértices:seja p um caminho que liga u a v; a probabilidade que p possuirá na distribuição de probabilidade será igual a $f_{\langle u,v\rangle}(p) = \min_{e\in p} f_i(e)$, ou seja, igual ao menor valor do fluxo entre todas as arestas do caminho. Subtraindo $f_{\langle u,v\rangle}(p)$ do valor do fluxo de cada aresta do caminho p (retirando da rede arestas cujo valor do fluxo se torne igual a zero) e repetindo o processo enquanto o valor de f_i seja maior que zero, é produzido um conjunto de caminhos $\mathcal{P}_{\langle u,v\rangle}$ que ligam u a v e possuem as seguintes propriedades:

$$\begin{aligned} 0 \leq f_{\langle u,v\rangle}(p) \leq cap(e) \quad & \forall e \in p \\ \sum_{p\in\mathcal{P}_{\langle u,v\rangle}} f_{\langle u,v\rangle}(p) = 1 \quad & \forall \langle u,v\rangle \in V^2 \end{aligned}$$

Como o algoritmo termina em tempo polinomial (no máximo m iterações), $\mathcal{P}_{\langle u,v\rangle}$ possui um número polinomial de caminhos.

Além de estabelecer um roteamento entre os vértices do grafo, um algoritmo de roteamento desinformado tem também como objetivo a minimização ou maximização de alguma função dada. Este trabalho é dedicado a algoritmos desinformados que minimizam o congestionamento.

A medição da qualidade destes algoritmos desinformados é feita por uma **análise competitiva**, isto é, pela comparação entre o valor do congestionamento produzido pelo algoritmo desinformado e o congestionamento ótimo para a mesma instância de entrada. Como estaremos interessados no pior caso para o algoritmo desinformado, vamos medir a qualidade do algoritmo pela entrada na qual a razão entre o valor do congestionamento do algoritmo desinformado e o ótimo é a maior possível. Chamamos esta razão de **razão de competitividade** CR, definida por:

$$CR = \max_{\mathcal{R}(G,\mathcal{K})} \left\{ \frac{OBL(\mathcal{R}(G,\mathcal{K}))}{OPT(\mathcal{R}(G,\mathcal{K}))} \right\}$$

Onde $OBL(\mathcal{R}(G,\mathcal{K}))$ e $OPT(\mathcal{R}(G,\mathcal{K}))$ significam, respectivamente, o congestionamento produzido por um algoritmo desinformado e o congestionamento produzido por um algoritmo ótimo informado (problema MCUP), ambos recebendo como entrada a tupla $\mathcal{R}(G,\mathcal{K})$.

2.3 Histórico

Os primeiros trabalhos sobre roteamento desinformado tiveram foco em redes não-direcionadas com topologias específicas: os trabalhos de Valiant e Brebner (1981), Borodin e Hopcroft (1982), Kaklamanis, Krizanc e Tsantilas (1990) e Vöcking (2001) abordam limites superiores e inferiores para a razão de competitividade de roteamentos desinformados no hipercubo (cf. [14]).

Räcke, em 2002, produziu o primeiro trabalho que aborda o problema do rotamento desinformado para redes sem topologia pré-definida, estabelecendo uma razão de competitividade polilogarítmica de $O(\log^3 n)$ para o problema em redes não-direcionadas com capacidades nas arestas [9]. Este algoritmo usa uma árvore de decomposição da rede, fazendo com que o roteamento na rede simule, de forma aproximada, um roteamento na árvore de decomposição.

Embora este algoritmo tenha causado surpresa pela generalidade do resultado com razão de competitividade polilogarítmica, o método possui um grande problema, pois tem como subrotina encontrar o corte mais esparso (corte que possui menor razão entre capacidade das arestas e demanda entre pares de vértices desconectados por ele), a complexidade de tempo do algoritmo não é polinomial.

No ano de 2003, em trabalhos de Bienkowski, Korzeniowski e Räcke [14] e Harrelson, Hildrum e Rao [15] (publicados de forma independente na mesma conferência) foram obtidos algoritmos de roteamento desinformado de tempo polinomial para redes não-direcionadas sem topologia pré-definida que possuam capacidades nas arestas. No primeiro, foi proposto um algoritmo muito semelhante ao de Räcke [9], porém mais simples, de tempo polinomial e com uma razão de competitividade um pouco maior – $O(\log^4 n)$ – do que o anterior. Já no segundo, foi apresentado um algoritmo de tempo polinomial com razão de competitividade $O(\log^2 n \log\log n)$. Estes trabalhos estão descritos em detalhes nos capítulos 6 e 7, respectivamente.

Além de produzirem roteamentos desinformados de tempo polinomial, estes dois trabalhos baseiam seus algoritmos na construção eficiente de uma decomposição hierárquica da rede. No capítulo 5 é feito um estudo detalhado sobre como a árvore de decomposição associada à decomposição hierárquica de uma rede é usada por estes algoritmos para obter um esquema de roteamento. Este tipo de decomposição é usada para obter a solução de alguns problemas importantes de computação dis-

tribuída [9, 16], sendo também utilizada como um precondicionador para resolver sistemas lineares esparsos [17].

Além destes trabalhos, Räcke [18], em 2008, mostrou um novo algoritmo para construir uma árvore de decomposição, baseado no método de Fakcharoenphol, Rao e Talwar [19]. Dentre outros problemas cujas soluções podem ser obtidas usando variações da árvore de decomposição construída pelo método deste trabalho, o roteamento desinformado com objetivo de minimizar do congestionamento em redes não-direcionadas com capacidades nas arestas alcança razão de competitividade $O(\log n)$.

Em [20], Gupta, Hajiaghayi e Räcke (2006) desenvolveram um *framework* para modelagem de problemas de roteamento desinformado, conseguindo assim generalizar o resultado obtido por Harrelson *et al.* em [15] para uma classe maior de funções, dentre outros resultados. O trabalho demonstra que quando a carga relativa das arestas é calculada por uma função que é norma sobre os fluxos que por ela passam, existe um algoritmo desinformado que visa minimizar o congestionamento na rede e possui razão de competitividade $O(\log^2 n \log\log n)$. A função C de carga é uma norma se monotônica, sub-aditiva e se $C(a\,F) = aC(F)$, onde a é uma constante e F é o conjunto de fluxos. Como exemplo, $\sum_{k\in K} f_k(e)$ é um caso particular de norma sobre os fluxos que passam na aresta e.

Em 2003, Azar, Cohen, Fiat, Kaplan e Räcke [21] demonstram que é possível encontrar um **roteamento desinformado ótimo** — isto é, aquele que encontra a melhor solução possível para um algoritmo desinformado — através da resolução de um programa linear que obtém uma família de fluxos que minimiza o congestionamento para todos os possíveis valores de demanda. Este resultado é válido para redes direcionadas ou não, tanto com capacidades nos vértices quanto nas arestas.

O programa linear utilizado é exatamente como esta descrição sugere: ele possui um número polinomial de variáveis, mas um número infinito de restrições que correspondem às diferentes demandas. Embora o número de restrições seja infinito, os autores mostram que ainda assim e possível resolver o programa linear em tempo polinomial, utilizando-se o método do elipsóide, com um oráculo separador adequado. O oráculo separador é implementado através da resolução de um conjunto de programas lineares, capazes de verificar que a solução é de fato viável ou de encontrar uma restrição não satisfeita do programa linear original.

O trabalho de Hajiaghayi, Kleinberg, Leighton e Räcke [22], publicado em 2005, é voltado especificamente para redes não-direcionadas com capacidades nos vértices e redes direcionadas, apresentando limites inferiores e superiores para a razão de competitividade em cada um destes casos. Estas duas variações de entrada são mais difíceis que a versão não-orientada com capacidades nas arestas, pois apresentam limites inferiores maiores, fator de aproximação de corte mais esparso maior, entre outros fatores. Os algoritmos apresentados em [22] obtêm razões de competitividade de $O(\sqrt{|\mathcal{K}|}\log n)$, no caso de redes com capacidades nos vértices, e $O(\sqrt{|\mathcal{K}|}n^{1/4}\log n)$ para redes direcionadas, onde $|\mathcal{K}|$ indica o número de *commodities* do problema. Este trabalho está descrito em detalhes no capítulo 4.

2.4 Corte mais esparso

A relação entre a fração de vazão ótima e a capacidade dos cortes da rede desempenhará um papel importante na análise e elaboração dos algoritmos desinformados abordados neste trabalho, pois fornece um limite superior para a quantidade de fluxo que pode ser roteado na rede.

No problema do fluxo máximo em redes, é um resultado amplamente conhecido que o valor do fluxo máximo é igual à capacidade de um corte mínimo que separa a fonte do sumidouro [2]. Além disso, um tal corte e seu valor podem ser facilmente determinados em tempo polinomial.

Para o problema do multifluxo será interessante quantificar a relação entre a capacidade de um corte e as demandas das *commodities* que precisam utilizá-lo. Tal relação é descrita pela esparsidade do corte.

Para um corte $\mathcal{C} = (S, \overline{S})$, a **esparsidade** de $\mathcal{C}$ é definida como a razão entre a soma das capacidades das arestas do corte e a soma das demandas das *commodities* que têm suas origens e destinos separados por $\mathcal{C}$. Se a segunda soma é nula, estabelecemos que a esparsidade do corte em questão é infinita. Em redes não-direcionadas, a esparsidade é definida por:

$$esp(\mathcal{C}) = \frac{cap(A,B)}{\sum_{k \in \mathcal{K}_\mathcal{C}} d_k},$$

onde $cap(A,B) = \sum_{e \in \mathcal{C}} cap(e)$ e $\mathcal{K}_\mathcal{C}$ é o conjunto de *commodities* com origens e

destinos separados pelo corte $\mathcal{C}$.

Em redes direcionadas, para $\mathcal{C} = (A, B)$, a esparsidade é definida da seguinte forma:

$$esp(\mathcal{C}) = \frac{\sum_{e=(u,v),u\in A,v\in B} cap(e)}{\sum_{k\in\mathcal{K},s_k\in A,t_k\in B} d_k}.$$

Um corte com esparsidade mínima em uma rede é chamado de **corte mais esparso**. Ao contrário do problema do corte mínimo, o problema de encontrar um corte mais esparso é NP-difícil [23]. Sua capacidade é, naturalmente, um limite superior para o valor da fração de vazão máxima do CMCF.

Lema 2.3. *Seja z^* o valor da fração de vazão ótima do CMCF e $\mathcal{C}$ um corte mais esparso da rede G. Temos que:*

$$z^* \leq esp(\mathcal{C}).$$

Demonstração. Lembrando que z^* é a fração do quanto é roteado da demanda de cada *commodity* na solução do CMCF, temos que para uma *commodity* k será roteada uma quantidade igual a z^*d_k. Como é necessário que o fluxo das *commodities* que possuem suas origens e destinos separados por $\mathcal{C}$ passe pelas arestas deste corte, vale que:

$$\sum_{e\in\mathcal{C}} Ld(e) \geq z^* \sum_{k\in\mathcal{K}_\mathcal{C}} d_k.$$

Além disso, a solução do CMCF satisfaz, para cada $e \in E$, $Ld(e) \leq cap(e)$. Juntando esta desigualdade com a anterior, temos as desigualdades:

$$z^* \sum_{k\in\mathcal{K}_\mathcal{C}} d_k \leq \sum_{e\in\mathcal{C}} Ld(e) \leq \sum_{e\in\mathcal{C}} cap(e),$$

que fornecem

$$z^* \leq esp(\mathcal{C}),$$

como requerido. □

Além do limite trivial provado acima, o **teorema do fluxo máximo e corte mínimo** [3], enunciado abaixo, também relaciona a esparsidade de um corte mais esparso $\mathcal{C}$ com z^*, fração de vazão ótima produzida por um multifluxo concorrente máximo:

Teorema 2.4. *Seja z^* a fração de vazão ótima de um CMCF e $\mathcal{C}$ um corte de esparsidade mínima. Então existe um valor $\alpha \geq 1$ tal que:*

$$\frac{esp(\mathcal{C})}{\alpha} \leq z^* \leq esp(\mathcal{C}). \tag{2.4}$$

O valor de α depende da classe da rede e do tipo de *commodities* dados na entrada do problema e foi assunto estudado por vários autores. Segue abaixo uma lista de resultados sobre o assunto, na qual as referências bibliográficas que foram omitidas podem ser encontradas em [24].

Leighton e Rao provaram que para o caso particular em que há uma *commodity* com demanda unitária entre todos os pares de vértices (caso chamado de **multifluxo de demanda uniforme**), $\alpha = O(\log n)$. Este resultado vale tanto para redes direcionadas quanto não-direcionadas.

Para o caso geral de multifluxos em redes não-direcionadas, Klein, Agrawal, Rao e Ravi provaram que $\alpha = O(\log C \log D)$, onde C e D são, respectivamente, a soma de todas as capacidades e todas as demandas. Demais resultados para este mesmo caso são: $O(\log n \log D)$ por Tragoudas, $O(\log |\mathcal{K}| \log D)$ por Garg, Vazirani e Yannakakis, $O(\log^2 |\mathcal{K}|)$ por Plotkin e Tardos e finalmente para $O(\log |\mathcal{K}|)$ por Aumann e Rabani [25]. Especificamente, para redes planares não-direcionadas, Klein, Plotkin e Rao provam que o fator α é $O(\log D)$ – melhorado para $O(\log |\mathcal{K}|)$ em [26]; para o caso de demanda uniforme, foi provado um fator de $\alpha = O(1)$ também por Klein, Plotkin e Rao.

Para redes direcionadas, Leighton e Rao mostraram que o fator de $O(\log n)$ também se aplica para o caso de demanda uniforme, posteriormente melhorado por Seymour. Para o caso de demanda simétrica, isto é, aquela em que a demanda entre o par $\langle u, v \rangle$ é igual à demanda de $\langle v, u \rangle$, um fator de $O(\log^3 n)$ foi obtido por Even, Naor e Schieber [27] e por Klein, Plotkin e Rao [28].

Além do estudo do valor de α, os trabalhos citados produziram também algoritmos para encontrar cortes mais esparsos de maneira aproximativa, encontrando um corte cuja esparsidade seja igual a $z^*\alpha$. Usando o teorema do fluxo máximo e corte mínimo, vale que:

$$\begin{array}{ccccc} \alpha \frac{esp(\mathcal{C})}{\alpha} & \leq & \alpha z^* & \leq & \alpha\, esp(\mathcal{C}) \\ esp(\mathcal{C}) & \leq & esp(\mathcal{C}_{apx}) & \leq & \alpha\, esp(\mathcal{C}), \end{array}$$

onde $\mathcal{C}_{apx}$ é um corte produzido por estes algoritmos.

Um corte conseguido por uma aproximação para o corte mais esparso será, ao longo do texto, chamado de **corte mais esparso aproximado**. Quanto menor o α, portanto, melhor a aproximação fornecida:w . Os algoritmos que encontram estes cortes aproximados serão usados como subrotina pelos algoritmos de roteamento desinformados dos próximos capítulos para conseguir detectar possíveis "gargalos" nas rotas estabelecidas e modificá-las para melhorar a fração de vazão encontrada.

Capítulo 3

Limites inferiores

Este capítulo apresenta os limites inferiores para a razão de competitividade de algoritmos desinformados randomizados cujo objetivo seja minimizar o congestionamento.

Estes limites são válidos apenas para algoritmos desinformados randomizados, já que Borodin e Hopcroft [29] provaram que a razão de competitividade de algoritmos desinformados determinísticos para redes com capacidades nas arestas possui um limite inferior de $\Omega(\frac{\sqrt{n}}{\Delta^{3/2}})$, onde Δ é o maior grau dos nós. Posteriormente, Kaklamanis [30] prova um limite inferior ainda maior para este mesmo caso: $\Omega(\frac{\sqrt{n}}{\Delta})$.

A tabela 3.1 contém os limites inferiores conhecidos até o momento para a razão de competitividade de algoritmos desinformados randomizados.

Tipo de rede		Limite inferior
Digrafos	capacidade nos vértices	$\Omega(\sqrt{n})$
	capacidade nas arestas	$\Omega(\sqrt{n})$
Grafos	capacidade nos vértices	$\Omega(\sqrt{n})$
	capacidade nas arestas	$\Omega(\log n)$

Tabela 3.1: Limites inferiores para a razão de competitividade de algoritmos desinformados randomizados que minimizam congestionamento.

As próximas seções são dedicadas às demonstrações destes resultados.

3.1 Limites inferiores em redes direcionadas com capacidades nas arestas

O resultado desta seção foi provado por Azar *et al.* [21]. Este limite inferior é demonstrado exibindo-se uma entrada do problema na qual qualquer algoritmo desinformado randomizado possui razão de competitividade $\Omega(\sqrt{n})$.

A rede direcionada D_N usada na demonstração possui 3 níveis: o primeiro contém $\binom{N}{2}$ vértices, denotados por $a_{i,j}$, $1 \leq i < j \leq N$; o segundo nível possui N vértices, chamados de b_i, $1 \leq i \leq N$; e o terceiro nível é composto somente pelo vértice t.

Todas as arestas de D_N possuem capacidade unitária e podem ser classificadas em dois conjuntos:

Arestas que ligam o primeiro nível ao segundo: Todo vértice $a_{i,j}$ possui duas arestas incidentes: $(a_{i,j}, b_i)$ e $(a_{i,j}, b_j)$.

Arestas que ligam o segundo nível ao terceiro: Todo b_i possui uma aresta incidente: (b_i, t).

A figura 3.1 mostra a rede D_4.

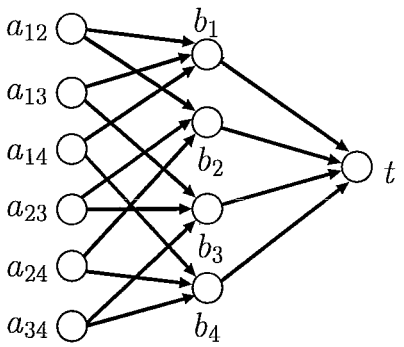


Figura 3.1: Rede D_4.

A rede D_N será usada no teorema a seguir para a demonstração do limite inferior:

Teorema 3.1. *Qualquer roteamento desinformado randomizado em D_N possui razão de competitividade $\Omega(\sqrt{n})$.*

Demonstração. A solução do roteamento desinformado consiste em determinar, para cada par de vértices, uma distribuição de probabilidade para os caminhos que os

ligam. Esta distribuição de probabilidade indica a fração da demanda da *commodity* com origem e destino nestes vértices que será roteada por cada caminho. Deste modo, a construção do esquema de roteamento pode ser vista como a escolha de caminhos para rotear uma demanda unitária entre cada par de vértices da rede.

No caso da rede D_N, podemos observar que as *commodities* entre pares $\langle a_{i,j}, b_i \rangle$ e entre pares $\langle a_{i,j}, b_j \rangle$, $1 \leq i < j \leq N$, possuem uma única aresta pela qual podem rotear sua demanda. Logo, por ter uma única opção, a carga absoluta nestas arestas será idêntica tanto no roteamento produzido por um algoritmo informado quanto no produzido por um desinformado, para qualquer instância do conjunto de *commodities*. Podemos argumentar de modo análogo para as *commodities* entre pares $\langle b_i, t \rangle$, $1 \leq i \leq N$. Logo, estas *commodities* não interessam para esta demonstração e serão ignoradas deste ponto em diante, restando apenas as *commodities* com origem nos vértices $a_{i,j}$ e destino em t.

Portanto, a construção do esquema de roteamento desinformado na rede D_N pode ser vista como a escolha de caminhos para rotear uma demanda unitária entre $\binom{N}{2}$ pares de vértices. Para cada vértice $a_{i,j}$, $p_{i,j}$ denotará a fração da *commodity* com origem neste vértice que será roteada pelo caminho $(a_{i,j}, b_i, t)$, enquanto $q_{i,j} = 1 - p_{i,j}$ denotará a porção que será roteada por $(a_{i,j}, b_j, t)$.

Nesta rede em particular, a carga absoluta (e também a carga relativa, pois todas as capacidades são unitárias) das arestas (b_i, t) será igual à quantidade de fluxo que chega até o vértice b_i. Logo:

$$Ld((b_i, t)) = \sum_{y=1}^{i-1} q_{y,i} + \sum_{z=i+1}^{N} p_{i,z}.$$

A primeira parcela representa a quantidade de fluxo que chega a b_i originada de vértices $a_{y,i}$, $y < i$, e a segunda representa a quantidade recebida por b_i de vértices $a_{i,z}$, $z > i$.

Como existem $\binom{N}{2}$ demandas unitárias que deverão ser roteadas passando pelos N vértices do segundo nível, a carga absoluta média das arestas (b_i, t) será:

$$\frac{\sum_{i=1}^{N} Ld((b_i, t))}{N} = \frac{\binom{N}{2}}{N} = \frac{(N-1)}{2}.$$

Concluímos, portanto, que existe uma aresta (b_x, t) tal que $Ld((b_x, t)) \geq \frac{(N-1)}{2}$.

Logo, para o roteamento desinformado randomizado vale que:

$$congestionamento(G) \geq \frac{Ld((b_x, t))}{cap((b_x, t))} \geq \frac{(N-1)}{2}.$$

Mostraremos agora um exemplo de conjunto $\mathcal{K}$ de *commodities* (isto é, uma possível entrada $\mathcal{R}(D_N, \mathcal{K})$) no qual qualquer roteamento desinformado randomizado obtém um congestionamento $\Omega(\sqrt{n})$ maior que o congestionamento ótimo obtido pelo roteamento informado para a mesma instância.

Seja b_x um vértice do segundo nível tal que $Ld((b_x, t)) \geq \frac{(N-1)}{2}$ no esquema de roteamento desinformado. Observe que, neste esquema, este vértice b_x recebe fluxo somente dos seguintes $N-1$ vértices: $a_{j,x}$, $1 \leq j < x-1$ e $a_{x,j}$, $x < j \leq N$. Na instância considerada, o conjunto $\mathcal{K}$ é composto de $N-1$ demandas unitárias que possuem origem exatamente nestes vértices e destino em t, ou seja, há *commodities* entre pares de vértices $\langle a_{j,x}, t\rangle$, $1 \leq j < x-1$, e entre pares $\langle a_{x,j}, t\rangle$, $x < j \leq N$.

Note que b_x, nesta instância do conjunto de *commodities*, continua recebendo fluxo dos mesmos $N-1$ vértices dos quais ele recebia no esquema de roteamento desinformado e, portanto, recebe a mesma carga que recebia naquele esquema. Logo, no roteamento desinformado das *commodities* desta instância, temos também $Ld((b_x, t)) \geq \frac{(N-1)}{2}$.

Portanto, usando o roteamento desinformado randomizado para rotear este conjunto de demandas, temos:

$$congestionamento(G) \geq \frac{(N-1)}{2}.$$

No entanto, um algoritmo informado pode rotear estas *commodities* da seguinte forma:

- As demandas com origem em $a_{j,x}$ serão roteadas pelo caminho $(a_{j,x}, b_j, t)$, para todo $1 \leq j \leq x-1$;
- As demandas com origem em $a_{x,j}$ serão roteadas pelo caminho $(a_{x,j}, b_j, t)$, para todo $x+1 \leq j \leq N$.

O algoritmo informado produz congestionamento igual a 1, pois cada aresta (b_i, t) está sendo utilizada para rotear a demanda de uma única *commodity*.

Calculando a razão de competitividade, chegamos a:

$$CR \geq \frac{\frac{(N-1)}{2}}{1}.$$

Lembrando que $n = \binom{N}{2} + N + 1$, concluímos que $N = \Omega(\sqrt{n})$. □

3.2 Limites inferiores em redes direcionadas com capacidades nos vértices

O problema do roteamento desinformado em uma rede direcionada com capacidades nos vértices pode ser reduzido a um problema equivalente em uma rede direcionada com capacidades nas arestas.

Pode-se transformar a rede de entrada $D(V, E)$ em uma rede $D'(V', E')$ e transformar o conjunto de *commodities* $\mathcal{K}$ em um conjunto $\mathcal{K}'$ da seguinte forma:

V': Cada vértice $v \in V$ será transformado em dois vértices em V': v' e v''.

E': Haverá, em D', uma aresta entre cada par (v', v'') de capacidade igual à capacidade de v em D.

Cada aresta $(u, v) \in E$ será transformada em uma aresta $(u'', v') \in E'$ possuindo capacidade infinita.

$\mathcal{K}'$: Cada *commodity* $k \in \mathcal{K}$ que possuir origem em um vértice s, destino em um vértice t ($s, t \in V$) e demanda d_k será transformada em uma *commodity* $k' \in \mathcal{K}'$ com a mesma demanda, origem em s' e destino em t''.

Na figura 3.2 é dado um exemplo desta transformação para uma rede D.

Uma solução para o problema em $\mathcal{R}'(D', \mathcal{K}')$ pode ser facilmente transformada em uma solução para o problema $\mathcal{R}(D, \mathcal{K})$: sendo $F' = \{f'_k \mid k \in \mathcal{K}'\}$ a solução do primeiro problema e $F = \{f_k \mid k \in \mathcal{K}\}$ a solução do segundo, temos que:

- Cada vez que um fluxo $f'_k \in F'$ passa por uma aresta (v', v''), o fluxo $f_k \in F$ passará pelo vértice v;
- Cada vez que um fluxo $f'_k \in F'$ passa por uma aresta (u'', v'), o fluxo $f_k \in F$ passará pela aresta (u, v).

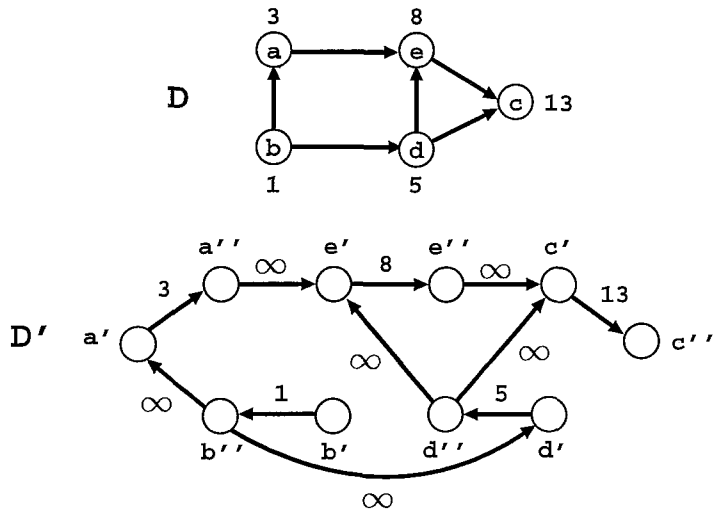


Figura 3.2: Transformação de uma rede D em D'. Cada vértice v pertencente a D é transformado em um par de vértices ligados por uma aresta de capacidade $cap(v)$ em D'. As arestas de D se transformam em arestas de capacidade infinita em D'.

Usando esta correspondência entre as soluções, a carga relativa de cada aresta (v', v'') terá o mesmo valor que a carga relativa que o vértice v (seu correspondente em D) e, com isto, o congestionamento obtido nos dois problemas será igual.

Como é possível transformar o problema de minimizar o congestionamento em vértices em uma rede direcionada para o problema de minimizar o congestionamento em arestas, o limite inferior para este problema também é $\Omega(\sqrt{n})$.

3.3 Limites inferiores em redes não-direcionadas com capacidades nos vértices

Hahuaghayi *et al.* [22] mostraram um limite inferior para redes não-direcionadas com capacidades nos vértices de forma muito similar à demonstração feita para redes direcionadas com capacidades nas arestas.

Na demonstração do resultado será usado uma rede G_N, uma versão não-direcionada da rede D_N usada na seção 3.1, possuindo agora capacidades unitárias nos vértices. Todos os vértices terão capacidade igual a 1, exceto t, que terá capacidade infinita.

A figura 3.3 mostra um exemplo da rede G_4.

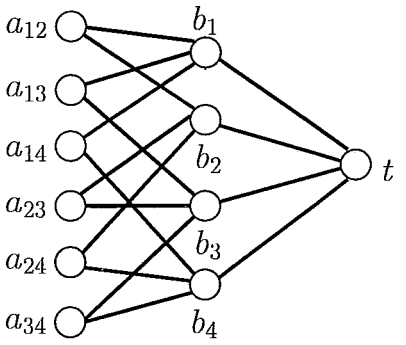


Figura 3.3: Grafo G_4.

Teorema 3.2. *Qualquer algoritmo de roteamento desinformado randomizado na rede G_N possui razão de competitividade $\Omega(\sqrt{n})$.*

Demonstração. Como na demonstração do teorema 3.1, analisaremos a carga absoluta de um vértice b_x pertencente ao segundo nível de G. Mas, neste caso, será preciso um pouco mais de cuidado: as arestas de G não são direcionadas, o que implica que não há como garantir que a carga absoluta de um vértice b_x vem apenas das *commodities* originadas dos vértices $a_{y,x}$, $1 \leq y < x$ e $a_{z,x}$, $x < z \leq N$.

Por isto, a análise do congestionamento será baseada em um limite inferior para o seu valor, dado pelo primeiro uso dos vértices do segundo nível ao rotear as *commodities* com origem em $a_{i,j}$ e destino em t. Os caminhos que possuem como origem um vértice $a_{i,j}$ serão divididos em dois conjuntos disjuntos: $(a_{i,j}, b_i, \ldots, t)$ e $(a_{i,j}, b_j, \ldots, t)$, mas a análise do congestionamento será restrita somente à passagem de fluxo entre os dois primeiros nós destes caminhos.

Novamente, para todo vértice $a_{i,j}$ definimos frações $p_{i,j}$ e $q_{i,j}$ tais que:

$$p_{i,j} + q_{i,j} = 1,$$

onde $p_{i,j}$ é a quantidade da *commodity* com demanda unitária com origem em $a_{i,j}$ que é roteada por um caminho cujo segundo vértice é b_i e $q_{i,j}$ é a quantidade roteada por caminhos que utilizam b_j como segundo passo.

Sendo $Ld_1(b_i)$ a carga relativa de um vértice b_i somente na primeira passagem

de fluxo, vale a seguinte igualdade:

$$\begin{aligned}\sum_{i=1}^{N} Ld_1(b_i) &= \sum_{i=1}^{N}\left(\sum_{y=1}^{i-1} q_{y,i} + \sum_{z=i+1}^{N} p_{i,z}\right) \\ &= \sum_{i<j}(p_{i,j} + q_{i,j}) \\ &= \binom{N}{2}\cdot 1 = \frac{N\cdot(N-1)}{2}.\end{aligned}$$

Calculando a média dos $Ld_1(b_i)$, $1 \leq i \leq N$, chegamos a:

$$\frac{\frac{N\cdot(N-1)}{2}}{N} = \frac{(N-1)}{2}.$$

É possível afirmar:

- $RLd(b_i) = \frac{Ld(b_i)}{cap(b_i)} \geq Ld_1(b_i)$, para todo $1 \leq i \leq N$;
- Existe um vértice b_x pertencente ao segundo nível de G tal que $Ld_1(b_x) \geq \frac{(N-1)}{2}$.

Mostraremos um conjunto de *commodities* – uma entrada para o problema na rede D_n – na qual o roteamento desinformado randomizado possui um congestionamento pelo menos $\sqrt{n}$ maior do que o congestionamento dado por um algoritmo informado. O conjunto de *commodities* será o mesmo do exemplo da seção 3.1: um conjunto de $\binom{N}{2}$ *commodities*, cada uma com origem em um vértice do primeiro nível de G, destino em t e demanda unitária.

Um roteamento informado poderá rotear desta maneira: sendo b_x um vértice tal que $Ld_1(b_x) \geq \frac{(N-1)}{2}$, as *commodities* com origem em vértices $a_{y,x}$, $1 \leq y < x$ serão roteadas pelos caminhos $(a_{y,x}, b_y, t)$ e as *commodities* com origem em $a_{x,z}$, $x < z \leq N$ usarão os caminhos $(a_{x,z}, b_z, \ldots, t)$. Este algoritmo produzirá congestionamento igual a 1 para esta entrada, pois cada vértice do segundo nível – que possui capacidade unitária – está roteando uma única *commodity* de demanda igual a um.

No roteamento desinformado, a carga relativa do vértice b_x é de pelo menos $Ld_1(b_x)$, ou seja, $\frac{(N-1)}{2}$. Calculando a razão de competitividade:

$$CR \geq \frac{Ld_1(b_x)}{1} = \frac{\frac{(N-1)}{2}}{1} = \frac{(N-1)}{2}.$$

Como $n = \binom{N}{2} + N + 1$, $N = \Omega(\sqrt{n})$. □

3.4 Limites inferiores em redes não-direcionadas com capacidades nas arestas

O fator de $\Omega(\log n)$ abordado nesta seção foi provado por Maggs *et al.* [16] e, de forma independente, por Bartal e Leonardi [31], estabelecendo um limite inferior para roteamentos desinformados randomizados em redes não-direcionadas com capacidades nas arestas.

O grafo que compõe a rede usada nesta demonstração é uma **malha** quadrada de dimensão 2 e lado m. A figura 3.4 é um exemplo de malha quadrada 8×8.

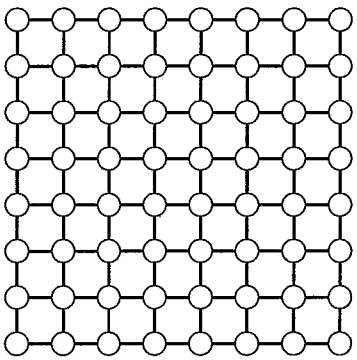


Figura 3.4: Malha 8×8

Teorema 3.3. *Qualquer algoritmo de roteamento desinformado randomizado em uma malha M de dimensão 2 possui razão de competitividade $\Omega(\log n)$.*

Demonstração. A rede usada para esta demonstração é uma malha quadrada $m \times m$, com $m = 2^x$, $x \geq 2$, que possui capacidades unitárias nas arestas.

Primeiramente definiremos a entrada do problema, para em seguida mostrar o congestionamento que um algoritmo informado e que um desinformado produzem ao roteá-la.

A entrada $\mathcal{R}(M, \mathcal{K})$ será dividida em $L = \log m - 1$ subproblemas definidos em submalhas de M. Cada subproblema $\mathcal{R}_i(M_i, \mathcal{K}_i)$, $1 \leq i \leq L$, terá sua entrada definida por:

M_i: Para o primeiro subproblema ($i = 1$) a rede a ser utilizada é a próprio malha de entrada do problema $\mathcal{R}$: $M_1 = M$.

Nos demais subproblemas, a submalha M_i na qual o problema $\mathcal{R}_i$ está definido é um dos quadrantes de M_{i-1}, submalha do problema anterior.

A figura 3.5 mostra uma malha M_i e seus 4 quadrantes, indicando as 4 possibilidades para a malha seguinte M_{i+1}.

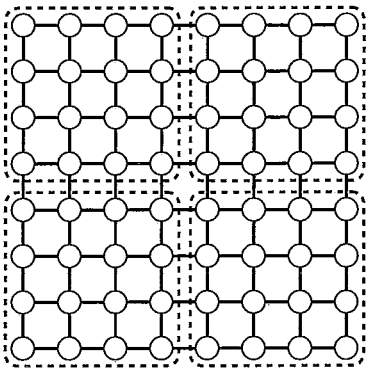


Figura 3.5: Malha M_i e seus quadrantes. A malha M_{i+1} é uma das submalhas circulada pela linha tracejada.

$\mathcal{K}_i$: O conjunto $\mathcal{K}_i$ possui $\frac{m_i}{2}$ *commodities* com demanda d. Cada uma delas possui seus vértices de origem e destino pertencentes à submalha M_i: numerando de 1 a m_i as linhas de M_i, uma *commodity* $k_j \in \mathcal{K}_i$, $1 \leq j \leq \frac{m_i}{2}$, possui origem no j-ésimo vértice da $m_i/2$-ésima linha e destino no $(j + \frac{m_i}{2})$-ésimo elemento da mesma.

A figura 3.6 mostra um conjunto de *commodities* $\mathcal{K}$ dentro de uma malha quadrada de dimensão 16.

É interessante ressaltar que a malha M_i de cada subproblema $\mathcal{R}_i$ é definida pela localização dos vértices de origem e destino das *commodities* pertencentes ao conjunto $\mathcal{K}_i$ e que, por esta característica, um algoritmo informado saberia de antemão qual é a seqüência de malhas M_i, enquanto um algoritmo desinformado não teria acesso a esta informação.

Definida a entrada do problema, vamos agora descrever como um algoritmo informado poderia rotear estas demandas. Este algoritmo sabe a priori todos os elementos do conjunto $\mathcal{K}$ e pode utilizar esta informação para escolher o melhor roteamento possível para as *commodities* especificadas. Para produzir o esquema de roteamento

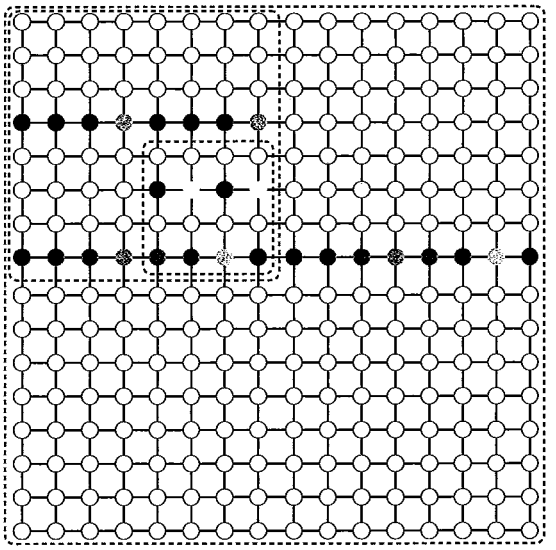


Figura 3.6: Malha de dimensão 16 e a representação de um conjunto de *commodities* $\mathcal{K}$. As submalhas M_i encontram-se demarcadas por linhas tracejadas e os vértices de origem e destino de uma mesma *commodity* possuem mesma cor.

para o problema $\mathcal{R}_i(M_i, \mathcal{K}_i)$, o algoritmo informado pode evitar usar arestas que estejam fora desta submalha, visto que todas as *commodities* de $\mathcal{K}_i$ possuem origem e destino dentro de M_i. Além disso, como o algoritmo também conhece as *commodities* do subproblema $\mathcal{R}_{i+1}$, ele também pode evitar usar arestas que pertencerão à submalha M_{i+1}.

Um esquema de roteamento ótimo para cada subproblema $\mathcal{R}_i$ pode ser fundamentado, portanto, em dois pontos básicos:

- Utilizar somente arestas que pertençam à submalha M_i;

- Não utilizar arestas que pertencerão à submalha M_{i+1}.

Um exemplo de roteamento com estas características para uma rede M de dimensão 8 está demonstrado no exemplo 3.4:

Exemplo 3.4. *A figura 3.7 dá um exemplo de como um algoritmo ótimo pode proceder dada uma submalha M com dimensão igual a* 8.

A malha M_1 tem dimensão 8 *e é demarcada pela linha tracejada mais externa. Nela existem* 4 commodities *para serem roteadas, cada uma delas está representada na figura 3.7 por pares de vértices de mesma cor. A malha M_2 é o quadrante inferior direito de M_1 e possui* 2 commodities.

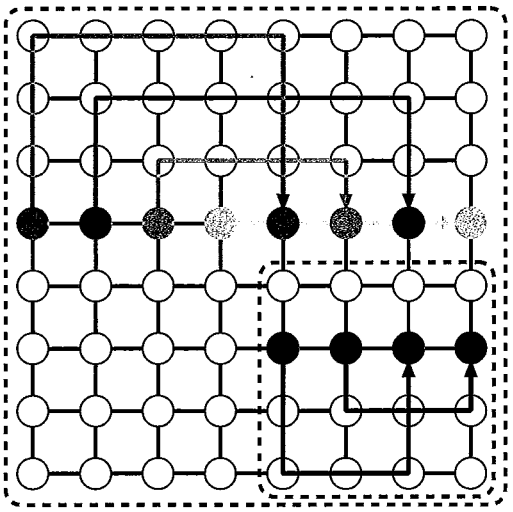


Figura 3.7: Um esquema de roteamento ótimo para um problema $\mathcal{R}(M, \mathcal{K})$. As *commodities* estão marcadas por pares de vértices de mesma cor, e as malhas dos problemas estão circuladas por linhas tracejadas. O caminho escolhido para rotear cada *commodity* está indicado por setas.

O roteamento das commodities *do subproblema $\mathcal{R}_1$ não ultrapassa as fronteiras de M_1 e, ao mesmo tempo, não utiliza arestas de M_2. Caso M_2 fosse a malha superior esquerda ou direita de M_1, o roteamento do subproblema $\mathcal{R}_1$ seria feito de forma análoga ao mostrado, porém utilizando as arestas dos quadrantes inferiores de M_1.*

Podemos estender a forma de escolha de rotas feita no exemplo 3.7 para qualquer dimensão de malha. Neste caso o congestionamento produzido por um algoritmo deste tipo é igual a d, pois os conjuntos de arestas utilizadas para rotear as *commodities* de cada $\mathcal{K}_i$ são disjuntos e dentro de cada M_i as arestas nunca são usadas para rotear mais de uma *commodity*.

Um algoritmo desinformado, ao contrário do que um informado faz, não poderia escolher seu roteamento em M_i baseado nas *commodities* que pertencem a $\mathcal{K}_{i+1}$. Calcularemos, então, o valor esperado do congestionamento que qualquer algoritmo desinformado randomizado produzirá ao rotear o conjunto $\mathcal{K}$.

Para facilitar esta análise, as arestas da malha M_i serão classificadas em dois conjuntos: arestas pares e ímpares. Na malha M_i existem $m_i - 1$ linhas de arestas horizontais e $m_i - 1$ colunas de arestas verticais, as quais serão atribuídas índices de 1 a $m_i - 1$; as arestas que estão nas linhas ou colunas de índice ímpar serão

arestas ímpares e as demais, pares.

Estaremos interessados em calcular o valor esperado da carga relativa do conjunto de arestas ímpares. Para isto, vamos inicialmente analisar a carga produzida pelo roteamento das *commodities* de $\mathcal{K}_i$ nas arestas ímpares da malha M_i.

Observe que qualquer caminho escolhido para rotear uma *commodity* k em um problema $\mathcal{R}_i(M_i, \mathcal{K}_i)$ utiliza pelo menos $\frac{m_i}{4}$ arestas ímpares. Isto acontece porque os vértices de origem e destino de cada *commodity* são separados por pelo menos $\frac{m_i}{4}$ colunas de arestas ímpares, como no exemplo da figura 3.8, no qual $m_i = 8$ e há 2 colunas de arestas ímpares no caminho entre origem e destino de uma *commodity*. Note também que este resultado vale ainda que o caminho escolhido contenha arestas que não pertençam à submalha M_i, fato que possivelmente acontece em um roteamento desinformado.

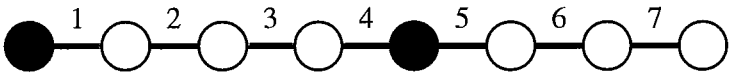


Figura 3.8: Colunas de arestas ímpares no caminho entre os vértices de origem e destino de uma *commodity*.

Como há $\frac{m_i}{2}$ *commodities* em $\mathcal{K}_i$, cada uma delas com demanda igual a d, a quantidade de demanda que deverá passar por arestas ímpares da malha M_i é de pelo menos $d \cdot \frac{m_i}{2}$.

Seja E_{imp}^{usada} o conjunto de arestas ímpares pelas quais estão passando fluxo no esquema de roteamento desinformado randomizado. Escolhendo de modo aleatório e uniforme uma aresta deste conjunto, a probabilidade de uma aresta $e_x \in E_{imp}^{usada}$ ser escolhida é de pelo menos $\frac{\frac{m_i}{4}}{m_i^2}$, já que pelo menos $\frac{m_i}{4}$ arestas ímpares estão sendo utilizadas.

Juntando as idéias desenvolvidas, o valor esperado da carga relativa das arestas pelas quais estão passando fluxo é:

$$E[Ld(e_x)] = \sum_{e \in E_{imp}^{usada}} p_e \, Ld_{\mathcal{K}_i}(e),$$

onde p_e é a probabilidade da aresta e ser escolhida e $Ld_{\mathcal{K}_i}(e)$ é a carga dada somente pelo fluxo das *commodities* do conjunto $\mathcal{K}_i$.

Desenvolvendo a desigualdade anterior com os valores conhecidos, temos:

$$\begin{aligned} E[Ld(e_x)] &= \sum_{e \in E_{imp}^{usada}} p_e \, Ld_{\mathcal{K}_i}(e) \\ &\geq \frac{\frac{m_i}{4}}{m_i^2} \left(\sum_{e \in E_{imp}^{usada}} Ld_{\mathcal{K}_i}(e) \right) \\ &= \frac{\frac{m_i}{4}}{m_i^2} \, d \cdot \frac{m_i}{2} \\ &= \frac{d}{8} \end{aligned}$$

Portanto, o roteamento das *commodities* do conjunto $\mathcal{K}_i$ produz uma carga relativa esperada de:

$$E[RLd(e_x)] = E\left[\frac{RLd(e_x)}{1}\right] \geq \frac{d}{8} \tag{3.1}$$

e esta equação vale para todas as submalhas M_i.

Como o algoritmo desinformado não conhece qual é o próximo conjunto $\mathcal{K}_{i+1}$, em cada submalha M_i do algoritmo, pode-se classificar a escolha da submalha M_{i+1} como uma escolha aleatória; logo, o próximo conjunto de arestas ímpares também é escolhido de forma aleatória e uniforme. Como um limite inferior para o congestionamento em uma aresta escolhida deste modo é dado por (3.1), o valor esperado para o congestionamento em uma aresta ímpar de M_i dado pelo roteamento de *commodities* que pertencem a $\mathcal{K}_i$, é $\frac{d}{8}$, $1 \leq i \leq L$.

Como uma aresta ímpar pode pertencer a todas as submalhas M_i, sua carga relativa será dada pela soma da contribuição de cada submalha:

$$\sum_{i=1}^{L} \frac{d}{8} = d \cdot \frac{\log(m) - 1}{8}.$$

Lembrando que $\log m = \log \sqrt{n}$, obtemos um limite inferior para o congestionamento de $\Omega(\log n)$. □

Capítulo 4

Algoritmos desinformados para redes com capacidades nos vértices e redes direcionadas

Este capítulo apresenta os algoritmos desinformados existentes na literatura para redes não-direcionadas com capacidades nos vértices e para redes direcionadas (com capacidades nas arestas ou vértices).

Os dois problemas são bastante similares, já que é possível facilmente transformar uma rede – direcionada ou não – que possui capacidades nos vértices em uma rede que possui capacidade nas arestas. Por isso, o algoritmo apresentado para roteamento desinformado em redes direcionadas poderá ser usado também para o caso de redes não-direcionadas com capacidades nos vértices, após modificação da entrada do problema.

As próximas seções deste capítulo são dedicadas a estes algoritmos.

4.1 Algoritmo para redes com capacidades nos vértices

O algoritmo de roteamento desinformado apresentado nesta seção foi proposto por Hajiaghayi *et al.* [22] e possui uma razão de competitividade de $O(\Delta \log^2 n \log\log n)$.

O método consiste em transformar o problema $\mathcal{R}(G, \mathcal{K})$, cujo objetivo é minimi-

zar congestionamento em uma rede não-direcionada com capacidades nos vértices, no problema $\mathcal{R}'(G', \mathcal{K}')$, cujo objetivo é a minimização do congestionamento em uma rede não-direcionada com capacidades nas arestas.

O algoritmo é composto de três passos básicos:

- Transformação do problema:
    - Transformação da rede $G(V, E)$ – não-direcionada com capacidades nos vértices – em uma rede $G'(V', E')$, não-direcionada com capacidade nas arestas;
    - Transformação do conjunto de *commodities* $\mathcal{K}$ em um conjunto $\mathcal{K}'$;
- Aplicação do algoritmo de Harrelson, Hildrum e Rao [15] (descrito no capítulo 7) na rede G';
- Transformação da solução não-informada de G' em uma solução do problema em G.

As próximas seções descrevem em detalhes os passos do algoritmo.

4.1.1 Transformação do problema

Este passo envolve duas transformações: a construção da rede G' a partir de G e a especificação do conjunto de *commodities* $\mathcal{K}'$ baseado em $\mathcal{K}$.

A construção de G' é descrita pela transformação de seus conjuntos V' e E':

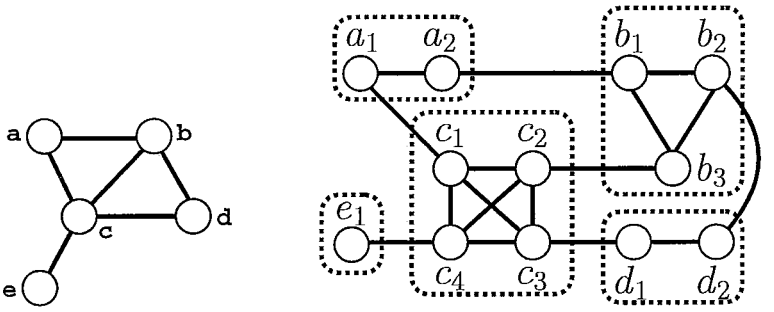
V': Cada vértice $v \in V$ (que possui capacidade $cap(v)$) é associado em V' a um conjunto de vértices $\mathcal{C}_v$, chamados de **clones** de v, de tal forma que:

- $\mathcal{C}_v = \{v_1, \ldots, v_{d(v)}\}$ é uma clique de tamanho $d(v)$ em G', onde $d(v)$ é o grau de v;
- Cada aresta de $\mathcal{C}_v$ possui capacidade igual a $\frac{cap(v)}{d(v)}$.

E': Além das arestas que ligam vértices de uma mesma clique, chamadas de **arestas internas**, o conjunto de arestas de G' conterá também arestas ligando vértices que pertencem a cliques diferentes, chamadas de **arestas externas**.

Para cada aresta $(u, v) \in E$ existe uma aresta $(v_i, u_j) \in E'$, onde v_i e u_j são, respectivamente, clones de v e u em G'. A escolha de quais vértices clones serão usados como extremidade de arestas externas é feita arbitrariamente, respeitando-se sempre que um vértice clone pode ter somente uma aresta externa incidindo nele.

A figura 4.1 exemplifica a transformação de uma rede G em uma rede G' pelo método descrito.



(a) (b)

Figura 4.1: Transformação de uma rede G em G'. A figura 4.1(a) mostra a rede inicial G. A figura 4.1(b) mostra a rede transformada G', com os clones de cada vértice de G demarcados por linhas pontilhadas.

As *commodities* de $\mathcal{K}'$ são construídas da seguinte forma: para cada *commodity* k de demanda d_k entre vértices $u, v \in V$ existirá uma *commodity* em $\mathcal{K}'$ de demanda idêntica entre u_i, um vértice clone de u, e v_j, um vértice clone de v. Os índices i e j são escolhidos de forma arbitrária dentro dos conjuntos $\{1, \ldots, d(u)\}$ e $\{1, \ldots, d(v)\}$, respectivamente.

4.1.2 Análise da solução do problema

A transformação de uma solução para o problema na rede G' para uma solução para o problema na rede G é imediata, visto que há uma bijeção entre as arestas externas de G' e as arestas da rede G: se um caminho P' em G' passa por algum vértice ou aresta de uma clique C_v, o caminho P, equivalente a P' em G, usa o vértice v. Por isso, a carga que passa em uma clique C_v em G' é igual à demanda passante por v em G.

A razão de competitividade do algoritmo é demonstrada no teorema 4.1:

Teorema 4.1. *O algoritmo desinformado apresentado na seção 4.1 possui uma razão de competitividade de* $O(\Delta \log^2 n \log\log n)$ *para o congestionamento em redes não-direcionadas com capacidades nos vértices.*

Demonstração. Aplicando o algoritmo de Harrelson, Hildrum e Rao [15] ao problema $\mathcal{R}'$, obtemos um esquema de roteamento que produz um congestionamento com um fator $O(\log^2 n \log\log n)$ em relação ao valor ótimo do congestionamento (produzido por um algoritmo informado ao rotear as *commodities* de $\mathcal{K}'$ em G'). Chamando de OBL_E e OPT_E os valores dos congestionamentos dados pelo algoritmo desinformado e pelo algoritmo ótimo na rede G', respectivamente, vale a igualdade:

$$OBL_E = O(\log^2 n \log\log n)\, OPT_E. \tag{4.1}$$

No entanto, para analisar a razão de competitividade do algoritmo para congestionamento nos vértices é necessário estabelecer relação entre OPT_V e OBL_V, o valor ótimo do congestionamento e o valor do congestionamento produzido pelo roteamento produzido pelo algoritmo desinformado na rede G, respectivamente. Para isto, serão usados dois resultados auxiliares, descritos a seguir nos lemas 4.2 e 4.3.

Lema 4.2. *Existe um fator de no máximo* 2 *entre os valores das congestionamentos ótimos nas redes* G *e* G'*:*

$$OPT_E \leq 2\, OPT_V.$$

Demonstração. A prova baseia-se em analisar como um roteamento que produz o congestionamento ótimo em G poderia se converter em um roteamento que produz a congestionamento ótimo em G', comparando estes dois valores.

Para analisar o congestionamento em G', é necessário apenas encontrar o maior valor dentre as cargas relativas de arestas internas às cliques, já que as arestas externas possuem capacidade infinita. Logo, o congestionamento em G' será determinado essencialmente pelo o que acontece quando uma fração da demanda passa por uma clique $\mathcal{C}_v$.

Por isso, uma transformação trivial de um esquema de roteamento em G para um esquema de roteamento em G' é converter cada caminho p que roteia uma fração ϵ

da demanda de uma *commodity* $k \in \mathcal{K}$ em um caminho p' em G' apenas trocando os vértices $v \in V$ por um esquema de roteamento dentro da clique $\mathcal{C}_v$.

Para detalhar o esquema de roteamento utilizado dentro de cada clique $\mathcal{C}_v$ ao receber uma fração ϵ de uma *commodity*, definimos um **vértice de origem** v_i (vértice pelo qual a *commodity* entrou na clique) e um **vértice de destino** v_j (vértice da clique que é o destino da *commodity* ou que possui aresta externa para algum vértice da próxima clique no caminho p'). O esquema de roteamento dentro da clique consiste em usar $d(v) - 1$ caminhos disjuntos para entregar a fração ϵ do vértice de origem ao vértice de destino: a aresta $\{v_i, v_j\}$ e cada caminho $\{v_i, v_k, v_j\}$, com $j \neq k$ e $j \neq i$, carregarão uma fração $\frac{\epsilon}{(d(v)-1)}$.

Neste esquema de roteamento, cada aresta interna de $\mathcal{C}_v$ recebe no máximo $\frac{1}{(d(v)-1)}$ de cada fração da demanda de *commodity* que entra nesta clique. Denotando por $Ld(\mathcal{C}_v)$ a soma das frações de todas as *commodities* que são roteadas passando pela clique $\mathcal{C}_v$, temos que para a carga relativa de qualquer aresta interna à clique $\mathcal{C}_v$ vale a seguinte desigualdade:

$$RLd(e) \leq \frac{Ld(\mathcal{C}_v)/(d(v)-1)}{cap(e)}.$$

Mas como $cap(e) = \frac{cap(v)}{d(v)}$,

$$RLd(e) \leq \frac{Ld(\mathcal{C}_v)/(d(v)-1)}{\frac{cap(v)}{d(v)}},$$

então

$$RLd(e) \leq \frac{d(v)}{(d(v)-1)} \frac{Ld(\mathcal{C}_v)}{cap(v)}.$$

Lembrando que $OPT_V \geq \frac{Ld(\mathcal{C}_v)}{cap(v)}$ para todo $v \in V$, conclui-se que

$$RLd(e) \leq \frac{d(v)}{(d(v)-1)} OPT_V.$$

Sabendo que $\frac{d(v)}{(d(v)-1)} \leq 2$ para qualquer valor de $d(v) \geq 2$ e observando que a equação anterior não existiria caso $d(v) = 1$, visto que não haverá arestas na clique $\mathcal{C}_v$ (logo, não faz sentido medir $RLd(e)$), consegue-se comparar o congestionamento ótimo em G' e em G:

$$OBL_E = congestionamento(G') \leq 2\,OPT_V.$$

□

Pelo resultado do lema anterior e considerando a equação (4.1), chega-se a seguinte equação:

$$OBL_E \leq 2\,O(\log^2 n \log\log n)\, OPT_V. \tag{4.2}$$

Lema 4.3. *Existe um fator de $O(\Delta)$ entre o congestionamento em arestas na rede G' e o congestionamento em vértices na rede G:*

$$OBL_V \leq \frac{\Delta - 1}{2} OBL_E,$$

onde Δ é o grau máximo de G.

Demonstração. Ao transformarmos novamente a rede G' em G, contraímos as cliques $\mathcal{C}_v$ para que se tornem novamente um único vértice em G. Queremos mostrar que, ao executarmos esta contração, a carga relativa de um vértice $v \in V$ pode aumentar de um fator de até $\frac{\Delta-1}{2}$ em relação ao valor do congestionamento em G'.

A prova analisa a relação entre a carga relativa de um vértice $v \in V$ e a carga relativa das arestas internas da clique $\mathcal{C}_v$ em G'.

A quantidade de demanda passando por um vértice $v \in V$ será a mesma passando pelas arestas que ligam seus vértices clones em G'. Denotando por $Ld(\mathcal{C}_v) = \sum_{e' \in \mathcal{C}_v} Ld(e')$ a soma das cargas absolutas das arestas internas da clique $\mathcal{C}_v$,

$$RLd(v) = \frac{Ld(\mathcal{C}_v)}{cap(v)}.$$

Lembrando que a clique $\mathcal{C}_v$ possui $\frac{d(v)(d(v)-1)}{2}$ arestas internas e percebendo que $Ld(\mathcal{C}_v) \leq \frac{d(v)(d(v)-1)}{2} \max_{e' \in \mathcal{C}_v}\{Ld(e')\}$, pode-se reescrever a equação anterior:

$$RLd(v) \leq \frac{\frac{d(v)(d(v)-1)}{2} \max_{e' \in \mathcal{C}_v}\{Ld(e')\}}{cap(v)}. \tag{4.3}$$

Lembrando que OBL_E denota o congestionamento que o algoritmo desinformado produz em G', vale para qualquer aresta $e' \in \mathcal{C}_v$ a seguinte inequação:

$$\begin{aligned} Ld(e') &= cap(e')\, RLd(e') \\ &\leq \frac{cap(v)}{d(v)} OBL_E. \end{aligned}$$

Utilizando este resultado para reescrever a equação (4.3):

$$\begin{aligned} RLd(v) &\leq \frac{\frac{d(v)(d(v)-1)}{2}\frac{cap(v)}{d(v)}OBL_E}{cap(v)} \\ &= \frac{(d(v)-1)}{2}OBL_E. \end{aligned}$$

Como $d(v) = \Delta$ para algum vértice $v \in V$, chega-se a um valor para o congestionamento em G:

$$OBL_V = congestionamento(G) \leq \frac{\Delta - 1}{2} OBL_E.$$

Logo, ao transformarmos uma solução para o problema em G' para uma solução para o problema em G através da contração das cliques, temos um fator de $O(\Delta)$ entre os valores dos congestionamentos. □

A prova do teorema segue do lema 4.3 e da equação (4.2):

$$OBL_V = O(\Delta \log^2 n \log\log n)\, OPT_V.$$

Portanto, este algoritmo produz um congestionamento que está a um fator de $O(\Delta \log^2 n \log\log n)$ do ótimo. □

4.2 Algoritmo para redes direcionadas com capacidades nas arestas

O algoritmo apresentado nesta seção foi proposto por Hajiaghayi *et al.* [22, 32] e possui uma razão de competitividade de $O(\sqrt{\alpha|\mathcal{K}|}\log n)$, onde α é o valor descrito no teorema 2.4 e $|\mathcal{K}|$ é o número de *commodities* da entrada do problema. Como visto anteriormente, o valor de α depende do tipo de rede e do tipo de demandas considerados. No caso mais geral de demandas para redes direcionadas, temos $\alpha = \sqrt{n}$ [33], resultando portanto em um algoritmo com razão de competitividade $O(\sqrt{|\mathcal{K}|}\, n^{\frac{1}{4}} \log n)$.

4.2.1 Esquema de roteamento

Primeiramente será descrito o algoritmo para calcular o esquema de rotamento para uma rede direcionada qualquer, para que então seja mostrado que este método possui a razão de competitividade desejada.

Novamente será utilizado o fato de que um esquema de roteamento pode ser visto como a escolha de caminhos para rotear uma demanda entre cada par de vértices da rede. Para calcular o esquema de roteamento, será usado um conjunto de demandas entre cada par de vértices, chamado de $\mathcal{D}$. Este conjunto é distinto e independente do conjunto de entrada $\mathcal{K}$: a demanda entre um par de vértices u, v em $\mathcal{D}$ depende do valor de um corte mínimo separando sua origem e destino: se o valor do corte mínimo entre u e v pertence ao conjunto $[2^j, 2^{j+1})$ para algum $j \geq 0$, então a demanda entre este par é igual a 2^j. A escolha das rotas é portanto independente do conjunto de demandas da entrada do algoritmo, sendo por isto um esquema de roteamento desinformado.

A fim de definir o esquema de roteamento, o conjunto $\mathcal{D}$ será particionado em subconjuntos disjuntos $\mathcal{D}_j$, $j = 0, 1, 2, \ldots$. Um par $\langle u, v \rangle$ pertence ao conjunto $\mathcal{D}_j$ se o valor de um corte mínimo entre sua origem e destino está no intervalo $[2^j, 2^{j+1})$. Esta partição, portanto, coloca no mesmo conjunto pares de vértices que possuam aproximadamente o mesmo valor de corte mínimo entre suas origens e destinos, e cujas demandas em $\mathcal{D}$ são iguais a 2^j.

Para o cálculo do esquema de roteamento, cada um dos conjuntos $\mathcal{D}_j$ será considerado um problema de roteamento independente. A solução do problema para o conjunto $\mathcal{D}$ será composta da solução de cada subconjunto $\mathcal{D}_j$, isto é, o esquema de roteamento desinformado será a união destes esquemas de roteamento para cada $\mathcal{D}_j$.

Conforme será mostrado posteriormente, o esquema de roteamento para um subconjunto $\mathcal{D}_j$ produzirá o congestionamento desejado ao rotear as *commodities* de qualquer problema de entrada que tenham origem e destino em pares pertencentes ao subconjunto $\mathcal{D}_j$, isto é, *commodities* cuja origem e destino estejam separados por um corte mínimo com capacidade no intervalo $[2^j, 2^{j+1})$.

Para cada $\mathcal{D}_j$, o roteamento será definido baseado na obtenção de um multifluxo concorrente máximo restrito às demandas dos pares que pertencem a este subconjunto. Na solução do multifluxo concorrente máximo, o objetivo é que o esquema de roteamento do conjunto $\mathcal{D}_j$ produza congestionamento de $O(\sqrt{\alpha|\mathcal{K}|})$. Baseado neste objetivo, pode-se dividir os pares de vértices pertencentes ao conjunto $\mathcal{D}_j$ em dois subconjuntos disjuntos:

- Os pares que conseguem atender simultaneamente uma demanda igual a 2^j

pelo multifluxo concorrente máximo produzindo congestionamento menor ou igual a $\sqrt{\alpha|\mathcal{K}|}$;

- Os pares cujas origens e destinos atravessam um corte com congestionamento maior que $\sqrt{\alpha|\mathcal{K}|}$ no multifluxo concorrente máximo.

Estes conjuntos serão chamados de A_j e B_j, respectivamente.

Dado um conjunto $\mathcal{D}_j$, os conjuntos A_j e B_j podem ser computados pelo algoritmo 1, apresentado a seguir, em tempo polinomial:

Algoritmo 1 Partição de $\mathcal{D}_j(G, \mathcal{D}_j)$

$S_j := \emptyset$;
$B_j := \emptyset$;
$A_j := \mathcal{D}_j$;
$F := CMCF(G, A_j)$;
enquanto $cong_F(G) > \sqrt{\alpha\mathcal{K}}$ **faça**
  $S' :=$ corte mais esparso aproximado(G, A_j);
  $S_j := S_j \cup S'$;
  $B_j := B_j \cup \mathcal{D}_{S'}$; {adiciona *commodities* separadas pelo corte S'}
  $A_j := A_j \setminus \mathcal{D}_{S'}$; {remove *commodities* separadas pelo corte S'}
retorna S_j, A_j, F;

O algoritmo começa considerando um multifluxo concorrente máximo do conjunto $\mathcal{D}_j$. Caso o congestionamento tenha um valor superior a $\sqrt{\alpha|\mathcal{K}|}$, pelos resultados da seção 2.1, a fração de vazão z^* do multifluxo concorrente máximo é tal que:

$$z^* < \frac{1}{\sqrt{\alpha|\mathcal{K}|}}.$$

Na seção 2.4 mencionamos que encontrar um corte mais esparso é um problema NP-difícil. Como deseja-se encontrar os subconjuntos A_j e B_j em tempo polinomial, será usado um algoritmo α-aproximativo de tempo polinomial – denotado no algoritmo pela função `corte mais esparso aproximado` – para encontrar um corte mais esparso aproximado S'.

Este algoritmo retornará um corte S' tal que $esp(S') = \alpha\, z^*$; portanto:

$$\begin{aligned} esp(S') &= \alpha\, z^* \\ &\leq \alpha \frac{1}{\sqrt{\alpha|\mathcal{K}|}} \\ &= \sqrt{\frac{\alpha}{|\mathcal{K}|}}. \end{aligned}$$

O conjunto de pares de vértices $\mathcal{D}_{S'}$ separados pelo corte S' produz congestionamento maior que $\sqrt{\alpha|\mathcal{K}|}$. Estes pares são adicionados ao conjunto B_j e as arestas de S' são inseridas no corte S_j, a fim de marcar que estas arestas separam pares de vértices cujos roteamentos das demandas produzem congestionamento maior do que o desejado. O processo continua até que um multifluxo concorrente máximo dos pares de vértices pertencentes ao conjunto A_j produza congestionamento menor que $\sqrt{\alpha|\mathcal{K}|}$. Como o conjunto A_j diminui a cada iteração e pode se tornar vazio, o algoritmo sempre pára em tempo polinomial em $|D_j|$, sendo portanto polinomial em n.

Como o conjunto B_j produz congestionamento maior do que $\sqrt{\alpha|\mathcal{K}|}$, existe um corte com esparsidade menor ou igual a $\frac{\alpha}{\sqrt{\alpha|\mathcal{K}|}} = \sqrt{\frac{\alpha}{|\mathcal{K}|}}$ separando os pares que pertencem a este conjunto. O lema 4.4 demonstra que o corte S_j produzido pelo algoritmo 1 produz um corte com a esparsidade citada.

Lema 4.4. *O corte S_j produzido pelo algoritmo 1 tem esparsidade menor ou igual a $\sqrt{\frac{\alpha}{|\mathcal{K}|}}$.*

Demonstração. Usaremos indução para provar o lema. Inicialmente vazio, o corte S_j é acrescido das arestas de S' a cada iteração do algoritmo. Na base, isto é, na primeira iteração, o corte S_j será igual ao corte S'. Denotando por $d(\mathcal{K}_{S'})$ a demanda entre pares de vértices de separados por S', temos

$$cap(S) \leq \sqrt{\frac{\alpha}{|\mathcal{K}|}}\, d(\mathcal{K}_{S'}),$$

porque o conjunto $d(\mathcal{K}_{S'})$ foi roteado com congestionamento maior que $\sqrt{\frac{\alpha}{|\mathcal{K}|}}$. Na base, portanto, a esparsidade de S_j será igual à esparsidade de S'; logo, a esparsidade de S_j também é menor ou igual a $\sqrt{\frac{\alpha}{|\mathcal{K}|}}$.

No passo indutivo, considera-se que depois de i passos do algoritmo, o corte S_j possui esparsidade menor ou igual a $\sqrt{\frac{\alpha}{|\mathcal{K}|}}$. Na iteração $i+1$, seja $d(S_j)$ a demanda dos pares separados pelo corte S_j até a iteração i do algortimo, isto é, $d(S_j)$

é igual à demanda do conjunto B_j construído até aquele momento do algoritmo. Como $cap(S_j) \leq \sqrt{\frac{\alpha}{|\mathcal{K}|}}\, d(S_j) = \sqrt{\frac{\alpha}{|\mathcal{K}|}}\, |B_j|\, 2^j$, então, na $i+1$-ésima iteração do algoritmo, temos:

$$\begin{aligned} cap(S_j \cup S') &\leq cap(S_j) + cap(S') \\ &= \sqrt{\frac{\alpha}{|\mathcal{K}|}}\, |B_j|\, 2^j + \sqrt{\frac{\alpha}{|\mathcal{K}|}}\, |\mathcal{K}_{S'}|\, 2^j \\ &= \sqrt{\frac{\alpha}{|\mathcal{K}|}}\, |B_j \cup \mathcal{K}_{S'}|\, 2^j. \end{aligned}$$

Observe que a primeira linha é uma desigualdade porque o conjunto de arestas de S' não é necessariamente disjunto de S_j.

Logo, a esparsidade do corte S_j produzido ao final da $i+1$-ésima iteração é:

$$\begin{aligned} esp(S_j \cup S') &= \frac{cap(S_j \cup S')}{d(S_j \cup S')} \\ &\leq \frac{\sqrt{\frac{\alpha}{|\mathcal{K}|}}\, |B_j \cup \mathcal{K}_{S'}|\, 2^j}{|B_j \cup \mathcal{K}_{S'}|\, 2^j} \\ &= \sqrt{\frac{\alpha}{|\mathcal{K}|}}. \end{aligned}$$

□

Para os pares do subconjunto A_j, o roteamento através dos caminhos entre cada par de vértices deste conjunto existentes na solução do multifluxo concorrente máximo (calculados no algoritmo 1) produz congestionamento menor ou igual a $\sqrt{\alpha|\mathcal{K}|}$. Para limitar o número de vezes que cada aresta é utilizada, o algoritmo evita utilizar no esquema de roteamento dos pares de vértices de $\mathcal{D}_j$ arestas que possuam "pouca" capacidade em relação às demandas que por elas passam: o esquema de roteamento do multifluxo concorrente máximo será modificado a fim de evitar que arestas que possuam capacidade menor que $\frac{2^j}{2n^4}$ sejam utilizadas. Este fato será usado para limitar o número de conjuntos $\mathcal{K}_j$ que podem utilizar uma aresta e, possibilitando assim obter um limite superior menor para o congestionamento deste esquema de roteamento.

Ao alterar o esquema de roteamento, torna-se necessário medir o impacto que esta mudança produzirá no congestionamento. A proposição 4.5 indica um limite superior para a quantidade de demanda que terá seus caminhos de roteamento alterados.

Proposição 4.5. *Na solução do problema do multifluxo concorrente máximo, no máximo metade da demanda de um par pertencente ao subconjunto $\mathcal{D}_j$ é roteada por caminhos que contenham arestas com capacidade menor que $\frac{2^j}{2n^4}$.*

Demonstração. Vamos denotar por $|E_{peq}|$ o número de arestas que possuem capacidade menor que $\frac{2^j}{2n^4}$ que são usadas para rotear a demanda de um par pertencente a A_j. Como temos um congestionamento de no máximo $\sqrt{\alpha|\mathcal{K}|}$, a quantidade de fluxo que passa por estas arestas é de no máximo

$$|E_{peq}| \frac{2^j}{2n^4} \sqrt{\alpha|\mathcal{K}|}.$$

Como $|E_{peq}| \leq n^2$, $\sqrt{|\mathcal{K}|} \leq n$ e é um limite trivial que $\alpha < |\mathcal{K}|$ [34], temos:

$$|E_{peq}| \frac{2^j}{2n^4} \sqrt{\alpha|\mathcal{K}|} \leq n^2 \frac{2^j}{2n^4} \sqrt{|\mathcal{K}|^2} \leq \frac{2^j}{2n^2} |\mathcal{K}| \leq \frac{2^j}{2}.$$

Como a demanda dos pares pertencentes ao conjunto $\mathcal{D}_j$ é 2^j, temos que no máximo metade da demanda de cada par é roteada por caminhos que contenham arestas pertencentes ao conjunto $|E_{peq}|$. □

Com o resultado da proposição 4.5, é possível alterar o esquema de roteamento no máximo duplicando a quantidade de fluxo que passa por caminhos que não contenham arestas com capacidade menor do que $\frac{2^j}{2n^4}$. Este esquema, portanto, terá seu congestionamento multiplicado por um fator de 2 em relação ao congestionamento produzido na solução do multifluxo concorrente máximo; logo, o roteamento das demandas dos pares pertencentes ao subconjunto A_j produz congestionamento menor ou igual a $2\sqrt{\alpha|\mathcal{K}|}$.

Para cada par $\langle u, v \rangle \in B_j$, o algoritmo desinformado usará o mesmo esquema de roteamento que um multifluxo concorrente máximo ao considerar a demanda entre u e v como a única *commodity* do problema. Desta forma, obtém-se um fluxo que roteia a demanda deste par de vértices e minimiza o congestionamento.

Do mesmo modo feito anteriormente, a solução final será o esquema de roteamento alterado para evitar arestas que possuam capacidade menor que $\frac{2^j}{2n^4}$. Como a proposição 4.5 vale para qualquer par de vértices do conjunto $\mathcal{D}_j$, a escolha de não rotear fluxo por estas arestas também aumentará o congestionamento do conjunto B_j em um fator de no máximo 2.

4.2.2 Análise da solução do problema

De forma análoga a feita anteriormente para definir subconjuntos $\mathcal{D}_j$, pode-se dividir o conjunto de *commodities* $\mathcal{K}$ em subconjuntos $\mathcal{K}_j$: uma *commodity* k pertence a $\mathcal{K}_j$ se o valor do corte mínimo entre s_k e t_k pertence ao conjunto $[2^j, 2^{j+1})$.

Com esta divisão, há uma relação entre as *commodities* de um conjunto $\mathcal{K}_j$ e os pares que no esquema de roteamento pertenciam ao conjunto $\mathcal{D}_j$: se $k \in \mathcal{K}_j$, então $\langle s_k, t_k \rangle \in \mathcal{D}_j$; logo, o esquema de roteamento do subconjunto $\mathcal{D}_j$ será usado para rotear as *commodities* de $\mathcal{K}_j$.

O lema 4.6 mostra que, para qualquer entrada dada, o esquema de roteamento proposto produz congestionamento a um fator de no máximo $O(\sqrt{\alpha|\mathcal{K}|})$ do ótimo para as *commodities* cujas demandas estejam no conjunto $[2^j, 2^{j+1})$, ou seja, para as *commodities* pertencentes ao subconjunto $\mathcal{K}_j$.

Lema 4.6. *O esquema de roteamento proposto produz congestionamento* $O(\sqrt{\alpha|\mathcal{K}|})$ *ao rotear as demandas das* commodities *de um conjunto* $\mathcal{K}_j$*, onde* $\mathcal{K}_j$ *é o conjunto de* commodities *cujo corte mínimo está no conjunto* $[2^j, 2^{j+1})$.

Demonstração. Seja $k \in \mathcal{K}_j$ uma *commodity* do problema de entrada. Se o valor ótimo do congestionamento para a entrada dada é OPT, temos que $d_k \leq OPT \cdot CM(s_k, t_k)$, onde $CM(s_k, t_k)$ é o valor do corte mínimo entre s_k e t_k.

Lembrando que a demanda entre o par s_k e t_k em $\mathcal{D}_j$ era de 2^j porque o valor de um corte mínimo separando este par pertencia ao conjunto $[2^j, 2^{j+1})$, concluímos que esta demanda é menor do que $2\,CM(s_k, t_k)$. Logo, a demanda da *commodity* k do problema de entrada não está a um fator maior do que $2OPT$ em relação à demanda entre o par s_k e t_k em $\mathcal{D}_j$. Logo, se o par de vértices $\langle s_k, t_k \rangle$ pertencia ao conjunto A_j, o congestionamento será aumentado de um fator de $2OPT$, resultando em um congestionamento de $2OPT\,(2\sqrt{\alpha|\mathcal{K}|})$; isso mostra que o congestionamento para o conjunto A_j está a um fator de $4\sqrt{\alpha|\mathcal{K}|}$ do valor ótimo para a mesma instância.

Para medir o congestionamento produzido ao rotearmos uma *commodity* do problema de entrada cujo par de vértices de origem e destino pertence ao conjunto B_j, precisamos medir a quantidade de fluxo que este conjunto passa por uma aresta e.

As rotas para a demanda destes pares são dadas pela solução de um multifluxo concorrente máximo que minimiza o congestionamento considerando somente esta *commodity* e evita arestas com capacidade menor que $\frac{2^j}{2n^4}$. Logo, esta *commodity* k produz carga relativa em uma aresta e tal que:

$$RLd(e) \leq \frac{2\,d_k}{CM(s_k, t_k)}.$$

Lembrando que as *commodities* de B_j são separadas pelo corte S_j, temos $d(S_j) = \sum_{k \in B_j} d_k$. Além disso, vale também para qualquer *commodity* k do conjunto: $2^j \leq CM(s_k, t_k)$.

Portanto, contando a contribuição de todas as *commodities* de B_j, temos:

$$RLd(e) \leq \frac{2\,d(S_j)}{2^j}.$$

Mas como existe uma solução com congestionamento ótimo igual a OPT, para o corte S_j vale que $d(S_j) \leq OPT\, cap(S_j)$. Reescrevendo a equação anterior:

$$RLd(e) \leq \frac{2\,OPT\, cap(S_j)}{2^j}.$$

Mas é resultado imediato do lema 4.4 que $cap(S_j) \leq \sqrt{\frac{\alpha}{|\mathcal{K}|}}\, 2^j\, |B_j|$, logo:

$$\begin{aligned} RLd(e) &\leq \frac{2\,OPT\, \sqrt{\alpha/|\mathcal{K}|}\, 2^j\, |B_j|}{2^j} \\ &= 2\,OPT\, \sqrt{\alpha/|\mathcal{K}|}\, |B_j|. \end{aligned}$$

Usando $|B_j| \leq |\mathcal{K}|$, concluímos que:

$$RLd(e) \leq 2\,OPT\, \sqrt{\alpha|\mathcal{K}|},$$

para toda aresta e da rede.

Para as *commodities* de B_j temos portanto um congestionamento que está a um fator de $2\sqrt{\alpha|\mathcal{K}|}$ do valor ótimo para a instância fornecida.

Podemos concluir que o algoritmo desinformado restrito às *commodities* de um conjunto $\mathcal{K}_j$ produz um congestionamento de $6\,OPT\, \sqrt{\alpha|\mathcal{K}|}$, que corresponde à soma das contribuições dos conjuntos A_j e B_j. Isto representa um fator de $6\sqrt{\alpha|\mathcal{K}|} = O(\sqrt{\alpha|\mathcal{K}|})$ entre o valor ótimo para o congestionamento e o valor dado pelo algoritmo desinformado, para qualquer instância dada. □

Falta apenas provar que cada aresta da rede é utilizada no esquema de roteamento de um número limitado de conjuntos $\mathcal{K}_j$, resultando na razão de competitividade desejada.

Lema 4.7. *No algoritmo desinformado, uma aresta e é utilizada somente no esquema de roteamento de um número limitado de conjuntos $\mathcal{K}_j$. Para esta aresta, podemos dividir estes conjuntos que a utilizam em dois tipos:*

- *Os conjuntos $\mathcal{K}_j$ tais que suas* commodities *podem mandar suas demandas integralmente pela aresta e e ainda produzir carga relativa menor ou igual a OPT;*
- *Os conjuntos $\mathcal{K}_j$ tais que cada um deles produz carga relativa de $O(\sqrt{\alpha|\mathcal{K}|})$ em relação ao congestionamento ótimo. Existem $O(\log n)$ conjuntos deste tipo.*

Demonstração. Uma aresta e é utilizada pelo esquema de roteamento de um conjunto $\mathcal{K}_j$ somente se $cap(e) \geq \frac{2^j}{2n^4}$. Logo, cada aresta possui uma faixa de valores de j para os quais esta desigualdade vale.

Seja $j_{\max}$ o maior j para o qual a aresta e ainda é utilizada pelo roteamento de $\mathcal{K}_j$. Para esta aresta, podemos dividir o conjunto de *commodities* que a utiliza em dois grupos:

- $\mathcal{K}_0, \ldots, \mathcal{K}_{j'-1}$: Existe um valor máximo para j, chamado aqui de j', para o qual vale:
$$cap(e) \geq 2^{j'} n^2.$$
- $\mathcal{K}_{j'}, \ldots, \mathcal{K}_{j_{\max}}$: Para estes conjuntos $\mathcal{K}_j$, $j' \leq j \leq j_{max}$, temos $cap(e) < 2^{j+1} n^2$.

Para os conjuntos $\mathcal{K}_0, \ldots, \mathcal{K}_{j'-1}$, como temos no máximo n^2 *commodities*, podemos rotear toda a demanda das *commodities* pela aresta e produzindo carga absoluta menor do que $cap(e)$. A carga relativa que estas *commodities* provocam em uma aresta e é de
$$RLd(e) = \frac{\sum_{k \in \mathcal{K}_0, \ldots, \mathcal{K}_{j'-1}} d_k}{cap(e)}.$$
Mas como $cap(e) \geq 2^{j'} n^2$, temos:
$$RLd(e) \leq \frac{\sum_{k \in \mathcal{K}_0, \ldots, \mathcal{K}_{j'-1}} d_k}{2^{j'} n^2}.$$

Lembrando que o valor de um corte mínimo entre cada origem e destino de *commodities* que pertença aos conjuntos $\mathcal{K}_j$, $0 \leq j < j'$, é maior ou igual a 2^j, para qualquer entrada do algoritmo que possua congestionamento ótimo OPT, se $k \in \mathcal{K}_0, \ldots, \mathcal{K}_{j'-1}$, temos:

$$d_k < 2^{j'} OPT.$$

Logo, $\sum_{k \in \mathcal{K}_0, \ldots, \mathcal{K}_{j'}} d_k < 2^{j'} OPT\, n^2$. Reescrevendo a carga da aresta e, temos:

$$RLd(e) \leq \frac{2^{j'} OPT\, n^2}{2^{j'} n^2} = OPT.$$

Concluímos então que para os conjuntos $\mathcal{K}_0, \ldots, \mathcal{K}_{j'-1}$ o algoritmo desinformado pode produzir carga relativa menor igual a OPT em qualquer aresta e.

Para os conjuntos $\mathcal{K}_{j'}, \ldots, \mathcal{K}_{j_{\max}}$, pelo resultado do lema 4.6 temos que cada um deles produz pelo esquema de roteamento congestionamento menor ou igual a $O(OPT\sqrt{\alpha|\mathcal{K}|})$. Com isso, esses $|j_{\max} - j'|$ conjuntos podem usar a aresta e, produzindo carga relativa $O(|j_{\max} - j'|\sqrt{\alpha|\mathcal{K}|\, OPT})$. Precisamos então saber o valor de $|j_{\max} - j'|$.

Como temos $2^{j'+1} n^2 > cap(e)$ e para que e seja usada no roteamento destes conjuntos vale que $cap(e) \geq \frac{2^{j_{max}}}{2n^4}$, temos $2^{j'+1} n^2 > cap(e) \geq \frac{2^{j_{max}}}{2n^4}$.

Logo:

$$\log(2^{j'+1} n^2) > \log(\tfrac{2^{j_{max}}}{2n^4})$$
$$j' + 1 + 2\log n > j_{max} - 8\log n$$
$$j_{max} < j' + 6\log n - 1.$$

Logo, $j_{max} - j' < 6\log n - 1 = O(\log n)$.

Temos, portanto, $O(\log n)$ conjuntos $\mathcal{K}_j$ que produzem carga relativa de $O(OPT\sqrt{\alpha|\mathcal{K}|})$. □

O teorema 4.8 conclui a prova da razão de competitividade do algoritmo.

Teorema 4.8. *O algoritmo desinformado proposto na seção 4.2.1 possui razão de competitividade $O(\sqrt{\alpha|\mathcal{K}|}\log n)$ para o congestionamento em redes direcionadas com capacidades nas arestas.*

Demonstração. A razão de competitividade segue dos resultados dos lemas 4.6 e 4.7.

Para conseguir a carga relativa de uma aresta e, somamos a contribuição de cada conjunto $\mathcal{K}_j$ que a utiliza. Somando a contribuição dos conjuntos $\mathcal{K}_0, \ldots, \mathcal{K}_{j'-1}$ e dos conjuntos $\mathcal{K}_{j'}, \ldots, \mathcal{K}_{j_{\max}}$, temos:

$$\begin{aligned} RLd(e) &\leq O(OPT + OPT\sqrt{\alpha|\mathcal{K}|}\log n) \\ &= O(OPT\sqrt{\alpha|\mathcal{K}|}\log n) \end{aligned}$$

Logo, o congestionamento do algoritmo desinformado está a um fator de $O(\sqrt{\alpha|\mathcal{K}|}\log n)$ do valor ótimo. □

Capítulo 5

A árvore de decomposição

Este capítulo apresenta o conceito de árvore de decomposição de uma rede, que será usado nos algoritmos B.K.R e H.H.R. (descritos nos capítulos 6 e 7, respectivamente) para obter um esquema de roteamento desinformado. Estes algoritmos diferem na forma de construir esta árvore, mas a utilizam de forma idêntica para obter um esquema de roteamento na rede.

A próxima seção apresenta o conceito de decomposição hierárquica e árvore de decomposição e a seção 5.2 descreve como os algoritmos dos capítulos 6 e 7 obtêm o esquema de roteamento a partir desta árvore.

5.1 Decomposição hierárquica

Denotando por $G[V']$ o subgrafo induzido por um conjunto $V' \subseteq V$, uma **decomposição hierárquica** $\mathcal{D}_G$ de um grafo $G = (V, E)$ é um conjunto de subgrafos induzidos de G que possui as seguintes propriedades:

1. $G \in \mathcal{D}_G$;
2. Para todo vértice $v \in V$, $G[\{v\}] \in \mathcal{D}_G$;
3. Para todo par de subgrafos $H_1, H_2 \in \mathcal{D}_G$, uma das afirmativas é verdadeira:
   - $H_1 \subset H_2$, ou
   - $H_1 \cap H_2 = \emptyset$.

Note que a definição permite que os subgrafos criados não sejam conexos. Porém, nos algoritmos para construção de árvore de decomposição que serão vistos nos próximos capítulos, os subgrafos criados sempre serão conexos.

A figura 5.1 mostra um grafo G e uma de suas possíveis decomposições hierárquicas, onde cada subgrafo pertencente à decomposição é representado por uma linha que o envolve e $\mathcal{D}_G = \{G[\{a\}], G[\{b\}], G[\{c\}], G[\{d\}], G[\{e\}], G[\{f\}], G[\{g\}], G[\{h\}], G[\{i\}], G[\{j\}], G[\{c,e\}], G[\{h,j\}], G[\{a,c,e\}], G[\{h,i,j\}], G[\{b,d,h,i,j\}], G[\{a,b,c,d,e,f,g,h,i,j\}]\}$.

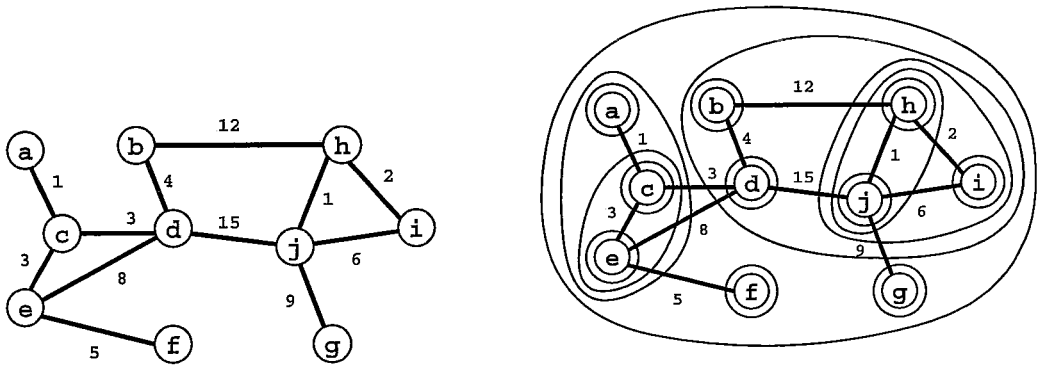


Figura 5.1: Um grafo G e uma possível decomposição hierárquica de G.

A primeira e segunda propriedades de uma decomposição hierárquica fazem com que o subgrafo induzido por V e os subgrafos $G[\{v\}]$, para todo $v \in V$, estejam sempre presentes em qualquer decomposição do grafo $G(V, E)$. Já a terceira propriedade fornece uma forma fácil para construir recursivamente uma decomposição hierárquica de um grafo G: iniciando a decomposição somente com o subgrafo induzido pelo conjunto V, encontra-se uma partição em vértices $\mathcal{P} = \{V_1, V_2, \ldots, V_p\}$ deste subgrafo. Os subgrafos $G[V_1], \ldots, G[V_p]$ são inseridos na decomposição de G e repete-se este procedimento para cada um deles. Executando este processo até que todos os subgrafos triviais tenham sido inseridos na partição obtém-se uma decomposição de G.

Portanto, uma decomposição hierárquica $\mathcal{D}_G$ de um grafo G pode ser recursivamente construída pelos seguintes passos:

- Caso o grafo G seja um grafo trivial, $\mathcal{D}_G = \{G\}$;

- Caso o grafo G possua mais de um vértice, seja $\mathcal{P}_G = \{V_1, V_2, \ldots, V_p\}$ uma partição em vértices de G. Temos então que $\mathcal{D}_G = \{G\} \cup_{i=1}^{p} \mathcal{D}_{G[V_i]}$, isto é, uma decomposição do grafo G é composta por ele próprio e por decomposições hierárquicas dos subgrafos induzidos por cada conjunto que compõe sua partição em vértices.

Por estas propriedades, a decomposição hierárquica pode ser vista como uma partição do conjunto de vértices do grafo G que cria uma família laminar de subconjuntos de $V(G)$.

Pela forma de construção de uma decomposição hierárquica, uma **árvore de decomposição** $\mathcal{T}_{\mathcal{D}_G}$ é diretamente associada a uma decomposição $\mathcal{D}_G$ de um grafo G. Na árvore, há um nó para cada subgrafo pertencente à decomposição hierárquica, e existe aresta entre dois nós se:

- $H_1 \subset H_2$, e,
- Não existe $H' \in \mathcal{D}_G$ tal que $H_1 \subset H' \subset H_2$.

A árvore de decomposição será usada de maneira essencial pelo algoritmo de roteamento desinformado.

A figura 5.2 mostra a árvore de decomposição associada a uma decomposição hierárquica de um grafo G, dados na figura 5.1.

Como há uma bijeção entre os subgrafos da decomposição $\mathcal{D}_G$ e o conjunto de nós da árvore de decomposição, $\mathcal{D}_G$ também será usado para denotar o conjunto de nós da árvore $\mathcal{T}_{\mathcal{D}_G}$.

Em particular, como usual, o **nível** de um nó H é dado pelo número de arestas do caminho entre a raiz de $\mathcal{T}_{\mathcal{D}_G}$ e H. A raiz, portanto, está no nível zero, e a **altura** da árvore é $h = \max_{H \in \mathcal{D}_G}\{\text{nível}(H)\}$.

Uma aresta de G é **cortada no nível** ℓ caso seus dois extremos pertençam ao mesmo nó no nível ℓ da árvore, mas no nível $\ell + 1$ estejam em nós distintos. Por exemplo, na figura 5.2, a aresta (a, c) é cortada no nível 1. É interessante notar o conjunto de arestas cortadas no nível ℓ é um conjunto de arestas que define de maneira única os nós do nível $\ell + 1$, pois sua retirada do grafo do nível ℓ cria os conjuntos disjuntos que serão os nós do nível $\ell + 1$. Veremos futuramente que os algoritmos de roteamento construirão árvores de decomposição através de cortes

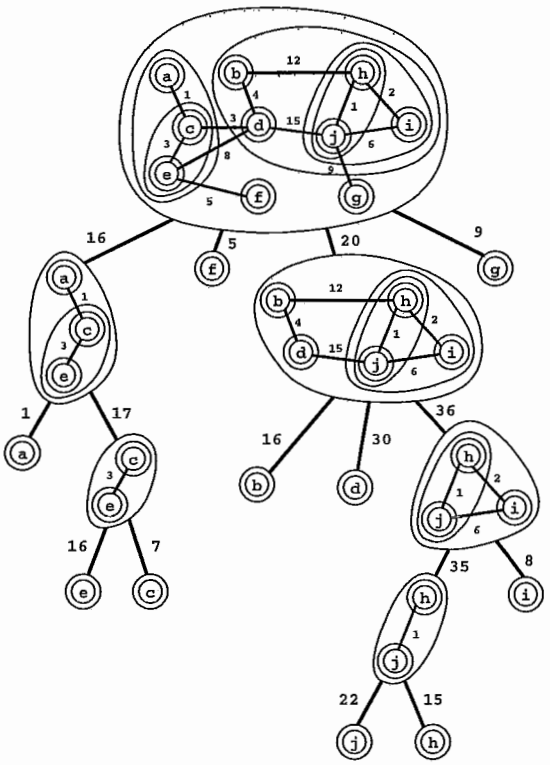


Figura 5.2: Árvore de decomposição $T_{\mathcal{D}_G}$ da decomposição hierárquica $\mathcal{D}_G$.

sucessivos, e que escolher adequadamente o conjunto de arestas cortadas em cada nível terá importante papel na razão de competitividade dos algoritmos.

Uma **aresta externa de um vértice v no nível ℓ** é uma aresta que liga v a um outro vértice que não pertence ao mesmo nó que v no nível ℓ. Ou seja, uma aresta é externa em um nível ℓ se e somente se ela é cortada em algum nível menor do que ℓ. Dado um grafo H no nível ℓ e $X \subseteq V(H)$, o **conjunto de arestas externas** de X é o conjunto das arestas externas de cada um dos vértices de X.

A **capacidade externa de um vértice** v no nível ℓ, $w_\ell(v)$, é a soma das capacidades das arestas externas de v no nível ℓ, isto é, a capacidade externa w_ℓ de um vértice v é dada por:

$$w_\ell(v) = \sum_{\substack{u \in N(v) \\ e=(v,u) \text{ é cortada no nível } k, k<\ell}} cap(e).$$

A **capacidade externa de um conjunto de vértices** X de um grafo H no nível ℓ, $w_\ell(X)$, é a soma da capacidade externa de cada vértice de X, isto é:

$$w_\ell(X) = \sum_{v \in X} w_\ell(v).$$

Na figura 5.2, o número associado a cada aresta da árvore é a capacidade externa do conjunto de vértices do subgrafo que está sendo ligado a seu pai.

Duas propriedades da função capacidade externa w_ℓ, que merecem destaque, são:

1. Para conjuntos de vértices X, X_1, X_2 tais que $X = X_1 \uplus X_2$, se os vértices de X pertencem a um mesmo nó no nível ℓ, temos: $w_\ell(X) = w_\ell(X_1) + w_\ell(X_2)$;
2. Para qualquer conjunto X, temos: $w_{\ell+1}(X) \geq w_\ell(X)$.

Além da capacidade externa, podemos também medir a **capacidade de borda** de um conjunto de vértices $X \subset V$, definida por:

$$brd(X) = cap(X, \overline{X}) = \sum_{v \in X} cap(v, \overline{X})$$

e todo vértice $v \in X$ tal que $cap(v, \overline{X}) > 0$ é chamado de **vértice de borda de** X.

Note que a capacidade externa de X se diferencia da capacidade de borda de X por considerar apenas as capacidades das arestas externas que ligam dois vértices que não pertencem a um mesmo nó em um nível ℓ, enquanto que a capacidade de borda considera sempre o corte entre X e $\overline{X}$, independente do nó ao qual os vértices destes conjuntos pertencem. Além disso, sendo H um nó do nível ℓ da árvore, se $X \subseteq V(H)$, temos:

$$brd(X) = w_\ell(X) + cap(X, V(H) \setminus X)$$

e para H, temos:

$$brd(H) = w_\ell(H).$$

Na próxima seção será visto de que forma os algoritmos usam a árvore de decomposição para construir um esquema de roteamento desinformado na rede de entrada.

5.2 Roteamento na árvore de decomposição

Os algoritmos desinformados apresentados nos capítulos seguintes usarão uma árvore de decomposição como base para seu roteamento de tal forma que o esquema de roteamento entre dois vértices u, v na rede G irá corresponder ao caminho entre as folhas que contêm os subgrafos $G[\{u\}]$ e $G[\{v\}]$ na árvore de decomposição.

Seja $\mathcal{T}_G$ uma árvore de decomposição de G e seja $P = \{H_1, \ldots, H_{|P|}\}$ o caminho feito nesta árvore entre as folhas $H_1 = G[\{u\}]$ e $H_{|P|} = G[\{v\}]$. Para estabelecer o esquema de roteamento entre os vértices u e v, o caminho P será representado na árvore por uma série de redistribuições de fluxo. A cada nó do caminho na árvore, cada vértice pertencente ao subgrafo deste nó receberá uma fração do fluxo que está sendo levado de u a v.

Este processo será chamado de **redistribuição** de fluxo e segue as seguintes diretivas: nas folhas da árvore, o procedimento é trivial, os vértices v e u receberão todo o fluxo. Em cada nó intermediário $H_i \in P$, com $1 < i < |P|$, o fluxo é dividido entre os vértices que o compõem. Esta redistribuição de fluxo se dá em duas etapas, descritas a seguir.

- Passagem de fluxo do nó H_{i-1}, antecessor de H_i em P, para o nó H_i: esta distribuição de fluxo entre os vértices de H_i dependerá se H_{i+1} é pai ou filho de H_i na árvore.
  - Caso H_{i-1} seja filho de H_i, o fluxo está subindo na árvore. Deste modo, é desejável que o fluxo seja enviado para vértices que possuam arestas de borda em H_i ou arestas que conectam vértices que pertençam a subconjuntos distintos de H_i. O fluxo será então redistribuído em função de $w_{\ell+1}(H_i)$: cada vértice $w \in H_i$ receberá uma fração $\frac{w_{\ell+1}(w)}{w_{\ell+1}(H_i)}$ do fluxo;
  - Caso H_{i-1} seja pai de H_i, o fluxo está descendo na árvore. Nesta situação, o fluxo deve ser distribuído entre os vértices que possuam arestas de borda em H_i, visto que o próximo conjunto do caminho é um superconjunto do atual. O fluxo neste caso será redistribuído em função de $w_\ell(H_i)$: cada vértice $w \in H_i$ receberá uma fração $\frac{w_\ell(w)}{w_\ell(H_i)}$;
- Preparação do fluxo para envio a H_{i+1}, nó sucessor de H_i no caminho P: Através de uma nova redistribuição de fluxo entre os vértices de H_i, envia-se o fluxo para vértices que possuem mais arestas que possam ser usadas para passar o fluxo ao próximo nó do caminho.
  - Caso H_{i+1} seja filho de H_i, o fluxo irá descer na árvore. Por argumentos análogos aos usados anteriormente, é interessante que o fluxo entre os vértices de H_i seja redistribuído em função de $w_{\ell+1}(H_i)$.

– Caso H_{i+1} seja pai de H_i, o fluxo irá subir na árvore. O fluxo será então redistribuído em função de $w_\ell(H_i)$.

Para ilustrar o processo de redistribuição, segue um exemplo.

Exemplo 5.1. *A figura 5.3 mostra o caminho P na árvore entre duas folhas $G[\{d\}]$ e $G[\{b\}]$.*

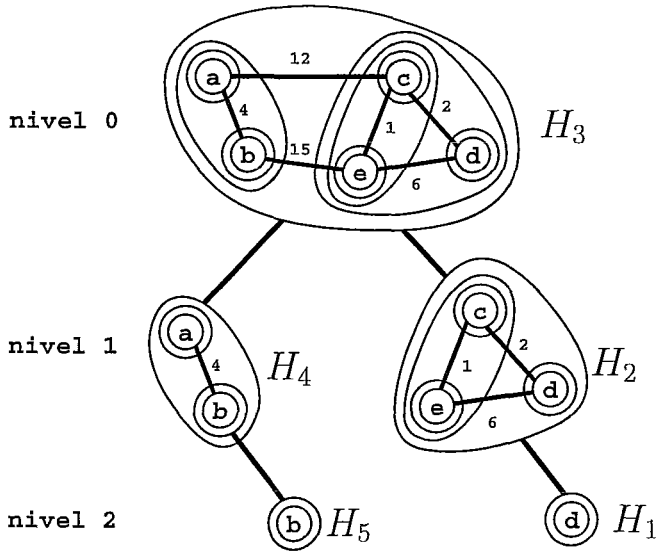


Figura 5.3: Caminho entre $G[\{d\}]$ e $G[\{b\}]$ na árvore de decomposição.

O exemplo mostrará a seqüência de redistribuições de fluxo feitas para transportar um fluxo unitário do vértice d ao vértice b usando o caminho $P = \{H_1, H_2, H_3, H_4, H_5\}$.

O caminho se inicia na folha H_1. *Neste caso o processo é trivial e, portanto, todo o fluxo permanece no vértice d, não havendo redistribuição de fluxo.*

O próximo subgrafo no caminho, H_2, *contém três vértices: e, d e c.* H_2 *pertence ao nível* 1 *e o fluxo está subindo na árvore, portanto a distribuição de fluxo entre os vértices de* H_2 *será feita em função de* w_2. *Como* $w_2(H_2) = w_2(e) + w_2(d) + w_2(c) = 12 + 15 + 2 \cdot 6 + 2 \cdot 2 = 43$, *temos:*

- $w_2(d) = 2 + 6 = 8$; *logo, i ficará com uma fração de* $\frac{8}{43}$ *do fluxo;*
- $w_2(e) = 15 + 6 = 21$; *j ficará com uma fração de* $\frac{21}{43}$ *do fluxo;*
- $w_2(c) = 12 + 2 = 14$; *será enviada a h uma fração de* $\frac{14}{43}$ *do fluxo.*

O processo de redistribuição segue com a preparação do fluxo para envio ao próximo nó da árvore, H_3. Como H_3 é pai de H_2 na árvore, a distribuição será feita em função de $w_1(H_2)$. Como $w_1(H_2) = w_1(d) + w_1(e) + w_1(f) = 12 + 15 = 27$, a distribuição será:

- $w_1(d) = 0$; *após esta redistribuição d não ficará com nenhuma fração do fluxo;*
- $w_1(e) = 15$; *e ficará com uma fração de $\frac{15}{27}$ do fluxo;*
- $w_1(c) = 12$; *c ficará com uma fração de $\frac{12}{27}$ do fluxo.*

Na passagem de fluxo entre H_2 e H_3, o fluxo será distribuído entre os vértices de H_3 em função de $w_1(H_3)$. Como $w_1(H_3) = w_1(a)+w_1(b)+w_1(c)+w_1(d)+w_1(e) = 2 \cdot (12) + 2 \cdot (15) = 54$, na raiz da árvore o fluxo ficará dividido entre os vértices da seguinte forma:

- $w_1(b) = 12$; *a ficará com uma fração de $\frac{12}{54}$ do fluxo;*
- $w_1(d) = 15$; *b ficará com uma fração de $\frac{15}{54}$ do fluxo;*
- $w_1(h) = 12$; *c ficará com uma fração de $\frac{12}{54}$ do fluxo;*
- $w_1(i) = 0$; *d não ficará com nenhuma fração do fluxo;*
- $w_1(j) = 15$; *e ficará com uma fração de $\frac{15}{54}$ do fluxo.*

Não será necessário redistribuir o fluxo novamente, pois o próximo nó no caminho é filho de H_3 e, portanto, a redistribuição a ser feita deveria também ser em função de $w_1(H_3)$.

No envio de fluxo entre H_3 e H_4, o fluxo está descendo na árvore. Portanto, os vértices de H_4 receberão fluxo em função de $w_1(H_4)$. Como $w_1(H_4) = w_1(a) + w_1(b) = 12 + 15 = 27$:

- $w_1(a) = 12$; *a recebe uma fração de $\frac{12}{27}$ do fluxo;*
- $w_1(b) = 15$ *b recebe uma fração de $\frac{15}{27}$ do fluxo.*

Para preparar o fluxo para envio a H_5, há a redistribuição em função de $w_2(H_4) = w_2(a) + w_2(b) = 15 + 12 + 2 \cdot 4 = 35$. Logo:

- $w_2(a) = 16$; *a recebe uma fração de $\frac{16}{35}$ do fluxo;*

- $w_2(b) = 19$, *b recebe uma fração de* $\frac{19}{27}$ *do fluxo.*

Em H_5, *todo o fluxo chegará ao vértice b. Portanto, um fluxo de valor* 1 *foi transportado de d a b na rede baseado no caminho entre suas folhas correspondentes na árvore de decomposição.*

A intuição que leva o algoritmo a fazer a redistribuição do fluxo é simples: se um vértice v pertencente a um nó H precisa se comunicar com algum vértice que não esteja em seu próprio nó, ele terá de enviar o fluxo através do corte que separa o conjunto de vértices de H dos demais vértices do grafo. Este fluxo será enviado de forma proporcional às capaciadades das arestas que os vértices possuem para fora do conjunto H.

Dado que o roteamento das demandas no algoritmo desinformado é feito proporcionalmente às capacidades de arestas de cortes, se no esquema de roteamento desinformado a carga relativa das arestas for igual a x, quando houver um pedido de roteamento cuja demanda aumente de x o congestionamento do algoritmo desinformado também aumentará proporcionamente de x, visto que o fluxo será escalonado pelas arestas do corte. Por outro lado, um algoritmo informado saberá que terá de rotear uma fração x a mais e, por isso, o congestionamento aumentará de no máximo x. Logo, nesta análise as *commodities* não são necessárias, dado que o procedimento será o mesmo para qualquer conjunto.

Definido como o caminho entre dois vértices na rede se baseará no caminho entre suas folhas na árvore de decomposição, ainda é preciso saber como o fluxo é transportado entre cada par de vértices e quais são os caminhos usados nesta redistribuição de fluxo.

Seja H_i um nó intermediário pertencente ao nível ℓ da árvore de decomposição. Em cada passagem de fluxo entre H_{i-1} (antecessor de H_i em um caminho entre duas folhas na árvore) e H_i, cada vértice $v \in V(H_i)$ receberá uma fração $\frac{w_\ell(v)}{w_\ell(H_i)}$ do fluxo (caso em que H_{i-1} é pai de H_i na árvore) ou uma fração $\frac{w_{\ell+1}(v)}{w_{\ell+1}(H_i)}$ do fluxo (caso em que H_{i-1} é filho de H_i na árvore). Vamos supor, sem perda de generalidade, que H_{i-1} é filho de H_i na árvore, para depois mostrar que o caso em que H_{i-1} é o pai é análogo a este.

Antes do envio de fluxo para os vértices de H_i, cada vértice $u \in V(H_{i-1})$ possui a fração $\frac{w_{\ell+1}(u)}{w_{\ell+1}(H_{i-1})} Q$, onde Q é o valor do fluxo que está sendo enviado entre as folhas

do caminho da árvore. Para que cada vértice v de H_i receba a fração $\frac{w_{\ell+1}(v)}{w_{\ell+1}(H_i)}Q$ na redistribuição entre H_{i-1} e H_i, cada vértice $u \in V(H_{i-1})$ enviará para cada vértice $v \in V(H_i)$ um fluxo de valor $\frac{w_{\ell+1}(u)}{w_{\ell+1}(H_{i-1})}\frac{w_{\ell+1}(v)}{w_{\ell+1}(H_i)}Q$. Deste modo, em cada redistribuição, o valor do fluxo que um vértice $v \in H_i$ receberá é igual a

$$\sum_{u \in V(H_{i-1})} \frac{w_{\ell+1}(u)}{w_{\ell+1}(H_{i-1})}\frac{w_{\ell+1}(v)}{w_{\ell+1}(H_i)}Q = \frac{w_{\ell+1}(H_{i-1})}{w_{\ell+1}(H_{i-1})}\frac{w_{\ell+1}(v)}{w_{\ell+1}(H_i)}Q = \frac{w_{\ell+1}(v)}{w_{\ell+1}(H_i)}Q$$

Logo, deste modo o vértice $v \in V(H_i)$ receberá uma fração $\frac{w_{\ell+1}(v)}{w_{\ell+1}(H_i)}$ do fluxo que entrará no nó H_i, como desejado.

No caso em que H_{i-1} é pai de H_i na árvore, temos:

$$\sum_{u \in V(H_{i-1})} \frac{w_\ell(u)}{w_\ell(H_{i-1})}\frac{w_\ell(v)}{w_\ell(H_i)}Q = \frac{w_\ell(H_i)}{w_\ell(H_{i-1})}\frac{w_\ell(v)}{w_\ell(H_i)}Q = \frac{w_\ell(v)}{w_\ell(H_i)}Q,$$

já que um vértice $u \in V(H_{i-1})$ possui uma fração $\frac{w_\ell(u)}{w_\ell(H_{i-1})}$ do fluxo e um vértice $v \in H_i$ precisa receber uma fração igual a $\frac{w_\ell(v)}{w_\ell(H_i)}$ deste mesmo fluxo.

A redistribuição que prepara para o envio de fluxo ao sucessor de H_i é análoga: se cada vértice v de H_i anteriormente detinha uma fração $\frac{w_\ell(v)}{w_\ell(H_i)}$ do fluxo, para que agora este fluxo seja redistribuído de modo que este mesmo vértice de H_i possua agora uma fração $\frac{w_{\ell+1}(v)}{w_{\ell+1}(H_i)}$, cada vértice $v \in V(H_i)$ deverá enviar a um vértice $u \in V(H_i)$ um fluxo de valor igual a $\frac{w_{\ell+1}(v)}{w_{\ell+1}(H_i)}\frac{w_\ell(u)}{w_\ell(H_i)}Q$. Somando a contribuição de todos os vértices no envio de fluxo para um vértice u, temos:

$$\sum_{v \in V(H_i)} \frac{w_{\ell+1}(v)}{w_{\ell+1}(H_i)}\frac{w_\ell(u)}{w_\ell(H_i)}Q = \frac{w_\ell(u)}{w_\ell(H_i)}Q.$$

Desta forma, u recebe uma fração $\frac{w_\ell(u)}{w_\ell(H_i)}$ do fluxo que entrou em H_i, como requerido. O caso em que a distribuição é de $w_{\ell+1}$ para w_ℓ é análogo.

Para definir os caminhos usados para transportar fluxo em cada redistribuição, isto é, para definir o esquema de roteamento dentro de cada nó intermediário, será usado um conjunto de CMCFs. Em cada nó intermediário da árvore será definido um CMCF cuja solução será usada na redistribuição para o transporte de fluxo entre pares de vértices pertencentes a este subgrafo. Neste CMCF haverá demandas entre cada par de vértices. A demanda entre um par de vértices u, v pertencentes a um nó H do nível ℓ da árvore é dada por:

$$d_H^\ell(u,v) = \frac{w_{\ell+1}(u) \cdot w_{\ell+1}(v)}{w_{\ell+1}(H)}. \tag{5.1}$$

Antes de prosseguir, serão feitas algumas observações sobre a escolha da fórmula da demanda deste CMCF, correlacionando-a com a redistribuição de fluxo que se deseja fazer no nó H.

No CMCF existirá uma demanda entre todos os pares de vértices igual a:

$$\begin{aligned}\sum_{v\in V}\sum_{u\in V} d_H(u,v) &= \sum_{v\in V}\sum_{u\in V}\frac{w_{\ell+1}(u)\cdot w_{\ell+1}(v)}{w_{\ell+1}(H)} \\ &= \frac{w_{\ell+1}(H)}{w_{\ell+1}(H)}\sum_{v\in V} w_{\ell+1}(v) \\ &= \frac{w_{\ell+1}(H)\cdot w_{\ell+1}(H)}{w_{\ell+1}(H)} \\ &= w_{\ell+1}(H).\end{aligned} \tag{5.2}$$

É importante ressaltar que $w_{\ell+1}(H)$ é um limite superior para a quantidade de fluxo que pode entrar em um nó intermediário H durante a passagem de fluxo no caminho entre duas folhas da árvore de decomposição $\mathcal{T}_{\mathcal{D}_G}$; portanto, a solução do CMCF fornecerá um esquema de roteamento que produz a maior fração de vazão possível ainda que todas as arestas pelas quais pode entrar fluxo em H sejam saturadas, o que representa a maior quantidade de fluxo possível entrando neste nó.

Outra propriedade interessante é que o roteamento da demanda especificada entre cada par de vértices no CMCF produz a redistribuição de fluxo da forma desejada. Na redistribuição, o vértice v deverá receber no máximo uma fração de $\frac{w_{\ell+1}(v)}{w_{\ell+1}(H)}$ do fluxo total, isto é, o vértice v terá de receber esta fração do fluxo de cada vértice de H. Usando o máximo de fluxo possível entrando em H, em particular, para um vértice u, o vértice v terá de receber uma fração de fluxo igual a $w_{\ell+1}(u)\cdot\frac{w_{\ell+1}(v)}{w_{\ell+1}(H)}$, que é exatamente a fórmula da demanda entre u e v no CMCF.

O esquema de roteamento desinformado entre um par de vértices u, v usando a árvore de decomposição pode então ser resumido em dois passos básicos:

- Dada uma árvore de decomposição $\mathcal{T}_{\mathcal{D}_G}$, identifica-se o caminho $P = \{H_1, \ldots, H_{|P|}\}$ entre as duas folhas H_1 e $H_{|P|}$, correspondentes respectivamente ao grafos triviais $G[\{u\}]$ e $G[\{v\}]$;

- No grafo, o caminho P feito na árvore será representado por redistribuições de fluxo entre vértices. Na etapa da redistribuição que representa a passagem de fluxo entre dois nós H_{i-1} e H_i, $1 < i < |P|$, os caminhos usados para

transportar fluxo entre pares de vértices do grafo são calculados na solução do problema CMCF definido em H_i, caso este seja pai de H_{i-1} na árvore; caso contrário, os caminhos usados são os calculados na solução do CMCF de H_{i-1}. Já na etapa da redistribuição que prepara o envio de fluxo a H_{i+1}, os caminhos usados para transportar fluxo entre pares de vértices de H_i serão dados pela solução do CMCF definido neste subgrafo.

Para calcular o congestionamento produzido pelo esquema de roteamento descrito neste capítulo, serão usados o lema 5.2 e o teorema 5.3.

Lema 5.2. *Seja z_{H_i} a fração de vazão obtida no problema CMCF estabelecido no nó intermediário H_i. As redistribuições de fluxo feitas entre os vértices de H_i para representar o caminho da árvore produzem congestionamento igual a $\frac{1}{z_{H_i}}$ quando roteadas pelos caminhos calculados na solução do CMCF deste subgrafo.*

Demonstração. Para demonstrar o lema basta verificar que em qualquer redistribuição o valor do fluxo que precisa ser roteado entre cada par de vértices é menor ou igual à demanda entre estes mesmos vértices no CMCF.

Quando o nó H_i do nível ℓ da árvore recebe fluxo de H_{i-1}, seu filho na árvore, é preciso passar de uma distribuição em que cada vértice $v \in V(H_{i-1})$ possui uma fração $\frac{w_{\ell+1}(v)}{w_{\ell+1}(H_{i-1})}$ para uma distribuição em que cada vértice $u \in H_i$ receberá uma fração $\frac{w_\ell(u)}{w_\ell(H_i)}$. Além disso, podemos perceber que o valor do fluxo que pode passar entre H_{i-1} e H_i é limitado pela soma das capacidades das arestas externas de H_{i-1}, isto é, $w_{\ell+1}(H_{i-1})$. Portanto, a quantidade de fluxo que passará de um vértice u para um vértice v será de

$$\frac{w_{\ell+1}(u)}{w_{\ell+1}(H_i)} \frac{w_{\ell+1}(v)}{w_{\ell+1}(H_{i-1})} w_{\ell+1}(H_{i-1}) = \frac{w_{\ell+1}(u) \cdot w_{\ell+1}(v)}{w_{\ell+1}(H_i)}.$$

Voltando à equação (5.1), podemos perceber que o fluxo roteado entre dois vértices é igual à demanda entre os mesmos no CMCF definido no nó H_i. Esta quantidade, portanto, poderá ser roteada pelos caminhos do CMCF produzindo congestionamento igual a $\frac{1}{z_{H_i}}$. Note que esta última equação vale também para o caso em que o nó H_{i+1} é filho de H_i na árvore, visto que passamos de uma distribuição em função de $w_{\ell+1}(H_i)$ para uma distribuição em função de $w_{\ell+1}(H_{i+1})$ e são usados os caminhos calculados na solução do CMCF definido em H_i.

No caso em que há em H_i a preparação para o envio ao próximo nó do caminho na árvore, haverá uma redistribuição de fluxo em função de $w_\ell(H_i)$ para uma distribuição em função de $w_{\ell+1}(H_i)$ (ou vice-versa). Neste caso, o valor do fluxo é limitado por $w_\ell(H_i)$, visto que para haver tal redistribuição o fluxo deverá subir na árvore ou ter vindo do pai de H_i na árvore. O fluxo entre $u, v \in H_i$ será dado por

$$\frac{w_{\ell+1}(u)}{w_{\ell+1}(H_i)} \frac{w_\ell(v)}{w_\ell(H_i)} w_\ell(H_i) = \frac{w_{\ell+1}(u) \cdot w_\ell(v)}{w_{\ell+1}(H_i)}.$$

Visto que $w_\ell(u) \leq w_{\ell+1}(u)$, o valor deste fluxo é menor do que a demanda entre estes dois vértices no CMCF definido em H_i.

A subida de fluxo a partir de H_i, isto é, a passagem de fluxo entre H_i e seu pai na árvore, é feita usando os caminhos da solução do CMCF definido no nó pai. Logo, esta operação é análoga ao argumento feito anteriormente para H_{i-1} e H_i, mudando somente o nível dos nós.

Visto que em todos os casos o fluxo a ser roteado é menor do que a demanda do CMCF, caso estes caminhos sejam usados para este roteamento o congestionamento resultante será de no máximo $\frac{1}{z_{H_i}}$. □

O teorema 5.3 calcula o congestionamento do algoritmo de roteamento.

Teorema 5.3. *Dada uma decomposição hierárquica de altura h, o algoritmo de roteamento apresentado possui razão de competitividade $O(\frac{h}{z_{\min}})$, onde $z_{\min}$ é a menor de todas as frações de vazões obtidas nas soluções ótimas dos CMCFs definidos nos nós da árvore de decomposição, isto é, $z_{\min} = \min_{H_i \in \mathcal{T}_{\mathcal{D}_G}} \{z_{H_i}\}$.*

Demonstração. Pelo lema 5.2, as redistribuições de fluxo dentro de um nó H_i produzem congestionamento $\frac{1}{z_{H_i}}$, onde z_{H_i} é a fração de vazão obtida na solução do problema CMCF definido neste subgrafo. Como esta solução é restrita às arestas e vértices deste subgrafo, este congestionamento atinge somente as arestas que estão dentro deste nó. Precisamos então calcular a quantos subgrafos diferentes uma aresta pode pertencer.

Como dito na seção anterior, podemos ver a relação entre um nó H_i e seus filhos na árvore como uma partição em vértices. Deste modo, temos duas alternativas para uma aresta e do grafo na partição de H_i em subgrafos:

e **foi uma aresta cortada em** H_i**:** desde modo, e não participará de nenhuma redistribuição de fluxo em nós descendentes de H_i;

e **não foi aresta cortada em** H_i**:** neste caso temos que e pertence a um único subconjunto de H_i, visto que estes constituem uma partição em vértices.

Pelo primeiro item, concluímos que uma aresta participa de todas as redistribuições de fluxo dos subgrafos aos quais pertence somente até o nível em que ela é cortada. Pelo segundo item, podemos concluir que uma aresta e faz parte de apenas um ramo da árvore, ou seja, ela faz parte – não sendo aresta externa – de no máximo um nó em cada nível da árvore.

Juntando as duas idéias, temos que em uma árvore de altura h uma aresta pode participar de no máximo h redistribuições de fluxo. Portanto, denotando por $z_{min} = \min_{H_i \in \mathcal{D}_G}\{z_{H_i}\}$, concluímos que o algoritmo produz congestionamento $O(\frac{h}{z_{min}})$. □

Capítulo 6

O algoritmo B.K.R.

Neste capítulo o algoritmo proposto por Bienkowski, Korzeniowski e Räcke [14] para redes não-direcionadas com capacidades nas arestas é descrito e analisado. Publicado em 2003, este método utiliza uma árvore de decomposição para, da forma descrita no capítulo 5, obter um esquema de roteamento desinformado na rede.

Este capítulo é dedicado ao algoritmo usado por estes autores para construir uma árvore de decomposição com tais propriedades que o roteamento desinformado obtido possua razão de competitividade $O(\log^4 n)$, onde n, como sempre, denota no número de vértices da rede, e que esta construção seja feita em tempo polinomial em n e em $c_{\max} = \max_{e \in E} cap(e)$ sendo, portanto, **pseudopolinomial**.

O algoritmo constrói a árvore a partir de sua raiz. Em cada nó da árvore, seu conjunto de filhos é construído por partições sucessivas baseadas em cortes mais esparsos aproximados, até que seja obtida uma partição na qual o CMCF descrito na seção 5.2 alcança certa fração de vazão desejada.

O algoritmo descrito neste capítulo assume que a capacidade das arestas do grafo está normalizada, isto é, a menor capacidade positiva dentre todas as arestas do grafo é igual a 1. Uma normalização destas capacidades é facilmente obtida dividindo todas as capacidades do grafo pela menor delas.

A seção 6.1 apresenta o algoritmo de construção da árvore e na seção 6.2 é feita a análise deste método, demonstrando sua razão de competitividade e verificando que ele termina em tempo polinomial.

6.1 Obtendo uma boa árvore de decomposição

Para que o algoritmo de roteamento desinformado apresentado possua uma boa razão de competitividade é preciso que a árvore de decomposição tenha uma altura h pequena e que, ao mesmo tempo, o problema CMCF de cada nó tenha fração de vazão grande. Esta seção é dedicada à descrição do método usado por B.K.R. para construir, para uma rede qualquer, uma árvore de decomposição que possui altura $O(\log n)$ e z_{min} é $\Omega(\frac{1}{\alpha^2 \cdot \log n})$, onde α é o fator de aproximação do algoritmo para obter um corte mais esparso aproximado.

O algoritmo 2 descreve o método já explicado na seção 5.1 para construir uma árvore de decomposição a partir de um grafo H dado como entrada. A chamada externa para a construção da árvore de decomposição $\mathcal{T}_{\mathcal{D}_G}$ é `Constrói Árvore`$(G, 0)$, onde G é a rede do problema, e a função `Particiona` – que será detalhada a seguir – retorna uma partição do grafo no qual ela é aplicada.

Algoritmo 2 Constrói Árvore(H, ℓ)

$\mathcal{P}_H :=$ Particiona(H, ℓ);
$Fila := \emptyset$;
para todo $H_f \in \mathcal{P}_H$ **faça**
  Insere H_f na $Fila$;
enquanto $Fila \neq \emptyset$ **faça**
  $H_i :=$ primeiro elemento da $Fila$;
  $\ell_{H_i} :=$ altura de H_i na árvore;
  $\mathcal{P}_{H_i} :=$ Particiona$(H_i, \ell_{H_i} + 1)$;
  $\mathcal{P}_H := \mathcal{P}_H \cup \mathcal{P}_{H_i}$;
  para todo $H_f \in \mathcal{P}_{H_i}$ **faça**
    Insere H_f na $Fila$;
  $\mathcal{P}_H := \mathcal{P}_H \cup \{H_i\}$;
  Retira primeiro elemento da $Fila$;
retorna $\mathcal{P}_H$;

Antes de descrever a subrotina `Particiona`, serão dadas algumas idéias gerais sobre seu funcionamento.

Na seção anterior, foi visto que a demanda estipulada para o problema CMCF em

cada nó intermediário é dada pela equação (5.1). Sendo H um nó da decomposição no nível ℓ e sendo a soma de todas as demandas entre os pares de vértices de H igual à função $w_{\ell+1}(H)$ (como visto na equação (5.2)), percebe-se que a fração de vazão do CMCF é diretamente influenciada pela partição em subgrafos escolhida para H, já que a demanda é especificada por $w_{\ell+1}$.

Sabendo que as arestas cortadas no nível ℓ têm seu valor somado duas vezes na função $w_{\ell+1}$, em um primeiro momento pode-se pensar em manter subgrafos grandes – e, portanto, com poucas arestas cortadas – na partição de H, diminuindo assim o valor da função $w_{\ell+1}(H)$; no entanto, é necessário lembrar que fazer uma partição deste tipo em cada nó intermediário pode levar a uma árvore de altura grande, o que afeta negativamente a razão de competitividade do algoritmo.

Por isto, o algoritmo de particionamento de um nó H inicia com a partição de H contendo somente subgrafos triviais: $\mathcal{P}_H = \{\{v\} \mid v \in H\}$. Desta forma, a demanda é a maior possível no início do processo, visto que a função $w_{\ell+1}$ alcança seu valor máximo. Caso a fração de vazão do problema CMCF no nó não alcance um certo valor pré-determinado $z_{inf} = \Omega(\frac{1}{\alpha^2 \cdot \log n})$, H é reparticionado de forma que $w_{\ell+1}(H)$ decresça. Este processo de reparticionamento é repetido a cada iteração do algoritmo, até que uma partição de H com fração de vazão z_H maior ou igual a z_{inf} seja obtida.

Resta então saber como, a cada iteração do algoritmo, efetuar o reparticionamento de forma a diminuir o valor da função $w_{\ell+1}$.

Neste ponto, deve-se lembrar que um corte mais esparso fornece um conjunto de arestas que possui carga relativa grande, isto é, este corte possui uma capacidade muito pequena em relação à soma das demandas entre os pares de vértices por ele separados. Um dos motivos da grande utilização destas arestas é que a demanda entre quaisquer dois vértices que pertençam a um nó H pode ser livremente roteada por suas arestas; por isso, pode acontecer que as arestas do corte podem estar sendo utilizadas para rotear demanda entre dois vértices que pertencem ao mesmo lado do corte.

Recordando de como o roteamento na árvore de decomposição funciona, percebe-se que só há demanda entre pares de vértices se eles possuem arestas ligando subconjuntos diferentes ou arestas para fora do nó H. Como estas últimas estarão sempre

invariantes dentro do nó H, a maneira de fazer com que menos demanda com origem e destino de um mesmo lado do corte utilize suas arestas é diminuindo o número de subconjuntos em um dos lados do corte. Deste modo, a demanda entre os dois lados deste corte no CMCF no nó H diminui e, portanto, o valor do fluxo passando por ele também diminui, aumentando assim as chances de, com esta nova partição, uma fração de vazão maior do que a anterior ser obtida.

Seja (A, B) um corte mais esparso aproximado para as demandas estabelecidas no CMCF definido no nó H. Para reduzir a demanda entre os dois lados do corte, é necessário diminuir $w_{\ell+1}(A)$ ou $w_{\ell+1}(B)$. Note que como $w_{\ell+1}(H) = w_{\ell+1}(A) + w_{\ell+1}(B)$, ao diminuir o valor de uma destas duas parcelas sem aumentar a outra, a cada iteração do algoritmo o valor de $w_{\ell+1}(H)$ será reduzido.

No entanto, o corte (A, B) pode separar vértices que pertencem a um mesmo subgrafo da partição atual de H. Não é desejável desfazer subgrafos que já estejam nesta partição, pois o fato de terem formado um subgrafo em alguma iteração anterior indica que possuem arestas com capacidade razoável entre todos os seus vértices.

Para superar este problema, o algoritmo efetua um **arredondamento do corte**, para que ele seja composto somente de subgrafos que possuam todos os seus vértices em um lado do corte. Os subgrafos da partição $\mathcal{P}_H$ são particionados em dois conjuntos: sendo A o conjunto de menor tamanho no corte (A, B), um subgrafo H_i de $\mathcal{P}_H$ pertencerá ao conjunto I_L caso possua pelo menos $\frac{3}{4}$ de seus vértices pertencentes a A; caso contrário, H_i pertence ao conjunto I_S.

O arredondamento de A produz um subgrafo A^* tal que

$$A^* = \cup_{H_i \in I_L} H[V(H_i)]$$

isto é, o subgrafo A^* é o subgrafo induzido pelo conjunto de vértices $\{A \cup_{H_i \in I_L} V(H_i)\} \setminus \{\cup_{H_i \in I_S} V(H_i)\}$; este conjunto de vértices é produzido pelos elementos do conjunto A adicionados dos demais vértices dos subgrafos do conjunto I_L que não pertecem a A, menos os vértices que pertencem a subgrafos do conjunto I_S.

Neste ponto do algoritmo, vemos o subgrafo A^* naturalmente particionado: $\mathcal{P}_{A^*} = I_L$. Esta partição será modificada através da função `Rearranjo`, que será estudada a seguir.

O algoritmo 3 contém os passos do particionamento de um nó H.

Algoritmo 3 Particiona(H, ℓ)

$\mathcal{P}_H := \{\{v\} \mid v \in V(H)\}$;

$d^e_H ll :=$ demanda do CMCF definido em H;

$F := CMCF(H, \mathcal{P}_H, d^\ell_H)$;

$z_H := \frac{1}{congestionamento(H,F)}$; {$z_H$ é a fração de vazão do CMCF em H}

enquanto $z_H < z_{inf}$ **faça**

$(A, B) :=$ Corte Esparso Aproximado(H, d^ℓ_H); {com $|A| \leq |B|$}

$A^* :=$Arredondamento$(A, \mathcal{P}_H)$;

para todo $H_f \in \mathcal{P}_{A^*}$ **faça**

$\mathcal{P}_H := \mathcal{P}_H \setminus \{H_f\}$;

$\mathcal{P}_H := \mathcal{P}_H \cup$ Rearranjo(A^*, ℓ); {Modifica partição de A^*}

retorna $\mathcal{P}_H$;

A função `Rearranjo` é responsável por refazer a partição de A^* de forma a diminuir o valor de $w_{\ell+1}$. A maior redução de demanda entre os dois lados do corte mais esparso calculado no algoritmo 3 seria ao manter um subgrafo contendo todos os vértices de A^*. No entanto, é necessário perceber que nem todo subconjunto de G consegue alcançar a fração de vazão z_{inf}. Pode-se dar como exemplo simples deste caso um subconjunto que possuas muitas arestas externas, porém as arestas de dentro do subgrafo tenham capacidade pequena em relação às primeiras. Pela demanda do CMCF (que é proporcional a $w_{\ell+1}$ e, por isso, proporcional às arestas externas), haverá muita demanda para rotear, mas pouca capacidade dentro do subgrafo, o que resultará em arestas com carga relativa alta.

A principal atribuição do `Rearranjo` será, portanto, além de produzir uma partição de A^* que diminua a função $w_{\ell+1}(H)$ na função `Particiona`, é produzir subconjuntos que consigam alcançar a fração de vazão.

Esta função recebe como entrada o subgrafo A^*, que é um subgrafo de um nó do nível ℓ da árvore de decomposição produzido na subrotina `Arredondamento` (evocada dentro do algoritmo 3), e retorna uma partição deste que possui as seguintes propriedades:

1. Para cada $R_i \in \mathcal{P}_{A^*}$, $|R_i| \leq \frac{2}{3}|A^*|$;

2. Cada $R_i \in \mathcal{P}_{A^*}$ possui uma propriedade invariante, chamada de precondição

de vazão:

Um nó S do nível ℓ possui a **precondição de vazão** se, para todo subconjunto de vértices $U \subset S$ tal que $|U| \leq \frac{3}{4}|S|$, temos:

$$\lambda \cdot cap(U, S \setminus U) \geq w_\ell(U), \tag{6.1}$$

onde $\lambda = O(\alpha \cdot \log n)$.

Logo, como A^*, que é parte de um nó do nível ℓ, está sendo particionado, os conjuntos R_i que serão produzidos no algoritmo `Rearranjo` pertencem ao nível $\ell+1$ da árvore. Por isso, para cada $R_i \in \mathcal{P}_{A^*}$ e para cada $U \subset R_i$ com $|U| \leq \frac{3}{4}|R_i|$, vale que:

$$\lambda \cdot cap(U, R_i \setminus U) \geq w_{\ell+1}(U). \tag{6.2}$$

Note que precondição assegura que não há um subconjunto de vértices que esteja mal-conectado (muita demanda a ser roteada pelo subconjunto e pouca capacidade de suas arestas).

O primeiro passo da função `Rearranjo` é desfazer a partição que A^* possuía, substituindo-a por um único conjunto contendo o subgrafo inteiro.

Como o objetivo é particionar de forma a obter subconjuntos que possuem a propriedade de precondição de vazão, a cada iteração verifica-se se cada conjunto $R_i \in \mathcal{P}_{A^*}$ já possui esta propriedade. Caso não possua, ele é particionado, usando dois lados de um corte mais esparso aproximado, em dois subgrafos A_{R_i} e B_{R_i} e repete-se a verificação nestes novos subgrafos, até que todos os subgrafos da partição $\mathcal{P}_{A^*}$ tenham a precondição de vazão.

A verificação de que o subgrafo R_i já possui a precondição de vazão é feita em cada subconjunto R_i utilizando uma instância do problema CMCF que possui demandas entre cada par de vértices iguais a

$$d_{R_i}(u, v) = \frac{w_{\ell+1}(u)}{|R_i|}.$$

Seja (A_{R_i}, B_{R_i}) um corte mais esparso aproximado em R_i associado às demandas especificadas no CMCF definido neste subgrafo. A esparsidade ψ deste corte é dada

por:

$$\begin{aligned}
\psi &= \frac{cap(A_{R_i}, B_{R_i})}{d_{K_{R_i}}(A_{R_i}, B_{R_i})} \\
&= \frac{cap(A_{R_i}, B_{R_i})}{\sum_{v \in A_{R_i}} \sum_{u \in B_{R_i}} \frac{w_{\ell+1}(u)}{|R_i|} + \sum_{v \in B_{R_i}} \sum_{u \in A_{R_i}} \frac{w_{\ell+1}(v)}{|R_i|}} \\
&= \frac{cap(A_{R_i}, B_{R_i})}{\sum_{v \in A_{R_i}} \frac{w_{\ell+1}(B_{R_i})}{|R_i|} + \sum_{v \in B_{R_i}} \frac{w_{\ell+1}(A_{R_i})}{|R_i|}} \\
&= \frac{cap(A_{R_i}, B_{R_i})}{w_{\ell+1}(B_{R_i}) \cdot \frac{|A_{R_i}|}{|R_i|} + w_{\ell+1}(A_{R_i}) \cdot \frac{|B_{R_i}|}{|R_i|}}. \qquad (6.3)
\end{aligned}$$

A cada iteração, o algoritmo verifica se, para todo conjunto $R_i \in \mathcal{P}_{A^*}$, a esparsidade ψ do corte mais esparso aproximado associado às demandas do problema CMCF do subgrafo R_i possui esparsidade maior do que $\Lambda = \frac{4\alpha}{\lambda}$. Adiante, na análise do método de construção da árvore, será visto que $\psi > \Lambda$ é uma condição suficiente para que o *cluster* R_i possua a precondição de vazão. Se $\psi > \Lambda$, o subgrafo R_i já possui a precondição de vazão, e pode permanecer na partição retornada pela função `Rearranjo`. Caso contrário, se $\psi \leq \Lambda$, este subgrafo é dividido em dois conjuntos: A_{R_i} e B_{R_i}, que representam os dois lados do corte mais esparso aproximado calculado.

O algoritmo 4 descreve a função `Rearranjo`.

6.2 Análise do algoritmo

Esta seção analisa a razão de competitividade que o algoritmo de roteamento desinformado alcança utilizando a árvore construída através do algoritmo apresentado, além de verificar que esta construção de fato é feita em tempo polinomial.

A análise será iniciada pela função `Rearranjo` (algoritmo 4). É preciso relembrar que esta função tem por objetivo, tomando como entrada um subgrafo A^*, construir uma partição de A^* em subgrafos R_i que possua duas propriedades:

- Cada subgrafo R_i possui a precondição de vazão;
- A partição $\mathcal{P}_{A^*}$ retornada pela função `Rearranjo` à função `Particiona` diminuirá a demanda no CMCF do nó H passado como parâmetro a esta última.

Algoritmo 4 Rearranjo(A^*, ℓ)

$\mathcal{P}_{A^*} := \{A^*\}$;

$Fila := \{A^*\}$;

enquanto $Fila \neq \emptyset$ **faça**

$R_i :=$ primeiro elemento da $Fila$;

$d_{R_i} :=$ demanda do CMCF em R_i;

$(A_{R_i}, B_{R_i}) :=$ Corte Esparso Aproximado(R_i, d_{R_i});

se $esp((A_{R_i}, B_{R_i})) \leq \frac{4\alpha}{\lambda}$ **então**

{Retira R_i e adiciona $\{A_{R_i}\}$ e $\{B_{R_i}\}$ à partição $\mathcal{P}_{A^*}$}

$\mathcal{P}_{A^*} := \mathcal{P}_{A^*} \setminus \{R_i\}$;

$\mathcal{P}_{A^*} := \mathcal{P}_{A^*} \cup \{A_{R_i}\} \cup \{B_{R_i}\}$

Insere $\{A_{R_i}\}$ e $\{B_{R_i}\}$ na $Fila$;

Retira primeiro elemento da $Fila$;

retorna $\mathcal{P}_{A^*}$;

Para provar que de fato o algoritmo retorna uma partição com estas características, seguem os lemas abaixo.

Lema 6.1. *A partição $\mathcal{P}_{A^*}$ retornada pelo algoritmo 4 contém somente subgrafos que possuam a precondição de vazão.*

Demonstração. Suponha, por contradição, que existe um subgrafo $R_i \in \mathcal{P}_{A^*}$ que não possua a precondição de vazão. Então existe um subgrafo $U \subset R_i$ tal que $|U| \leq \frac{3}{4}|R_i|$ para o qual temos:

$$\lambda\, cap(U, R_i \setminus U) < w_{\ell+1}(U).$$

Mas, como $|U| \leq \frac{3}{4}|R_i|$, vale que $|R_i \setminus U| \geq \frac{1}{4}|R_i|$. Portanto, $4\frac{|R_i \setminus U|}{|R_i|} \geq 1$, e temos

$$\lambda\, cap(U, R_i \setminus U) < w_{\ell+1}(U) \leq 4\frac{|R_i \setminus U|}{|R_i|} w_{\ell+1}(U).$$

Logicamente, $\frac{|U|}{|R_i|} > 0$ e $w_{\ell+1}(R_i \setminus U) > 0$, então:

$$\lambda\, cap(U, R_i \setminus U) \leq 4\left(\frac{|R_i \setminus U|}{|R_i|} w_{\ell+1}(U) + \frac{|U|}{|R_i|} w_{\ell+1}(R_i \setminus U)\right).$$

Mas podemos perceber que $\frac{|R_i \setminus U|}{|R_i|} w_{\ell+1}(U) + \frac{|U|}{|R_i|} w_{\ell+1}(R_i \setminus U) = d_{R_i}(U, R_i \setminus U)$.

Juntando esta informação com a equação anterior, chegamos a:

$$\begin{aligned} \lambda\, cap(U, R_i \setminus U) &\leq 4\left(\frac{|R_i \setminus U|}{|R_i|}\, w_{\ell+1}(U) + \frac{|U|}{|R_i|}\, w_{\ell+1}(R_i \setminus U)\right) \\ &\leq 4\, d_{R_i}(U, R_i \setminus U), \end{aligned}$$

o que implica que

$$esp(U, R_i \setminus U) \leq \frac{4}{\lambda}.$$

Como a esparsidade do corte $(U, R_i \setminus U)$ – corte que liga um subconjunto U, que não possui a precondição de vazão, ao resto do subgrafo R_i – é de no máximo $\frac{4}{\lambda}$, podemos concluir que, se todos os cortes de R_i possuem esparsidade maior que este valor, certamente R_i possui a precondição de vazão.

Conhecendo um corte mais esparso em R_i, precisamos somente comparar o valor desta esparsidade com $\frac{4}{\lambda}$. No entanto, como visto na seção 2.4, um corte mais esparso não pode ser calculado em tempo polinomial; logo, como queremos que o algoritmo de construção da árvore seja polinomial, não poderemos encontrar este corte de forma exata.

Usaremos então uma condição suficiente para testar se a esparsidade de um corte mínimo está dentro do limite desejado: pela equação (2.4), um algoritmo α-aproximativo para encontrar um corte mais esparso aproximado retorna, para R_i, um corte cuja esparsidade ψ é tal que $\psi = z^*\,\alpha \leq \alpha\, esp(\mathcal{C})$, onde $\mathcal{C}$ é um corte mais esparso e z^* é a fração de vazão ótima do CMCF definido no subgrafo R_i. Logo, se R_i não possui a precondição de vazão, temos:

$$\begin{aligned} \psi &\leq \alpha\, \mathcal{C} \\ &\leq \alpha\, \tfrac{4}{\lambda} = \Lambda. \end{aligned}$$

No entanto, se a esparsidade de um corte mais aproximado (A_{R_i}, B_{R_i}) é menor ou igual a Λ, o subgrafo R_i não estará presente na partição final, visto que será substituído por $\{A_{R_i}\}$ e $\{B_{R_i}\}$ durante o algoritmo. Portanto, todos os subgrafos da partição $\mathcal{P}_R$ possuem a precondição de vazão. □

Para provar que a modificação da partição feita a cada iteração da função `Particiona` de fato diminui a demanda do CMCF definido no nó H, vamos analisar o que acontece com o valor de $w_{\ell+1}(H)$ quando a função `Rearranjo` refaz a partição de A^*.

Em uma iteração do algoritmo, após encontrar um corte mais esparso e arrendondá-lo, obtendo o conjunto A^*, dois passos são executados:

- Os conjuntos de A^* são removidos da partição $\mathcal{P}_H$; com isto, $w_{\ell+1}(H)$ decresce de $w_{\ell+1}(A^*)$. Como estamos tratando da partição anterior à aplicação do algoritmo `Rearranjo`, para evitar ambigüidade chamaremos a partir de agora este valor de $w_{\ell+1}(A^*_{ant})$;

- A partição de A^* é refeita através da função `Rearranjo`; neste passo, o valor da função $w_{\ell+1}(H)$ é acrescido de $w_{\ell+1}(A^*)$, que representa a soma das capacidades das arestas que ligam vértices pertencentes a subgrafos distintos na nova partição de A^*. Novamente, para evitar ambigüidade, usaremos $w_{\ell+1}(A^*_{novo})$ para nos referirmos ao valor de $w_{\ell+1}$ da partição de A^* após a aplicação do algoritmo 4.

Após uma iteração do algoritmo, temos:

$$\Delta W = -w_{\ell+1}(A^*_{ant}) + w_{\ell+1}(A^*_{novo}), \tag{6.4}$$

onde ΔW é a variação que $w_{\ell+1}(H)$ sofre em uma iteração do algoritmo.

Logo, precisamos mostrar a relação que existe entre $w_{\ell+1}(A^*_{ant})$ e $w_{\ell+1}(A^*_{novo})$ para demonstrar que o algoritmo diminui o valor de $w_{\ell+1}(H)$. Para isto, usaremos os lemas 6.2 e 6.3, descritos a seguir.

Lema 6.2. *A partição $\mathcal{P}_R$ retornada pelo algoritmo 4 é tal que $\sum_{R_i \in \mathcal{P}_R} w_{\ell+1}(R_i) \leq 2\, brd(R)$.*

Demonstração. A demonstração será baseada em uma análise amortizada da capacidade que um novo corte (A_{R_i}, B_{R_i}) pode ter em relação à soma da capacidade das arestas de borda do subgrafo R_i.

Em um subgrafo R_i que foi dividido na iteração, seja (A_{R_i}, B_{R_i}) o corte mais esparso aproximado associado às suas demandas. Dividiremos as arestas de borda de R_i em dois conjuntos: E_a e E_b. Uma aresta de borda e pertence a E_a se o vértice de sua extremidade que está no conjunto R_i pertence ao lado A_{R_i} do corte mais esparso aproximado (A_{R_i}, B_{R_i}); caso contrário, e pertence a E_b.

Na análise amortizada, diremos que as arestas do corte (A_{R_i}, B_{R_i}) geram um *custo* para as arestas de borda de R_i. Nosso objetivo será mostrar que as arestas de

borda pagam todo o custo gerado pelas arestas do corte (A_{R_i}, B_{R_i}) e que, mesmo pagando todo este custo, qualquer aresta da borda de R_i não é onerada a pagar mais do que uma fração de sua capacidade.

Vamos definir então a função *custo*. Uma aresta e_c pertencente a um corte mais esparso (A_{R_i}, B_{R_i}) gera um custo para uma aresta de borda $e_a \in E_a$ com um valor igual a:

$$custo(e_c, e_a) = 2\Lambda \frac{cap(e_a)\frac{|B_{R_i}|}{|R_i|}}{cap(A_{R_i}, B_{R_i})} cap(e_c).$$

Caso a aresta de borda e_b pertença a E_b, o valor do custo é de:

$$custo(e_c, e_b) = 2\Lambda \frac{cap(e_b)\frac{|A_{R_i}|}{|R_i|}}{cap(A_{R_i}, B_{R_i})} cap(e_c).$$

A divisão de R_i em dois conjuntos A_{R_i} e B_{R_i} gera na função $w_{\ell+1}(A^*)$ um acréscimo de $2\sum_{e_c \in (A_{R_i}, B_{R_i})} cap(e_c)$. Note que as arestas do corte são contadas duas vezes porque cada aresta (u, v) pertencente a ele será somada em $w_{\ell+1}(u)$ e em $w_{\ell+1}(v)$. Portanto, se cada aresta de borda e pagar o dobro da capacidade de uma aresta e_c do corte, conseguiremos pagar por toda capacidade do corte.

Note que a função *custo* alcança este objetivo:

$$\begin{aligned}
\sum_{e \in E_a \cup E_b} custo(e_c, e) &= \sum_{e_a \in E_a} custo(e_c, e_a) + \sum_{e_b \in E_b} custo(e_c, e_b) \\
&= \sum_{e_a \in E_a} 2\Lambda \frac{cap(e_a)\frac{|B_{R_i}|}{|R_i|}}{cap(A_{R_i}, B_{R_i})} cap(e_c) \\
&\quad + \sum_{e_b \in E_b} 2\Lambda \frac{cap(e_b)\frac{|A_{R_i}|}{|R_i|}}{cap(A_{R_i}, B_{R_i})} cap(e_c)
\end{aligned}$$

Lembrando que $w_\ell(A_{R_i}) = \sum_{e \in E_a} cap(e)$ e $w_\ell(B_{R_i}) = \sum_{e \in E_b} cap(e)$, temos:

$$\begin{aligned}
&= 2\Lambda \frac{w_\ell(A_{R_i})\frac{|B_{R_i}|}{|R_i|}}{cap(A_{R_i}, B_{R_i})} cap(e_c) + 2\Lambda \frac{w_\ell(B_{R_i})\frac{|A_{R_i}|}{|R_i|}}{cap(A_{R_i}, B_{R_i})} cap(e_c) \\
&= 2\Lambda \frac{cap(e_c)}{cap(A_{R_i}, B_{R_i})} \left(w_\ell(A_{R_i}) \frac{|B_{R_i}|}{|R_i|} + w_\ell(B_{R_i}) \frac{|A_{R_i}|}{|R_i|} \right).
\end{aligned}$$

A esparsidade ψ do corte (A_{R_i}, B_{R_i}) é descrita pela equação (6.3). Podemos então reescrever a equação anterior:

$$\sum_{e \in E_a \cup E_b} custo(e_c, e) = 2\Lambda\, cap(e_c) \frac{1}{\psi}.$$

Mas como R_i foi dividido nesta iteração, sabemos que $\psi \leq \Lambda$, logo:

$$\sum_{e \in E_a \cup E_b} custo(e_c, e) = 2\Lambda\, cap(e_c)\, \frac{1}{\psi} \geq 2\, cap(e_c). \tag{6.5}$$

Portanto, o custo atribuído às arestas de borda de R_i é suficiente para pagar pelas arestas do corte (A_{R_i}, B_{R_i}).

Agora precisamos provar que uma aresta e pertencente a A^* não paga mais do que uma fração de sua capacidade, ou seja, queremos comparar $\sum_{e' \in R} custo(e', e)$ com a capacidade de e.

Seja $e = (u, v) \in E(A^*)$. Sabemos que e pagará algum custo em toda iteração em que ela for aresta de borda de um subgrafo R_i que precisa ser dividido. Isto equivale a dizer que e pagará porque ocorreu pelo menos uma destas duas situações:

- o subgrafo a que v pertence foi dividido;
- o subgrafo a que u pertence foi dividido.

Seja x o número de iterações que a função `Rearranjo` executa até retornar a partição $\mathcal{P}_{A^*}$. Vamos denotar por $V_0, V_1, \ldots, V_x$ a seqüência de subgrafos aos quais o vértice v pertenceu em cada iteração do algoritmo. De forma análoga, denotaremos por $U_0, \ldots, U_x$ os conjuntos aos quais u pertenceu. Para qu e pague na iteração i, vamos supor, sem perda de generalidade, que na iteração i o conjunto V_i foi dividido em V_{i+1} e $V_i \setminus V_{i+1}$. O custo que esta divisão proporciona para e é de:

$$\begin{aligned}
\sum_{e_c \in (A_{V_i}, B_{V_i})} custo(e_c, e) &= \sum_{e_c \in (A_{V_i}, B_{V_i})} 2\Lambda\, cap(e)\, \frac{|V_i \setminus V_{i+1}|}{|V_i|}\, \frac{cap(e_c)}{cap(A_{V_i}, B_{V_i})} \\
&= 2\Lambda\, cap(e)\, \frac{|V_i \setminus V_{i+1}|}{|V_i|}\, \frac{cap(A_{V_i}, B_{V_i})}{cap(A_{V_i}, B_{V_i})} \\
&= 2\Lambda\, cap(e)\, \frac{|V_i \setminus V_{i+1}|}{|V_i|}.
\end{aligned}$$

Analogamente, se o conjunto dividido na iteração i for o subgrafo a que u pertence, temos a divisão de U_i em U_{i+1} e $U_i \setminus U_{i+1}$ e:

$$\sum_{e_c \in (A_{U_i}, B_{U_i})} custo(e_c, e) = 2\Lambda\, cap(e)\, \frac{|U_i \setminus U_{i+1}|}{|U_i|}.$$

Para as x iterações do algoritmo `Rearranjo`, calculamos um total C pago pela aresta e:

$$\begin{aligned} C &= 2\Lambda\, cap(e) \left[\left(\frac{|V_0| - |V_1|}{|V_0|} + \frac{|V_1| - |V_2|}{|V_1|} + \cdots + \frac{|V_{x-1}| - |V_x|}{|V_{x-1}|} \right) \right. \\ &\quad \left. + \left(\frac{|U_0| - |U_1|}{|U_0|} + \frac{|U_1| - |U_2|}{|U_1|} + \cdots + \frac{|U_{x-1}| - |U_x|}{|U_{x-1}|} \right) \right]. \end{aligned}$$

Mas para qualquer seqüência de números $n \geq n_0 > \cdots > n_s \geq 1$ vale que

$$\frac{n_0 - n_1}{n_0} + \frac{n_1 - n_2}{n_1} + \cdots + \frac{n_{s-1} - n_s}{n_{s-1}} \leq 2 \log n.$$

Logo:

$$\begin{aligned} C &\leq 2\Lambda\, cap(e)\, (2 \log n + 2 \log n) \\ &= 8\Lambda \log n\, cap(e) \\ &= 8 \tfrac{4\alpha}{\lambda} \log n\, cap(e) \\ &= 32 \tfrac{\alpha}{\lambda} \log n\, cap(e). \end{aligned}$$

Com $\lambda = 64\,\alpha \log n$, concluímos que:

$$C \leq \frac{cap(e)}{2}. \tag{6.6}$$

Por fim, vamos comparar quanto a aresta paga pelas demais e o quanto que é pago por ela ao final do algoritmo. Vamos chamar B o balanço que existe entre estes dois valores ao final do algoritmo para cada aresta. Este balanço pode ser expresso por:

$$B = \sum_{e \in E(A^*)} \Big(\sum_{e' \in E(A^*)} custo(e', e) - \sum_{e' \in E(A^*)} custo(e, e') \Big).$$

Podemos separar as arestas de R em dois grupos: as arestas de borda – conjunto de arestas que chamaremos de $Borda(A^*)$ – e as arestas que não são de borda, que chamaremos de **internas**. Note que as arestas de borda somente pagam pelas demais; assim, podemos reescrever

$$B = B_{int} + B_{borda}$$

onde

$$B_{int} = \sum_{\substack{e \in E(A^*) \\ e \notin Borda(A^*)}} \Big(\sum_{e' \in E(A^*)} custo(e', e) - \sum_{e' \in E(A^*)} custo(e, e') \Big)$$

e

$$B_{borda} = - \sum_{\substack{e \in E(A^*) \\ e \in Borda(A^*)}} \sum_{e' \in A^*} custo(e, e').$$

Mas podemos notar que B é sempre zero, pois ao somarmos tudo que é pago pelas arestas ao longo do algoritmo certamente será igual a tudo que foi cobrado por elas. Logo, usando este fato e os resultados das equações (6.5) e (6.6), temos:

$$\begin{aligned}
B &= B_{int} + B_{borda} \\
0 &= \sum_{\substack{e \in E(A^*) \\ e \notin Borda(A^*)}} \Big(\sum_{e' \in E(A^*)} custo(e', e) - \sum_{e' \in E(A^*)} custo(e, e') \Big) - \sum_{\substack{e \in E(A^*) \\ e \in Borda(A^*)}} \sum_{e' \in E(A^*)} custo(e', e) \\
&\geq \sum_{\substack{e \in E(A^*) \\ e \notin Borda(A^*)}} (2\, cap(e) - \frac{1}{2} cap(e)) - \sum_{\substack{e \in E(A^*) \\ e \in Borda(A^*)}} \frac{1}{2} cap(e) \\
&= \sum_{\substack{e \in E(A^*) \\ e \notin Borda(A^*)}} \frac{3}{2} cap(e) - \sum_{\substack{e \in E(A^*) \\ e \in Borda(A^*)}} \frac{1}{2} cap(e).
\end{aligned}$$

Como $\sum_{\substack{e \in E(A^*) \\ e \in Borda(A^*)}} cap(e) = brd(A^*)$, reescrevemos:

$$\begin{aligned}
\tfrac{1}{2}\, brd(A^*) &\geq \tfrac{3}{2} \sum_{\substack{e \in E(A^*) \\ e \notin Borda(A^*)}} cap(e) \\
brd(A^*) &\geq 3 \sum_{\substack{e \in E(A^*) \\ e \notin Borda(A^*)}} cap(e).
\end{aligned}$$

Somando $brd(A^*)$ nos dois lados, temos:

$$\begin{aligned}
brd(A^*) + brd(A^*) &\geq brd(A^*) + 3 \sum_{\substack{e \in E(A^*) \\ e \notin Borda(A^*)}} cap(e) \\
2\, brd(A^*) &\geq brd(A^*) + 3 \sum_{\substack{e \in E(A^*) \\ e \notin Borda(A^*)}} cap(e)
\end{aligned}$$

Como $2 \sum_{\substack{e \in E(A^*) \\ e \notin Borda(A^*)}} cap(e)$ representa o somatório das arestas que separam subgrafos de A^*, vale que

$$2 \sum_{\substack{e \in E(A^*) \\ e \notin Borda(A^*)}} cap(e) + brd(A^*) = \sum_i w_{\ell+1}(R_i).$$

Usando este fato par continuar a última desigualdade, temos:

$$\begin{aligned} 2\,brd(A^*) &\geq brd(A^*) + 3 \sum_{\substack{e\in E(A^*)\\ e\notin Borda(A^*)}} cap(e) \\ &\geq brd(A^*) + 2 \sum_{\substack{e\in E(A^*)\\ e\notin Borda(A^*)}} cap(e) \\ &= \textstyle\sum_i w_{\ell+1}(R_i). \end{aligned}$$

Portanto, $2\,brd(A^*) \geq w_{\ell+1}(R_i)$. □

Usando o resultado do lema 6.2, podemos substituir $w_{\ell+1}(A^*_{novo})$ na equação (6.4).

$$\begin{aligned} \Delta W &= -w_{\ell+1}(A^*_{ant}) + w_{\ell+1}(A^*_{novo}) \\ &\leq -w_{\ell+1}(A^*_{ant}) + 2\,brd(A^*). \end{aligned} \tag{6.7}$$

Note que $brd(A^*_{novo}) = brd(A^*_{ant})$, logo, não faremos distinção se a partição de A^* é anterior ou posterior à aplicação do algoritmo 4 e denotaremos este valor por $brd(A^*)$.

Para terminar a prova e verificar se ΔW é positivo ou negativo, vamos relacionar o valor de $w_{\ell+1}(A^*_{ant})$ com $brd(A^*)$. Esta relação está demonstrada a seguir, no lema 6.3.

Lema 6.3. *O arredondamento do corte mais esparso feito no algoritmo 3 produz um conjunto de subgrafos A^* tal que $w_{\ell+1}(A^*) \geq 3\,brd(A^*)$.*

Demonstração. Calculando a quantidade de demanda entre vértices separados pelo corte mais esparso aproximado (A, B) da função `Particiona`, temos:

$$\begin{aligned} d^\ell_H(A,B) &= \sum_{u\in A}\sum_{v\in B}\Big(d^\ell_H(u,v) + d^\ell_H(v,u)\Big) \\ &= \sum_{u\in A}\sum_{v\in B}\Big(\frac{w_{\ell+1}(u)\,w_{\ell+1}(v)}{w_{\ell+1}(H)} + \frac{w_{\ell+1}(u)\,w_{\ell+1}(v)}{w_{\ell+1}(H)}\Big) \\ &= 2\,\frac{w_{\ell+1}(A)\,w_{\ell+1}(B)}{w_{\ell+1}(H)} \\ &\leq 2\,w_{\ell+1}(A). \end{aligned}$$

Logo,

$$esp(A,B) = \frac{cap(A,B)}{d^\ell_H(A,B)} \geq \frac{cap(A,B)}{2\,w_{\ell+1}(A)}.$$

Como $esp(A,B) \leq \alpha\, z_{inf}$,

$$\alpha\, z_{inf} \geq \frac{cap(A,B)}{d_H^{\ell}(A,B)} \geq \frac{cap(A,B)}{2\, w_{\ell+1}(A)}. \tag{6.8}$$

Mas como no algoritmo o conjunto A é arredondado, dando origem ao subgrafo A^*, queremos relacionar de alguma forma $cap(A,B)$ com A^*. Para isto, vamos definir os conjuntos $A_i = A \cap H_i$ e $B_i = B \cap H_i$. Denotando por $\overline{H_i} = \mathcal{P}_H \setminus H_i$ e lembrando que I_S é o conjunto de subgrafos que contém interseção pequena com A (menos de $\frac{3}{4}$ de seus vértices pertencem ao conjunto A) e que o conjunto I_L é o conjunto de subgrafos que possuem interseção grande com A, usaremos a relação entre A e A^* para reescrevemos a equação anterior:

$$\begin{aligned}\frac{cap(A,B)}{w_{\ell+1}(A)} &= \frac{cap(A,B)}{\sum_{H_i \in \mathcal{P}_H} cap(A_i, \overline{H_i})} \\ &= \frac{cap(A,B)}{\sum_{H_i \in I_L} cap(A_i, \overline{H_i}) + \sum_{H_i \in I_S} cap(A_i, \overline{H_i})}.\end{aligned}$$

Mas $A_i \subseteq V(H_i)$ para todo $H_i \in \mathcal{P}_H$. Logo, $\sum_{H_i \in I_L} cap(A_i, \overline{H_i}) \leq \sum_{H_i \in I_L} cap(H_i, \overline{H_i})$, e temos:

$$\frac{cap(A,B)}{w_{\ell+1}(A)} \geq \frac{cap(A,B)}{\sum_{H_i \in I_L} cap(H_i, \overline{H_i}) + \sum_{H_i \in I_S} cap(A_i, \overline{H_i})}.$$

Mas, como $A^* = I_L$, $\sum_{H_i \in I_L} cap(H_i, \overline{H_i}) = w_{\ell+1}(A^*)$. Reescrevendo a equação anterior:

$$\begin{aligned}\frac{cap(A,B)}{w_{\ell+1}(A)} &\geq \frac{cap(A,B)}{w_{\ell+1}(A^*) + \sum_{H_i \in I_S} cap(A_i, \overline{H_i})} \\ &\geq \frac{cap(A,B)}{w_{\ell+1}(A^*)}.\end{aligned} \tag{6.9}$$

Agora vamos provar uma relação entre $cap(A,B)$ e $brd(A^*)$.

Primeiramente vamos argumentar que a precondição de vazão (equação (6.1)) também vale para o nó H dado como entrada do algoritmo `Particiona`. Para construir a árvore, executamos o algoritmo `Constrói Árvore` (algoritmo 2) passando como parâmetro o grafo de entrada do problema e a altura da raiz (zero); a precondição de vazão vale para a raiz da árvore de decomposição porque $w_0(V) = 0$, logo

$$\lambda\, cap(U, V \setminus U) \geq w_0(V) = 0$$

para qualquer $U \subset V$.

A partir da raiz da árvore, o algoritmo 2 usa o algoritmo `Particiona` (algoritmo 3) que, por sua vez, utiliza o algoritmo `Rearranjo` (algoritmo 4) para produzir uma partição em subgrafos. Mas como cada subgrafo produzido pelo algoritmo 4 possui a precondição de vazão, todos os subgrafos passados como parâmetro para o algoritmo 3 também possuem esta propriedade.

Portanto, podemos falar que para o nó H da função `Particiona`, como $|A| \leq \frac{1}{2}|H| \leq \frac{3}{4}|H|$, vale que

$$\lambda\, cap(A, B) \geq w_\ell(A).$$

Mas $(A^* \cap A) \subset A$, logo

$$\lambda\, cap(A, B) \geq w_\ell(A) \geq w_\ell(A^* \cap A). \tag{6.10}$$

Para os subgrafos de H, também vale a condição estabelecida para ter a precondição de vazão, visto que eles também foram formados através da função `Rearranjo`. Para todo $H_i \in I_S$ vale que $|A_i| \leq \frac{1}{4}|H_i| \leq \frac{3}{4}|H_i|$, assim temos

$$\forall H_i \in I_S \qquad \lambda\, cap(A_i, B_i) \geq w_{\ell+1}(A_i).$$

Analogamente, para os subgrafos $H_i \in I_L$, temos $|B_i| \leq \frac{1}{4}|H_i| \leq \frac{3}{4}|H_i|$. Portanto:

$$\forall H_i \in I_L \qquad \lambda\, cap(A_i, B_i) \geq w_{\ell+1}(B_i).$$

Somando $cap(A_i, B_i)$ nos dois lados das duas equações anteriores:

$$\begin{aligned} \forall H_i \in I_S \quad & (\lambda + 1)\, cap(A_i, B_i) \geq w_{\ell+1}(A_i) + cap(A_i, B_i) = brd(A_i) \\ \forall H_i \in I_L \quad & (\lambda + 1)\, cap(A_i, B_i) \geq w_{\ell+1}(B_i) + cap(A_i, B_i) = brd(B_i). \end{aligned} \tag{6.11}$$

Somando a equação anterior para todos os subgrafos de I_L, temos:

$$\begin{aligned} (\lambda + 1) \sum_{H_i \in I_L} cap(A_i, B_i) &\geq \sum_{H_i \in I_L} \Big(w_{\ell+1}(B_i) + cap(A_i, B_i) \Big) \\ &= \sum_{H_i \in I_L} w_{\ell+1}(B_i) + \sum_{H_i \in I_L} cap(A_i, B_i) \\ \lambda \sum_{H_i \in I_L} cap(A_i, B_i) &\geq \sum_{H_i \in I_L} w_{\ell+1}(B_i) \end{aligned}$$

Como $\sum_{H_i \in I_L} w_{\ell+1}(B_i) \geq \sum_{H_i \in I_L} w_\ell(B_i)$,

$$\begin{aligned} \lambda \sum_{H_i \in I_L} cap(A_i, B_i) &\geq \sum_{H_i \in I_L} w_\ell(B_i) \\ &= w_\ell(\bigcup B_i) = w_\ell(A^* \setminus A). \end{aligned}$$

Usando a desigualdade anterior e somando as desigualdades (6.11) respectivamente para todos os subgrafos em I_S e I_L, ou seja, para todos os subgrafos de $\mathcal{P}_H$, obtemos

$$\begin{aligned}(\lambda+1)\Big(\sum_{H_i\in I_L} cap(A_i,B_i)+\sum_{H_i\in I_S} cap(A_i,B_i)\Big) &\geq \textstyle\sum_{H_i\in I_L} brd(B_i)+\sum_{H_i\in I_S} brd(A_i)\\ &= w_\ell(A^*\setminus A)+\textstyle\sum_{H_i\in I_S} brd(A_i).\end{aligned}$$

Mas percebe-se que $cap(A,B)\geq \sum_{H_i\in I_L} cap(A_i,B_i)+\sum_{H_i\in I_S} cap(A_i,B_i)$, porque $\sum_{H_i\in I_L} cap(A_i,B_i)+\sum_{H_i\in I_S} cap(A_i,B_i)$ conta somente arestas que ligam vértices que ficaram em lados separados pelo corte (e deixa de contar, por exemplo, arestas que ligam vértices em um conjunto A_i a vértices em um conjunto B_j, com $i\neq j$). Usando esta informação na equação anterior, chegamos a:

$$\begin{aligned}(\lambda+1)\,cap(A,B) &\geq (\lambda+1)\Big(\textstyle\sum_{H_i\in I_L} cap(A_i,B_i)+\sum_{H_i\in I_S} cap(A_i,B_i)\Big)\\ &\geq \sum_{H_i\in I_L} brd(B_i)+\sum_{H_i\in I_S} brd(A_i)\\ &= w_\ell(A^*\setminus A)+\sum_{H_i\in I_S} brd(A_i).\end{aligned} \tag{6.12}$$

Há uma relação entre $cap(A^*, H\setminus A^*)$ e $cap(A,B)$. Primeiramente,

$$\begin{aligned}cap(A^*,H\setminus A^*) &= cap\Big(\bigcup_{H_i\in I_L} H_i,\bigcup_{H_i\in I_S} H_i\Big)\\ &= cap\Big(\bigcup_{H_i\in I_L} A_i,\bigcup_{H_i\in I_S} B_i\Big)+cap\Big(\bigcup_{H_i\in I_L} A_i,\bigcup_{A_i\in I_S} A_i\Big)\\ &\quad +cap\Big(\bigcup_{H_i\in I_L} B_i,\bigcup_{H_i\in I_S} A_i\Big)+cap\Big(\bigcup_{H_i\in I_L} B_i,\bigcup_{H_i\in I_S} B_i\Big).\end{aligned} \tag{6.13}$$

Por outro lado, temos:

$$\begin{aligned}cap(A,B) &= \sum_{\substack{H_i\in\mathcal{P}_H\\ A_i\neq\emptyset\\ B_i\neq\emptyset}} cap(A_i,B_i)+cap\Big(\bigcup_{H_i\in I_L} A_i,\bigcup_{H_i\in I_S} B_i\Big)\\ &\geq cap\Big(\bigcup_{H_i\in I_L} A_i,\bigcup_{H_i\in I_S} B_i\Big)+cap\Big(\bigcup_{H_i\in I_L} B_i,\bigcup_{H_i\in I_S} A_i\Big).\end{aligned} \tag{6.14}$$

Ademais,

$$\begin{aligned}\sum_{H_i\in I_L} brd(B_i) &= cap\Big(\bigcup_{H_i\in I_L} B_i, V\setminus\bigcup_{H_i\in I_L} B_i\Big)\\ &\geq cap\Big(\bigcup_{H_i\in I_L} B_i,\bigcup_{H_i\in I_S} B_i\Big).\end{aligned} \tag{6.15}$$

Analogamente, para $cap(\bigcup_{H_i \in I_L} A_i, \bigcup_{A_i \in I_S} A_i)$, temos:

$$\begin{aligned} \sum_{H_i \in I_S} brd(A_i) &= cap(\bigcup_{H_i \in I_L} A_i, V \setminus \bigcup_{H_i \in I_L} A_i) \\ &\geq cap(\bigcup_{H_i \in I_L} A_i, \bigcup_{H_i \in I_S} A_i). \end{aligned} \tag{6.16}$$

Usando as equações (6.14), (6.15) e (6.16), reescrevemos a equação (6.13):

$$cap(A^*, H \setminus A^*) \leq cap(A, B) + \sum_{H_i \in I_S} brd(A_i) + \sum_{H_i \in I_S} brd(B_i).$$

Mas, pela equação (6.12), temos que $\sum_{H_i \in I_L} brd(B_i) + \sum_{H_i \in I_S} brd(A_i) \leq (\lambda + 1)\left(\sum_{H_i \in I_L} cap(A_i, B_i) + \sum_{H_i \in I_S} cap(A_i, B_i)\right)$, logo:

$$\begin{aligned} cap(A^*, H \setminus A^*) &\leq cap(A, B) + \textstyle\sum_{H_i \in I_S} brd(A_i) + \sum_{H_i \in I_S} brd(B_i) \\ &\leq cap(A, B) + (\lambda + 1)\left(\textstyle\sum_{H_i \in I_L} cap(A_i, B_i) + \sum_{H_i \in I_S} cap(A_i, B_i)\right) \\ &= (\lambda + 2)\, cap(A, B). \end{aligned} \tag{6.17}$$

E, finalmente, usaremos as equações (6.10), (6.12) e (6.17) para relacionar $brd(A^*)$ e $cap(A, B)$:

$$\begin{aligned} brd(A^*) + \sum_{H_i \in I_S} brd(A_i) &\leq cap(A^*, H \setminus A^*) + w_\ell(A^*) + \sum_{H_i \in I_S} brd(A_i) \\ &\leq cap(A^*, H \setminus A^*) + w_\ell(A^* \cap A) \\ &\quad + w_\ell(A^* \setminus A) + \sum_{H_i \in I_S} brd(A_i) \\ &\leq cap(A, B)\,((\lambda + 2) + \lambda + (\lambda + 1)) \\ &\leq 4\lambda\, cap(A, B). \end{aligned} \tag{6.18}$$

Usando este último resultado, podemos substituir $cap(A, B)$ na equação (6.9), resultando em:

$$\begin{aligned} \frac{cap(A, B)}{w_{\ell+1}(A)} &\geq \frac{cap(A, B)}{w_{\ell+1}(A^*)} \\ &\geq \frac{1}{4\lambda} \frac{brd(A^*) + \sum_{H_i \in I_S} brd(A_i)}{w_{\ell+1}(A^*)} \\ &\geq \frac{1}{4\lambda} \frac{brd(A^*)}{w_{\ell+1}(A^*)}. \end{aligned} \tag{6.19}$$

Usando as equações (6.8) e (6.19), temos:

$$\begin{aligned} \alpha\, z_{inf} &\geq \frac{cap(A,B)}{2\, w_{\ell+1}(A)} \\ &\geq \frac{1}{8\lambda}\frac{brd(A^*)}{w_{\ell+1}(A^*)}. \end{aligned}$$

Logo,

$$w_{\ell+1}(A^*) \geq \frac{1}{8\,\alpha\,\lambda\, z_{inf}}\, brd(A^*).$$

Com $z_{inf} = \frac{1}{24\,\lambda\,\alpha}$, concluímos que

$$w_{\ell+1}(A^*) \geq 3\, brd(A^*).$$

□

Com o lema 6.3, podemos reescrever a equação (6.7), já que $w_{\ell+1}(A^*_{ant}) \geq 3\, brd(A^*)$:

$$\begin{aligned} \Delta W &\leq -w_{\ell+1}(A^*_{ant}) + 2\, brd(A^*) \\ &\leq -(3\, brd(A^*)) + (2\, brd(A^*)) \\ &= -brd(A^*). \end{aligned}$$

Como as capacidades das arestas do grafo de entrada foram normalizadas, sabemos que $brd(A^*) \geq 1$. Logo, $w_{\ell+1}(H)$ decresce de pelo menos 1 em cada iteração e, portanto, o algoritmo de particionamento termina em tempo polinomial em n e em $c_{\max}$, a maior das capacidades das arestas de G.

Por fim, falta mostrar que a árvore produzida não possui altura muito grande.

Lema 6.4. *Seja $\mathcal{T}_{\mathcal{D}_G}$ a árvore de decomposição produzida através da chamada da função* `Particiona`$(G, 0)$. *A altura h de $\mathcal{T}_{\mathcal{D}_G}$ é $O(\log n)$.*

Demonstração. Começamos o algoritmo `Particiona` somente com subgrafos triviais. A cada iteração, achamos um corte mais esparso (A, B) e o algoritmo de arredondamento é usado em seu menor lado, produzindo um corte (A^*, B^*).

Supondo $|A| \leq |B|$, o arredondamento do lado A produz um subgrafo A^*, que será a entrada da função `Rearranjo`. Mas podemos notar o tamanho máximo dos subgrafos que a função `Rearranjo` retornará é $|V(A^*)|$, visto que esta função começa com um único subgrafo contendo todos os vértices e nas demais iterações do algoritmo podemos apenas particionar subgrafos existentes, diminuindo assim o tamanho dos mesmos.

Para obter o tamanho de $V(A^*)$, é preciso lembrar que este conjunto é construído através da retirada de subgrafos com interseção pequena com A e adicionando a parte restante dos subgrafos com interseção grande com A. O tamanho de $V(A^*)$ pode ser calculado por:

$$|V(A^*)| \leq |A| + \sum_{H_i \in I_L} \frac{1}{4}|V(H_i)|.$$

Como $|A| \leq |B|$, temos que $|A| \leq \frac{1}{2}|H|$:

$$\begin{aligned} |V(A^*)| &\leq |A| + \sum_{H_i \in I_L} \frac{1}{4}|V(H_i)| \\ &\leq \tfrac{1}{2}|V(H)| + \tfrac{1}{4} \sum_{H_i \in I_L} |V(H_i)|. \end{aligned}$$

Lembrando que todo $H_i \in I_L$ também foi produzido pelo algoritmo `Rearranjo` em alguma iteração anterior, temos que $\sum_i |V(H_i)| \leq |V(A^*)|$, logo:

$$\begin{aligned} |V(A^*)| &\leq \tfrac{1}{2}|V(H)| + \tfrac{1}{4}|V(A^*)| \\ \tfrac{3}{4}|V(A^*)| &\leq \tfrac{1}{2}|V(H)| \\ |V(A^*)| &\leq \tfrac{4}{3}\tfrac{1}{2}|V(H)| \\ |V(A^*)| &\leq \tfrac{2}{3}|V(H)|. \end{aligned}$$

Como em cada nível da árvore temos que um nó é limitado a ter uma fração de no máximo $\frac{2}{3}$ dos vértices de seu nó pai, concluímos que a árvore possui uma altura $O(\log n)$. □

Juntando o teorema 5.3, o lema 6.4 e o fato de $z_{inf} = \frac{1}{24\,\alpha\,\lambda}$ (valor obtido na demonstração do lema 6.3), concluímos que o algoritmo apresentado é polinomial e possui razão de competitividade igual a

$$O\left(\frac{h}{z_{inf}}\right) = O\left(\frac{\log n}{1/(24\,\alpha\,\lambda)}\right) = O\left(\frac{\log n}{1/(24\,\alpha\,64\,\alpha\,\log n)}\right) = O(24 \cdot 64\,\alpha^2 \log^2 n).$$

Usando $\alpha = O(\log n)$ (seção 2.4), a razão de competitividade do algoritmo é de:

$$O(24 \cdot 64\,\alpha^2 \log^2 n) = O(\log^4 n).$$

Capítulo 7

O algoritmo H.H.R.

Este capítulo será dedicado a apresentar e analisar o algoritmo de Harrelson, Hildrum e Rao [15], que possui razão de competitividade $O(\log^2 n \log\log n)$ para o congestionamento em redes não-direcionadas com n vértices e capacidades nas arestas.

Este algoritmo também utiliza uma árvore de decomposição do modo descrito no capítulo 5, diferindo do algoritmo do capítulo 6 na forma de construção desta. No algoritmo H.H.R., cada nó da árvore de decomposição possui dois tipos de subgrafos em sua partição, chamados de subgrafos primários e secundários. Por isso, a redistribuição de fluxo na rede que representa a passagem de fluxo dentro de um nó da árvore também é feita de forma distinta: os caminhos calculados no CMCF descrito na seção 5.2 serão usados para transportar fluxo entre subgrafos primários, enquanto um problema de fluxo máximo será usado para transmitir fluxo entre subgrafos primários e secundários.

O algoritmo constrói a árvore a partir da raiz. Em cada nó da árvore é usado um problema CMCF com as demandas descritas na seção 5.2 e uma partição destes subgrafos é construída usando cortes mais esparsos aproximados, construindo um conjunto de subgrafos primários. Diferentemente do capítulo anterior, mesmo após a construção desta partição para um nó H da árvore é possível que o problema CMCF definido em alguns de seus subgrafos primários não consigam alcançar a fração de vazão desejada e, por isso, esta partição terá que ser refeita. Neste caso, cada um dos subgrafos primários que não alcançaram tal fração de vazão serão reparticionados em subgrafos secundários, que serão os filhos de H na árvore.

Assim como o algoritmo do capítulo 6, este algoritmo também é pseudopolinomial em função de n e da maior capacidade das arestas. Portanto, será assumido que a razão entre as capacidades máxima e mínima é polinomial em n.

No resto do capítulo suporemos que:

-
- $n > 4$, ou seja, $\log\log n \geq 1$;
- $\alpha = \gamma \log n$, isto é, γ é a constante no fator de aproximação do algoritmo usado para obter um corte mais esparso aproximado com fator $\alpha = O(\log n)$.

Além disso, as seguintes igualdades serão usadas:

- $$\delta(H) = \frac{1}{8\gamma(\log \frac{|P|}{|H|}) \log\log n},$$
  onde H e P são nós da árvore de decomposição e P é pai de H na árvore;
- $z_{inf} = \frac{\delta(H)}{\log n}$.

A seção 7.1 descreve o método de construção da árvore de decomposição; a seção 7.2 descreve como acontece o roteamento entre subgrafos primários e secundários dentro de um nó da árvore. Por fim, na seção 7.3 demonstra-se sua razão de competitividade e que ele executa em tempo polinomial.

7.1 O algoritmo de construção da árvore de decomposição

O algoritmo de construção da árvore de decomposição terá duas etapas para cada nó H da árvore: a primeira produz uma família de subgrafos de H que serão chamados de **subgrafos primários** e a segunda faz um refinamento destes subgrafos, produzindo outros chamados de **subgrafos secundários**. Estes últimos serão os filhos de H na árvore.

As próximas subseções, 7.1.1 e 7.1.2, são dedicadas a estas duas etapas do algoritmo.

7.1.1 Etapa de pré-processamento

Esta etapa do algoritmo recebe como entrada um subgrafo H e retorna uma partição $\mathcal{P}_H$ para a qual é possível rotear no CMCF uma fração maior ou igual a z_{inf} da demanda entre cada par de vértices. Esta etapa cria um conjunto de candidatos a filhos de H na árvore, pois são conjuntos entre os quais é possível rotear com congestionamento baixo. Ao contrário do algoritmo B.K.R., é possível que não exista particionamento de H capaz de produz fração de vazão maior ou igual a z_{inf}, caso em que o algoritmo retornará falha.

A demanda do CMCF em um nó H do nível ℓ da árvore é dada por

$$d_H^\ell(u,v) = \frac{w_{\ell+1}(u) \cdot w_{\ell+1}(v)}{w_{\ell+1}(H)} \qquad \forall u, v \in V. \tag{7.1}$$

Caso tal partição não seja possível de ser construída, o algoritmo retorna falha, indicando o corte mais esparso aproximado, conforme descrito no algoritmo 5.

Algoritmo 5 Pre-processamento(H, ℓ)

$\mathcal{P}_H := \{\{v\} \mid v \in V(H)\}$;
repita
  $d_H^\ell :=$ demanda no CMCF de H;
  $F := CMCF(H, \mathcal{P}_H, d_H^\ell)$;
  $z_H := \frac{1}{congestionamento(G,F)}$; {$z_H$ é fração de vazão do CMCF em H}
  se $z_H \geq z_{inf}$ **então**
    $(A, B) :=$ Corte Esparso Aproximado(G, d_H^ℓ); $\{|A| \leq |B|\}$
    se $w_{\ell+1}(A) - w_\ell(A) \geq \frac{1}{2\gamma\delta(H)} cap(A, B)$ **então**
      $\mathcal{P}_H := \{A\} \cup \left(\biguplus_{H_i \in \mathcal{P}_H} B_i\right)$; {onde $B_i = H_i \cap B$}
    senão
      retorna falha$((A, B))$;
até que $z_H \geq z_{inf}$
retorna $\mathcal{P}_H$;

No início do algoritmo, a partição possui $|V(H)|$ subgrafos, cada um deles contendo um vértice pertencente a H.

A cada iteração o algoritmo testa se já é possível rotear a demanda do CMCF com fração de vazão maior ou igual a z_{inf} com a partição atual. Em caso afirmativo, o

algoritmo pára e retorna $\mathcal{P}_H$. Caso contrário, o algoritmo encontra um corte mais esparso aproximado (A, B), com $|A| \leq |B|$. Se $w_{\ell+1}(A) - w_\ell(A) \geq \frac{1}{2\gamma\delta(H)} cap(A, H \setminus A)$, o algoritmo modifica $\mathcal{P}_H$: os vértices de A são retirados da partição (esta operação pode gerar subgrafos cujo conjunto de vértices é vazio, que são retirados de $\mathcal{P}_H$) e são adicionados novamente à partição formando um único subgrafo, $H[A]$. Com esta nova partição, o algoritmo inicia uma nova iteração, testando novamente a fração de vazão do CMCF.

Caso $w_{\ell+1}(A) - w_\ell(A) < \frac{1}{2\gamma\delta(H)} cap(A, B)$, o algoritmo pára com falha e retorna o corte mais esparso aproximado (A, B). Este evento será chamado de **nó inválido**.

7.1.2 Etapa de refinamento

Após o término da etapa de pré-processamento em um nó H os conjuntos primários $H_i \in \mathcal{P}_H$ são candidatos a serem filhos de H na árvore de decomposição. Porém, não é garantido que a etapa de pré-processamento fez um particionamento de forma a que todos os subconjuntos de H consigam alcançar a fração de vazão z_{inf}. O objetivo da etapa de refinamento é evitar que sejam criados subgrafos que não possam ser particionados de tal forma que seus CMCFs consigam obter uma fração de vazão maior ou igual a z_{inf}.

O algoritmo de refinamento recebe como entrada a partição $\mathcal{P}_H$ composta de subgrafos primários produzidos pelo algoritmo de pré-processamento e retorna subgrafos secundários, que serão os filhos de H na árvore. A diferença para a etapa anterior é que é garantido que estes novos conjuntos consigam alcançar a fração de vazão. Ao particionar os subgrafos primários em secundários, cria-se conjuntos ainda menores que os anteriores, o que pode diminuir ainda mais a altura da árvore. Além disso, será aproveitado o roteamento entre os conjuntos primários e secundários, garantindo assim que existe uma forma eficiente de rotear entre eles.

A etapa de refinamento em H acontece simultaneamente com a etapa de pré-processamento em seus subgrafos primários. No refinamento, tenta-se particionar — usando a função Pré-processamento — cada subgrafo primário $H_i \in \mathcal{P}_H$ construído na etapa de pré-processamento de H. Caso o pré-processamento de um subgrafo primário H_i retorne falha, ele é particionado em uma família de subgrafos secundários; já os subgrafos primários que não retornam falha na etapa de pré-

processamento são marcados como secundários.

No caso em que há falha na etapa de pré-processamento de um subgrafo primário S de um nó H da árvore, pelo algoritmo 5 pode-se concluir que $w_{\ell+1}(S) - w_\ell(S)$ é pequeno (isto é, menor do que $\frac{1}{2\gamma\delta(S)}\, cap(A,B)$). Como no CMCF do nó S a demanda que será mandada para os vértices de S é igual a $w_{\ell+1}(S)$, estes dois fatos implicam que há muita demanda a ser roteada dentro de S, porém há pouca capacidade nas arestas internas em comparação a das arestas externas para transportar este fluxo a cada vértice de S.

A solução dada pelo algoritmo para remediar esta situação é particionar em subgrafos secundários cada subgrafo primário que falha em seu pré-processamento. A cada iteração na etapa de refinamento, um subgrafo S cujo pré-processamento retorne falha é retirado da partição $\mathcal{P}_H$ e em seu lugar são adicionados dois subgrafos: U e $S \backslash U$. Este processo é repetido até que não haja mais na partição de H subgrafos cujas etapas de pré-processamento retornem falha.

No entanto, a criação de mais subgrafos aumenta a demanda a ser roteada no CMCF do nó H, já que $\sum_{v\in H}\sum_{u\in H} d^\ell_H(u,v) = w_{\ell+1}(H)$ (pela equação (7.1)) e $w_{\ell+1}(H)$ aumenta com as capacidades das arestas que ligam vértices em subgrafos distintos.

Seja S um subgrafo que será particionado em U e $S \setminus U$ em alguma iteração da etapa de refinamento. A quantidade de demanda criada pela divisão de S é igual a $2\, cap(U, S \setminus U)$ e a origem desta demanda gerada no CMCF são os vértices que são extremidades das arestas que pertencem ao corte $(U, S \setminus U)$. Para rotear esta nova quantidade de demanda também pelos caminhos usados na solução do CMCF calculada na etapa de pré-processamento de H, o algoritmo enviará esta quantidade por um fluxo das extremidades do corte até os vértices de borda de U (supondo $|U| \leq |S \setminus U|$).

Dado que (A, B) foi o corte mais esparso aproximado retornado pela falha na etapa de pré-processamento de S, a partição deste nó em A e B separa os dois lados do corte que não se comunicam bem em S, criando subgrafos diferentes. Porém, para dividir S desta forma e aplicar a solução dada anteriormente, é necessário assegurar que todo este novo fluxo consiga ir dos vértices da extremidade do corte até os vértices de borda de A, ou seja, é preciso assegurar que não haja um corte

dentro do subgrafo $S[A]$ cuja capacidade seja menor que o valor do fluxo que precisa chegar da extremidade do corte (A, B) ao vértices que ligam A a $V \setminus S$.

Para evitar este problema, o algoritmo usa uma função auxiliar, que recebe como entrada o corte (A, B) retornado pela etapa de processamento e encontra um corte $(U, S \setminus U)$ com as seguintes propriedades:

1. $U \subseteq A$ e portanto $|U| \leq |S \setminus U|$;
2. A capacidade do corte $(U, S \setminus U)$ é igual à capacidade do corte mínimo em $G[U]$; com isto é possível rotear toda a demanda criada pela divisão de S em U e $S \setminus U$;
3. $esp((U, S \setminus U)) \leq esp((A, B))$; isto indica que o corte $U, S \setminus U$ é tão ou mais esparso que (A, B) e, por isso, a separação de S nestes dois subgrafos será tão boa quanto ou até melhor do que a divisão em A e B.

Esta função auxiliar, que será chamada de `FluxoMáximo`, utiliza um problema de fluxo máximo e corte mínimo para encontrar o menor corte no subgrafo $G[A]$. O algoritmo inicializa o corte $(U, S \setminus U)$ sendo igual ao corte (A, B). Para encontrar se há um corte mínimo que impeça a chegada de todo o fluxo das extremidades do corte à borda de U, é usado um problema de fluxo máximo $\mathcal{F}(U)$ em uma rede $R(U)$ construída da seguinte forma:

- $R(U)$ possui todos os vértices de U e mais dois novos vértices, s e t;
- para todo $v \in U$ há arestas (s, v) com capacidade igual a $cap(v, S \setminus U)$;
- para todo $v \in U$ há arestas (v, t) com capacidade igual a $\frac{w_\ell(v)}{w_\ell(U)} cap(U, S \setminus U)$.

Em $\mathcal{F}$, o vértice s será a fonte e o vértice t, sumidouro. O valor do fluxo máximo (e, consequentemente, do corte mínimo) neste problema indica a quantidade de fluxo que pode chegar aos vértices de borda de U, visto que somente há arestas com capacidade positiva entre um vértice $v \in U$ e t quando v é vértice de borda. Além disso, o valor máximo a que o fluxo pode chegar é igual a $\sum_{v \in U} cap((s, v)) = cap(U, S \setminus U)$, caso em que toda a demanda consegue ser roteada para a borda de U.

Se o corte mínimo é igual a $(s, U \cup \{t\})$ ou $(U \cup \{s, t\})$, então o valor do corte é $cap(U, S \setminus U)$ e o algoritmo retorna o corte $(U, S \setminus U)$. Os caminhos usados na

solução de $\mathcal{F}$ também serão usados como o esquema de roteamento que levará o fluxo da borda do corte $(U, S \setminus U)$ para a borda de U.

Caso s e t estejam do mesmo lado do corte, o algoritmo modifica o corte $(U, S \setminus U)$ para que este seja igual ao corte mínimo de $\mathcal{F}$, excluindo-se os vértices s e t. Na próxima iteração do algoritmo, como o conjunto U foi modificado, o problema $\mathcal{F}(U)$ também terá sua rede $R(U)$ modificada. Este processo é repetido até que o valor do fluxo máximo seja igual ao valor do corte $(U, S \setminus U)$.

A função `FluxoMáximo`, descrita no algoritmo 6, recebe como entrada o subgrafo S e o corte mais esparso (A, B) e retorna um corte $U, S \setminus U)$, com as características descritas.

Algoritmo 6 FluxoMáximo$(S, (A, B))$

$U := A$;
repita
  $\mathcal{C} :=$ `CorteMínimo`$(\mathcal{F}(U))$; {com $\mathcal{C} = (X, Y)$ e $t \in X$ }
  se $\mathcal{C} = (\{s\}, U \cup t)$ ou $\mathcal{C} = (U \cup \{s\}, \{t\})$ **então**
    retorna $(U, S \setminus U)$;
  senão
    $U := X \cap U$;
até que valor do fluxo maximo $= cap(U, S \setminus U)$;

O lema 7.1 contém a prova de corretude do algoritmo 6.

Lema 7.1. *O algoritmo 6 termina em tempo polinomial e retorna um corte* $(U, S \setminus U)$ *com as seguintes características:*

1. $U \subseteq A$;
2. $cap(U, S \setminus U) =$`CorteMínimo`$(\mathcal{F}(U))$;
3. $esp((U, S \setminus U)) \leq esp((A, B))$.

Demonstração. Se $(U, S \setminus U) = (A, B)$ for o corte mínimo, as propriedades são satisfeitas imediatamente. Caso isto não aconteça, a cada iteração o algoritmo encontra o corte mínimo $\mathcal{C}$ e compara sua capacidade com a capacidade do corte atual, $cap(U, S \setminus U)$.

Se $cap(\mathcal{C}) < cap(U, S \setminus U)$, seja U' o conjunto de vértices de U que pertencem ao mesmo lado que t em $\mathcal{C}$. Temos que $U' \subsetneq U$, já que se $U' = U$ um lado do corte seria vazio (o que não pode ocorrer) ou s estaria do lado oposto de $U \cup \{t\}$ (mas neste caso o corte teria capacidade igual a $cap(U, S \setminus U)$ e chegaríamos em uma contradição). A propriedade 1, portanto, é satisfeita.

Agora vamos comparar a capacidade de $\mathcal{C}$ e a capacidade de $(U, S \setminus U)$:

$$cap(\mathcal{C}) = cap(\{s\}, U') + cap(U \setminus U', U') + cap(U' \setminus U, \{t\}).$$

Mas

$$cap(\{s\}, U') = \sum_{v \in U'} cap(\{v\}, S \setminus U) = cap(U', S \setminus U)$$

e

$$cap(U', S \setminus U) + cap(U \setminus U', U') = cap(U', S \setminus U'),$$

então:

$$cap(\mathcal{C}) = cap(U', S \setminus U') + cap(U' \setminus U, \{t\}).$$

Como

$$cap(U' \setminus U, \{t\}) = \sum_{v \in U' \setminus U} \frac{w_\ell(v)}{w_\ell(U)} cap(U, S \setminus U) = \frac{w_\ell(U' \setminus U)}{w_\ell(U)} cap(U, S \setminus U),$$

chegamos a:

$$cap(\mathcal{C}) = cap(U', S \setminus U') + \frac{w_\ell(U' \setminus U)}{w_\ell(U)} cap(U, S \setminus U).$$

Pela última equação, como o segundo termo não é negativo, $cap(\mathcal{C}) > cap(U', S \setminus U')$. Sabendo que $cap(\mathcal{C}) < cap(U, S \setminus U)$ e $cap(U, S \setminus U) = \frac{w_\ell(U)}{w_\ell(U)} cap(U, S \setminus U)$, concluímos que:

$$\begin{aligned} cap(\mathcal{C}) &= cap(U', S \setminus U') + \tfrac{w_\ell(U' \setminus U)}{w_\ell(U)} cap(U, S \setminus U) \\ &< cap(U, S \setminus U) \\ &= \tfrac{w_\ell(U)}{w_\ell(U)} cap(U, S \setminus U), \end{aligned}$$

portanto

$$\begin{aligned} cap(U', S \setminus U') &< \tfrac{w_\ell(U)}{w_\ell(U)} cap(U, S \setminus U) - \tfrac{w_\ell(U' \setminus U)}{w_\ell(U)} cap(U, S \setminus U) \\ &= cap(U, S \setminus U) \left(\tfrac{w_\ell(U)}{w_\ell(U)} - \tfrac{w_\ell(U' \setminus U)}{w_\ell(U)} \right) \\ &= cap(U, S \setminus U) \, \tfrac{w_\ell(U')}{w_\ell(U)}. \end{aligned}$$

Logo,

$$esp((U', S \setminus U')) = \frac{cap(U', S \setminus U')}{w_\ell(U')} < \frac{cap(U, S \setminus U)}{w_\ell(U)} = esp((U, S \setminus U)),$$

demonstrando a propriedade 3 do lema.

Por fim, percebemos que, como a cada iteração o tamanho do conjunto U diminui de pelo menos um, o algoritmo termina em no máximo $|A| - 1$ iterações (porque inicialmente $U = A$). Além disso, para qualquer entrada, dada o algoritmo termina com pelo menos um vértice em U, caso em que trivialmente a propriedade 2 é alcançada. □

A etapa de refinamento consiste em tratar todos os eventos de nós inválidos dos subgrafos do nó H. O algoritmo 7 contém os passos executados na etapa de refinamento.

Algoritmo 7 Refinamento($\mathcal{P}_H$)

para todo $H_i \in \mathcal{P}_H$ **faça**
  marque H_i como incorreto;
enquanto algum H_i está incorreto **faça**
  se Pré-processamento(H_i) = falha((A, B)) **então**
    $(U, H_i \setminus U) :=$ `FluxoMaximo`$(H_i, (A, B))$;
    {Retira H_i e adiciona $\{H[U]\}$ e $\{H[H_i \setminus U]\}$ à partição $\mathcal{P}_H$}
    $\mathcal{P}_H := \mathcal{P}_H \setminus \{H_i\}$;
    $\mathcal{P}_H := \mathcal{P}_H \cup \{U\} \cup \{H_i \setminus U\}$;
    marque $\{U\}$ e $\{H_i \setminus U\}$ como incorretos;
  senão
    marque H_i como correto;
para todo $H_i \in \mathcal{P}_H$ **faça**
  se $|H_i| > 1$ **então**
    Refinamento($\mathcal{P}_{H_i}$);

Inicialmente, todos os subgrafos da partição $\mathcal{P}_H$ são marcados como incorretos. Para cada $H_i \in \mathcal{P}_H$ que está incorreto, verfica-se sua etapa de pré-processamento: caso esta retorne com sucesso, o subgrafo é marcado como correto. Caso contrário, o algoritmo 6 é usado para particionar H_i em dois subgrafos, U e $H_i \setminus U$, e estes

são marcados como incorretos. Estes passos são repetidos até que não existam subgrafos marcados como incorretos, resultando em uma partição de H em subgrafos secundários cujas etapas de pré-processamento retornam com sucesso.

7.1.3 Construção da árvore

A construção da árvore é dada pela aplicação recursiva dos algoritmos 5 e 7 para cada nó da árvore. A chamada externa é feita pelo algoritmo 8 recebendo como parâmetro o conjunto de vértices da rede.

Algoritmo 8 Constrói Árvore(G)

$\mathcal{P}_G$:=Pre-processamento($V(G), 0$);

Refinamento($\mathcal{P}_G, 0$);

7.2 Caminhos para o roteamento na árvore

Na seção 5.2 foi definido como ocorre a redistribuição de fluxo em cada nó da árvore de decomposição, definida a quantidade de fluxo que será enviada a cada vértice que pertence ao nó. Nesta seção será visto como é feita a escolha dos caminhos que irão rotear este fluxo, a fim de que o congestionamento produzido no roteamento deste fluxo seja pequeno.

O algoritmo 5 cria uma decomposição $\mathcal{P}_H$ capaz de rotear a demanda do CMCF do nó H com fração de vazão igual ou superior a z_{inf}. No entanto, após a criação dos subgrafos secundários pelo algoritmo 7, o valor desta demanda pode aumentar, visto que a criação de novos subgrafos (isto é, a retirada de subgrafos primários para inserção de secundários, que são partições dos primeiros) implica em aumento da função $w_{\ell+1}(H)$. Será mostrada uma forma de rotear fluxo dentro de H de forma a usar os caminhos calculados pelo CMCF no nó H para rotear este novo valor da demanda sem aumentar muito o congestionamento obtido anteriormente.

A cada iteração da etapa de refinamento de H, um subgrafo S da partição $\mathcal{P}_H$ é dividido em dois outros: U e $S \setminus U$. Como a etapa de refinamento inicia com subgrafos primários, ao fim do algoritmo 7 cada um deles pode ter dado origem a uma família de subgrafos secundários.

Seja H_i um subgrafo primário produzido na etapa de pré-processamento do nó H. Dado que a etapa de pré-processamento de H_i falhou, na etapa de refinamento de H este subgrafo foi particionado, dando origem a uma família $\mathcal{S}_{H_i}$ de subgrafos secundários. É possível representar as sucessivas partições do subgrafo H_i como uma árvore estritamente binária em que a raiz representa o subgrafo primário, as folhas representam os subgrafos secundários cada par de nós irmãos representam que seu pai foi particionado em alguma iteração do algoritmo. Esta árvore será chamada **árvore de refinamento**. A figura 7.1 mostra um exemplo desta árvore.

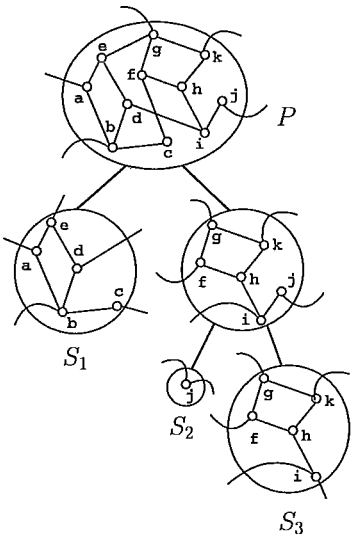


Figura 7.1: Exemplo de representação do refinamento de um subgrafo primário P em uma família de subgrafos secundários $\mathcal{S}_P = \{S_1, S_2, S_3\}$ em uma árvore de refinamento.

É importante ressaltar que esta árvore não é parte da árvore de decomposição, sendo apenas uma representação de como o particionamento de um subgrafo primário evoluiu ao longo do algoritmo.

Para rotear a nova demanda criada pelos subgrafos secundários, cada vértice de borda destes enviará sua demanda no CMCF a vértices de borda do subgrafo primário do qual se originou. O caminho que o fluxo faz é baseado na árvore de refinamento: o fluxo dos subgrafos secundários para os primários segue o caminho das folhas até a raiz e em cada nó X que possui filhos X' e $X \setminus X'$ na árvore, $|X'| \leq |X \setminus X'|$, os vértices de borda de X recebem a demanda criada na subárvore enraizada por este nó, de tal forma que:

- Os vértices de borda de X que também pertençam a X' recebem a demanda criada pelo corte $(X', X \setminus X')$; isto garante que é possível usar os caminhos calculados pelo fluxo máximo no vértice X (algoritmo 6);

- Os demais vértices de borda de X já contém o resto do fluxo que está subindo na árvore (já que também são vértices de borda em X' e $X \setminus X'$) e, por isto, não enviam fluxo a nenhum outro.

Para ilustrar o uso da árvore no envio de fluxo entre subgrafos primários e secundários, segue um exemplo:

Exemplo 7.2. *A figura 7.2 é um exemplo de uma árvore onde os vértices de borda do subgrafo primário estão marcados em vermelho e em cada nó as arestas que fazem parte do corte estão marcadas em azul. Para simplificar o exemplo, apenas as capacidades das arestas de borda ou que fazem parte de cortes estão representadas.*

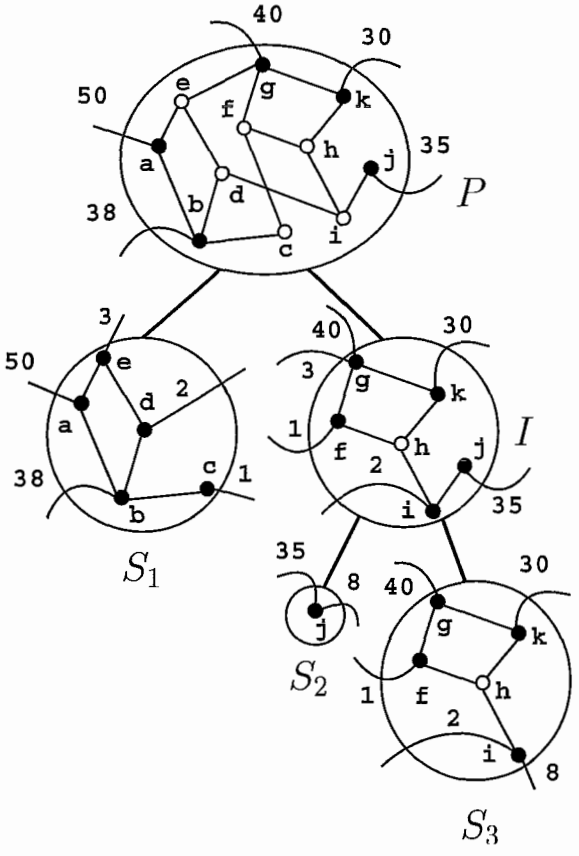


Figura 7.2: Representação em árvore da transformação de um subgrafo primário P. Os vértices de borda de P estão marcados em vermelho e as arestas do corte em cada nível, de azul.

A demanda do CMCF usando $\mathcal{P}_H$ contendo subgrafos secundários em um vértice v é igual a $\frac{w_{\ell+1}(v)}{w_{\ell+1}(H)}$, onde H é o nó da árvore de decomposição ao qual os subgrafos secundários pertencem. Como $w_{\ell+1}(H)$ é constante durante a execução deste algoritmo, já que a partição de H não irá se modificar durante este processo, este valor será ignorado e o exemplo será dado somente em função de $w_{\ell+1}(v)$. Portanto, em S_1, os vértices a, b, c, d e e possuem demandas iguais a 50, 38, 1, 2, 3, respectivamente. Em S_2, o vértice j tem uma demanda de 43 e em S_3, as demandas dos vértices f, g, i e k são, respectivamente, 1, 40, 10 e 30.

O fluxo inicia nas folhas da árvore. Na passagem de fluxo de S_2 e S_3 para I, a demanda criada no corte irá para os vértices de borda de I que também pertençam a S_2 (o menor lado do corte), ou seja, uma demanda de 8 irá para o vértice j. Os demais vértices não enviarão sua demanda a nenhum outro, pois eles também são vértices de borda em I. Em I, os vértices f, g, i, j e k receberam (ou permaneceram) com fluxo igual a 1, 40, 2, 51 ($35+2\times 8$, pois ele recebe o fluxo do corte na passagem de S_3 para I) e 30.

Na passagem de I e S_1 para P, os vértices a e b receberão a demanda criada no corte (S_1, I). A quantidade que cada um deles receberá dependerá dos caminhos escolhidos no fluxo máximo em P, porém sabe-se que toda a demanda do corte (isto é, uma demanda de $2(3+2+1) = 12$) irá para estes vértices. Como a e b são vértices de borda e o fluxo em g, j e k não muda (e eles também são vértices de borda em P), toda a demanda dos subgrafos secundários foi enviada para os vértices de borda do subgrafo primário do qual se originaram.

Portanto, para rotear fluxo entre dois vértices de H, os passos usados são os seguintes:

1. Rotear fluxo dos subgrafos secundários para seus respectivos subgrafos primários, usando o método apresentado;
2. Rotear fluxo entre os subgrafos primários, usando os caminhos da solução do CMCF no nó H;
3. Rotear novamente o fluxo dos subgrafos primários para os subgrafos secundários. Este passo é análogo ao primeiro.

Lembrando que, conforme visto na seção 5.2, pode haver duas redistribuições de fluxo em cada nó H do nível ℓ da árvore de decomposição (uma redistribuição em função de w_ℓ e outra em $w_{\ell+1}$), estes três passos podem ocorrer no máximo duas vezes em uma passagem de fluxo que corresponda ao uso do nó H no caminho feito entre duas folhas da árvore.

7.3 Análise do algoritmo

7.3.1 Complexidade de tempo

O algoritmo de partição é composto das etapas de pré-processamento e refinamento para cada nó da árvore. As demonstrações de que estas duas etapas terminam em tempo polinomial estão nos lemas 7.3 e 7.4.

Lema 7.3. *O algoritmo 5 termina em tempo polinomial.*

Demonstração. O algoritmo 5 termina quando a fração de vazão z_{inf} é alcançada no CMCF de H ou uma falha é retornada. O número de passos executados até que um destes eventos ocorra é determinado pelo comando `enquanto`. Dentro deste, o número de repetições é igual ao número de vezes em que a partição $\mathcal{P}_H$ é modificada, visto que o caso de falha termina o algoritmo.

O objetivo do algoritmo é obter uma partição em que seja possível rotear toda a demanda do CMCF obtendo fração de vazão maior ou igual a z_{inf}. Logo, basta mostrar que a demanda do CMCF diminui a cada iteração. Somando a demanda entre cada par de vértices de H:

$$\sum_{v \in H} \sum_{u \in H} d_H(u, v) = w_{\ell+1}(H),$$

pela equação (7.1). Logo, se a demanda diminui, o valor da função $w_{\ell+1}(H) - w_\ell(H)$ também diminui, visto que $w_\ell(H)$ é constante. Como o critério de parada usado no algoritmo 5 é $w_{\ell+1}(H) - w_\ell(H) < \frac{1}{2\gamma\delta(S)} cap(A, B)$, a diminuição desta função fará com que o algoritmo pare.

A cada repetição no comando `enquanto` os vértices de A são retirados dos subgrafos da partição $\mathcal{P}_H$ e o subgrafo induzido pelo conjunto A é adicionado à partição. A retirada dos vértices de A diminui pelo menos $w_{\ell+1}(A) - w_\ell(A)$ de $w_{\ell+1}(H)$, já

que arestas que ligavam dois vértices de subgrafos distintos em A, que anteriormente eram contadas em $w_{\ell+1}(A)$, foram retiradas. Em seu lugar é colocado o subgrafo induzido pelo conjunto A, aumentando de $2cap(A,B)$ a função $w_{\ell+1}(A)$.

Visto que o algoritmo modificará a partição $\mathcal{P}_H$, vale que

$$w_{\ell+1}(A) - w_\ell(A) \geq \frac{1}{2\gamma\delta(H)}\, cap(A,B).$$

Mas como $\frac{1}{2\gamma\delta(H)} \geq 2\log\log n > 2$, $w_{\ell+1}(A) - w_\ell(A) > 2cap(A,B)$. Pode-se ver então que a demanda do CMCF aumenta de $2cap(A,B)$ com a inclusão do novo subgrafo contendo todos os vértices de A, mas diminui mais do que este valor na retirada dos subgrafos anteriores. Dado que a demanda começa em um valor polinomial e diminui de valores inteiros a cada iteração, o algoritmo termina em tempo polinomial. □

Lema 7.4. *O algoritmo 7 termina em tempo polinomial.*

Demonstração. O algoritmo 7 trata os eventos de nó inválido em cada subgrafo da partição de $\mathcal{P}_H$. O tratamento deste evento em cada $H_i \in \mathcal{P}_H$ leva tempo polinomial, já que consiste em achar um corte mínimo (algoritmo 6). Como cada H_i pode gerar no máximo $|H_i|$ eventos de nó inválido (caso em que H_i é dividido em vários subgrafos que também produzem eventos de nó inválido), a execução de `Refinamento`$(\mathcal{P}_H)$ termina em tempo polinomial. □

A altura da árvore é $O(\log n)$, visto que cada subgrafo primário produzido no algoritmo 5 possui tamanho de no máximo metade do tamanho de H e os subgrafos secundários possuem sempre tamanho menor ou igual ao primário do qual se originaram. Portanto, o número de eventos de nó inválido é $O(n \log n)$ durante toda construção da árvore. Conclui-se, portanto, que a árvore é construída em tempo polinomial.

7.3.2 Razão de competitividade

Como visto na seção 7.2, uma aresta transporta fluxo em duas ocasiões: no fluxo máximo e nos caminhos do CMCF em cada nó. Vamos calcular nos lemas a seguir a carga relativa que cada um destes fluxos pode produzir em uma aresta da rede.

Lema 7.5. *O roteamento de fluxo dos subgrafos secundários para o seu subgrafo primário produz uma carga relativa de no máximo 2 para qualquer aresta usada na solução do problema de fluxo máximo.*

Demonstração. Seja $\mathcal{T}$ a árvore definida na seção 7.1.2 para um subgrafo primário P e sua família $\mathcal{S}_P$ de subgrafos secundários. Vamos convencionar que o filho esquerdo de um nó desta árvore é sempre menor que seu irmão, isto é, se um nó X é dividido em X' e $X \setminus X'$ com $|X'| \leq |X \setminus X'|$, X' será o filho esquerdo de X em $\mathcal{T}$.

Vamos definir a **altura esquerda de um nó** X na árvore como o número de nós que são filhos esquerdos existentes no caminho entre a raiz da árvore e X, mais uma unidade. A raiz, portanto, tem altura esquerda igual a um e todo nó da árvore possui a mesma altura esquerda que seu filho direito. A **altura esquerda de um vértice** v é igual a altura esquerda do nó em que ele aparece pela primeira vez como vértice de borda.

A quantidade de demanda relativa à transformação dos subgrafos primários em secundários será calculada por uma análise amortizada: cada vértice de borda do subgrafo primário receberá a demanda criada pelos cortes da árvore de refinamento. Isto é equivalente a dizer que as arestas de borda do subgrafo primário pagarão pelas arestas de cada corte existente na árvore de refinamento.

A análise da quantidade de demanda que uma aresta de borda receberá será feita baseada em sua capacidade. Seja $\phi(j)$ a quantidade de demanda por unidade de capacidade que será enviada às arestas de borda de um vértice v cuja altura esquerda seja no máximo j.

Mostraremos por indução que

$$\phi(j) \leq \prod_{i=j}^{\log n} \left(1 + \frac{1}{i \log\log n}\right). \tag{7.2}$$

Na base da indução, temos que um vértice de borda pode ter altura esquerda igual a uma folha da árvore. Calculando a maior altura esquerda possível, percebemos que o subgrafo primário que é raiz da árvore foi construído pelo algoritmo 5 e, portanto, a raiz possui no máximo $\frac{n}{2}$ vértices. Além disso, como cada nó esquerdo possui no máximo metade dos vértices de seu pai, já que ele é menor que seu irmão. Portanto, uma folha não poderá ter altura esquerda maior que $\log \frac{n}{2} + 1 = \log n - 1 + 1 = \log n$. Logo, na base da indução, $\phi(\log n) = 1$ e, portanto, a equação (7.2) vale.

Supondo que (7.2) vale para todo $i \geq j + 1$, seja X um vértice cuja altura esquerda seja igual a j, que possua filho esquerdo X' e filho direito $X \setminus X$. Seja v um vértice de borda que pertence a X. Temos que ele receberá a demanda gerada

nas extremidades do corte $(X', X \setminus X')$, caso também pertença a X'.

O lema abaixo demonstra que o corte mais esparso aproximado retornado na etapa de pré-processamento possui capacidade menor ou igual a $2\gamma\delta(X)w_\ell(X')$.

Lema 7.6. *O corte mais esparso (A, B) retornado na falha da etapa de pré-processamento em um subgrafo S (algoritmo 5) é tal que $\frac{cap(A,B)}{w_\ell(A)} \leq 2\gamma\delta(S)$.*

Demonstração. Como a etapa de pré-processamento de S retornou falha, também sabemos que

$$w_{\ell+1}(A) - w_\ell(A) < \frac{1}{2\gamma\delta(S)} cap(A, B). \tag{7.3}$$

Sendo (A, B) um corte mais esparso aproximado, temos que

$$\begin{aligned} esp((A, B)) &\leq \alpha \log n \\ esp((A, B)) &= \gamma\delta(S) \\ \tfrac{cap(A,B)}{w_{\ell+1}(A)} &< \gamma\delta(S) \end{aligned}$$

e chegamos a $w_{\ell+1}(A) > \frac{cap(A,B)}{\gamma\delta(S)}$. Substituindo este resultado em (7.3):

$$\frac{cap(A, B)}{\gamma\delta(S)} - w_\ell(A) < w_{\ell+1}(A) - w_\ell(A) < \frac{cap(A, B)}{2\gamma\delta(S)}$$

Logo,

$$\begin{aligned} -w_\ell(A) &< \frac{cap(A, B)}{2\gamma\delta(S)} - \frac{cap(A, B)}{\gamma\delta(S)} \\ w_\ell(A) &> \frac{cap(A, B)}{\gamma\delta(S)} - \frac{cap(A, B)}{2\gamma\delta(S)} \\ &= \frac{2\, cap(A, B) - cap(A, B)}{2\gamma\delta(S)} \\ &= \frac{cap(A, B)}{2\gamma\delta(S)} \end{aligned}$$

Portanto, $\frac{cap(A,B)}{w_\ell(A)} < 2\gamma\delta(S)$. □

De acordo com o lema 7.1, os cortes produzidos pelo algoritmo 6 possuem capacidade menor ou igual aos produzidos pelo algoritmo 5. Logo, o resultado do lema 7.6 também vale para $(X', X \setminus X')$. Temos, portanto, que $cap(X', X \setminus X') < 2\gamma\delta(X)w_\ell(X)$.

Pelas arestas do corte $(X', X \setminus X')$, v receberá no máximo a $cap(X', X \setminus X)$, adicionado com o fluxo que já chegou a estes vértices vindo das folhas da árvore. Para calcular a quantidade de fluxo que já foi enviada a estes vértices, podemos ver

que se X possui altura esquerda igual a j, X' possui altura esquerda igual a $j+1$ e $X \setminus X$, também possui j. Logo, os vértices de X' receberam no máximo $\phi(j+1)$ e os de $X \setminus X'$, $\phi(j)$.

Portanto, temos:

$$\phi(j) \leq \phi(j+1) + 2\gamma\delta(X)\phi(j) + 2\gamma\delta(X)\phi(j+1).$$

Como $\phi(j+1) \geq \phi(j)$:

$$\begin{aligned} \phi(j) &\leq \phi(j+1) + 2\gamma\delta(X)\phi(j) + 2\gamma\delta(X)\phi(j+1) \\ &\leq \phi(j+1) + 4\gamma\delta(X)\phi(j) \\ \phi(j)(1 - 4\gamma\delta(X)\phi(j)) &\leq \phi(j+1) \\ \phi(j) &\leq \tfrac{\phi(j+1)}{(1-4\gamma\delta(X))}. \end{aligned}$$

Mas

$$1 - 4\gamma\delta(X) = 1 - 4\gamma \frac{1}{8\gamma \log(\frac{|P|}{|H|}) \log\log n},$$

e como $\frac{1}{2\gamma \log(\frac{|P|}{|H|}) \log\log n} < \frac{1}{2}$, temos que $1 - 4\gamma\delta(X) \geq \frac{1}{2}$.

Portanto,

$$\phi(j) \leq \phi(j+1)(1 + 8\gamma\delta(X)).$$

Como cada filho esquerdo tem no máximo metade do tamanho de seu pai e existem $j-1$ filhos esquerdos no caminho até X, temos que $|X| \leq \frac{|P|}{2^j}$, onde P é o subgrafo primário que está na raiz da árvore. Logo, $\log \frac{|X|}{|P|} \geq j$ e temos $\delta(X) \leq \frac{1}{8\gamma j \log\log n}$. Substituindo este valor na equação anterior:

$$\begin{aligned} \phi(j) &\leq \phi(j-1)\left(1 + \frac{8\gamma}{8\gamma j \log\log n}\right) \\ &\leq \left(\prod_{i=j+1}^{\log n} 1 + \frac{1}{i \log\log n}\right)\left(1 + \frac{1}{j \log\log n}\right) \\ &= \prod_{i=j}^{\log n}\left(1 + \frac{1}{i \log\log n}\right). \end{aligned}$$

Para calcular a demanda enviada até a raiz da árvore, temos

$$\phi(1) \leq \prod_{i=1}^{\log n}\left(1 + \frac{1}{i \log\log n}\right). \tag{7.4}$$

O lema 7.7 calcula a fórmula fechada do produtório acima.

Lema 7.7. *Para qualquer* $c \leq \frac{1}{\log x}$, *vale que* $\prod_{i=1}^{x}(1 + c/i) \leq 2$.

Demonstração. Seja $y = \prod_{i=1}^{x}(1 + c/i) \leq 2$. Calculando o logaritmo natural em ambos os lados:

$$\begin{aligned} \ln y &= \ln\left(\prod_{i=1}^{x}(1 + c/i)\right) \\ &= \ln(1 + c) + \ln(1 + c/2) + \ldots + \ln(1 + c/x) \\ &= \sum_{i=1}^{x} \ln(1 + c/i) \end{aligned}$$

Mas $\ln(1 + a) \leq a$, portanto, $\sum_{i=1}^{x} \ln(1 + c/i) \leq \sum_{i=1}^{x} c/i$. Chegamos a

$$\begin{aligned} \ln y &\leq \sum_{i=1}^{x} c/i \\ &= c \sum_{i=1}^{x} 1/i \\ &= c \ln x. \end{aligned}$$

Como $\ln y \leq c \ln x \leq \frac{\ln x}{\log x} = \ln 2$, concluímos que $y \leq e^{\ln 2} = 2$. □

Substituindo o resultado do último lema na equação (7.4) com $x = \log n$ e $c/i = 1/(i \log\log n)$, temos que $\phi(1) \leq 2$.

Usando os caminhos calculados no algoritmo 6, temos congestão igual a um quando enviamos um fluxo unitário com origem nos vértices que são extremidades de arestas do corte com destino aos vértices de borda do conjunto. Pelo resultado deste último lema, o fluxo enviado para os vértices de borda do subgrafo primário será de no máximo 2 unidades. A carga relativa de qualquer aresta no roteamento deste fluxo será, portanto, igual a 2. □

Teorema 7.8. *A carga relativa em qualquer aresta no esquema de roteamento dado pelo algoritmo é* $O(\log^2 n \log\log n)$.

Demonstração. No algoritmo, cada aresta pode ser usada em dois tipos de roteamento: na solução do CMCF e na solução do problema de fluxo máximo.

Pelo lema 7.5, o roteamento entre os subgrafos secundários e primários usando a solução do problema de fluxo máximo produz carga relativa de no máximo 2 em qualquer aresta. No entanto, cada transferência de fluxo entre subgrafos primários e secundários pode ocorrer no máximo quatro vezes dentro de um mesmo nó H do nível ℓ da árvore de decomposição, conforme visto na seção 7.2. Portanto, a

carga relativa que uma aresta pode ter quando o fluxo está em H da árvore de decomposição é igual a $4 \cdot 2 = 8$.

Mas uma aresta pode pertencer a no máximo $\log n$ nós, já que cada aresta pertence a somente um ramo da árvore de decomposição (os nós irmãos são partições em vértices de seu pai) e, como dito na seção 7.3.1, a árvore possui altura $\log n$. Concluímos então que uma aresta pode participar da solução do fluxo máximo de no máximo $\log n$ conjuntos, resultando em uma carga relativa de no máximo $8 \log n$.

No CMCF de um nó H da árvore de decomposição, uma aresta pode ter carga relativa de no máximo $\frac{1}{z_H}$, onde z_H é a fração de vazão obtida neste CMCF. Mas como $z_H \geq \frac{\log n}{\delta(H)}$ para todo nó H da árvore de decomposição (visto que sua etapa de pré-processamento retornou com sucesso), concluímos que a carga relativa de uma aresta no CMCF de um nó H é no máximo $\frac{\delta(H)}{\log n}$.

Uma aresta $e = (u, v)$ participa do CMCF sempre que u e v pertencem a um mesmo nó da árvore de decomposição. Como a cada nível os nós irmãos são partições em vértices do nó pai, uma aresta pertence somente a um ramo da árvore a partir da raiz. Seja t o número de nós ao qual uma aresta pertence, ou seja, seja t a distância entre a raiz da árvore e o nó de nível mais alto onde u e v ainda pertencem ao mesmo nó. A carga relativa de e dada pelo roteamento de todos os CMCFs dos nós aos quais ela pertence é igual a:

$$\begin{aligned} RLd(e) = \sum_{x=1}^{t-1} \frac{\log n}{\delta(S_x)} &= (\log n) \sum_{x=1}^{t-1} \frac{1}{\delta(S_x)} \\ &= (\log n) \sum_{x=1}^{t-1} 8\gamma \log \frac{|S_{x-1}|}{|S_x|} \log\log n \\ &= 8\gamma (\log n) \log\log n \sum_{x=1}^{t-1} \log \frac{|S_{x-1}|}{|S_x|} \\ &= 8\gamma (\log n) \log\log n \log \left(\frac{|S_0|}{|S_1|} \cdot \ldots \cdot \frac{|S_{t-1}|}{|S_t|} \right), \end{aligned} \quad (7.5)$$

onde S_i é i-ésimo nó a partir da raiz no ramo da árvore ao qual a aresta e pertence.

Simplificando o produtório:

$$\frac{|S_0|}{|S_1|} \frac{|S_1|}{|S_2|} \cdots \frac{|S_{t-1}|}{|S_t|} = \frac{|S_0|}{|S_t|}.$$

Como $|S_0| \leq n$ e $|S_t| \geq 1$, $\frac{|S_0|}{|S_t|} \leq n$. Calculando o log de ambos os lados, chegamos a $\log \frac{|S_0|}{|S_t|} < \log n$.

Utilizando este resultado na equação (7.5), temos:

$$RLd(e) < 8\gamma(\log^2 n)\log\log n.$$

Mas a demanda em cada vértice do CMCF pode no máximo dobrar, como resultado do lema 7.5 e por isso $RLd(e) < 16\gamma(\log^2 n)\log\log n$.

Como uma aresta pode participar tanto de fluxos máximos quanto de CMCFs, a carga relativa total é dada pela contribuição destes dois fluxos:

$$RLd(e) = 8\log n + 16\gamma\log^2 n\log\log n \qquad \forall e \in E.$$

Portanto, o congestionamento é $O(\log^2 n\log\log n)$. □

Capítulo 8

Conclusão

Nesta dissertação foi feita uma análise de alguns dos principais resultados existentes na literatura de algoritmos desinformados visando minimizar o congestionamento. Além da exposição dos limites inferiores para este problema, foi dada especial atenção aos algoritmos de Hajiaghayi *et al.* [22, 32] para grafos com capacidades nos vértices e para grafos direcionados e aos algoritmos de Bienkowski *et al.* [14] e Harrelson *et al.* [15] para grafos com capacidades nas arestas.

No capítulo 2 são dadas as definições usadas ao longo do texto, bem como a formalização do problema do roteamento desinformado e a apresentação de problemas correlatos ao mesmo.

No capítulo 3 são dadas algumas demonstrações de limites inferiores para a razão de competitividade de algoritmos desinformados. Estes limites são: $O(\log n)$ para redes não-direcionadas com capacidades nas arestas; $O(\sqrt{n})$ para os demais casos (redes não-direcionadas com capacidades nos vértices e redes direcionadas com capacidades nas arestas ou vértices).

O capítulo 4 é dedicado a algoritmos desinformados para redes com capacidades nos vértices e para redes direcionadas, apresentando seu funcionamento e analisando sua razão de competitividade. Para ambos os casos são apresentados algoritmos de tempo polinomial publicados em [22] que, até onde sabemos, é o único trabalho na literatura do problema do roteamento desinformado que aborda especificamente estas duas variações do problema. O algoritmo desinformado para redes com capacidades nos vértices possui razão de competitividade $O(\Delta \log^2 n \log\log n)$, onde Δ é o maior grau dos vértices, e o algoritmo para redes direcionadas, $O(\sqrt{|\mathcal{K}|}\, n^{1/4} \log n)$,

onde $|\mathcal{K}|$ denota o número de *commodities*.

O algoritmo de Hajiaghayi *et al.* [22] para redes com capacidades nos vértices usa uma transformação da entrada do problema, modificando-o para tornar o rede com capacidade nas arestas. A resolução deste problema modificado é feita usando o algoritmo desinformado de [15], para então fazer uma correspondência entre a solução na rede com capacidade nas arestas para uma solução na rede com capacidade nos vértices. Este algoritmo possui razão de competitividade de $O(\alpha \log n \log\log n)$; no entanto, a conversão de uma solução fornecida por este algoritmo para uma solução para uma rede com capacidade nos vértices aumenta em um fator de Δ, resultando em uma razão de competitividade de $O(\Delta\, \alpha \log n \log\log n)$.

Já o algoritmo para redes direcionadas publicado neste mesmo trabalho divide as *commodities* por tamanho de sua demanda e produz, para cada faixa de demandas entre potências de 2 consecutivas, um esquema de roteamento diferente. Em cada um destes esquemas consegue-se um congestionamento de $O(\sqrt{\alpha\,|\mathcal{K}|})$ e cada aresta pode participar de no máximo $O(\log n)$ esquemas, resultando em uma razão de competitividade de $O(\sqrt{\alpha\,|\mathcal{K}|}\log n)$.

Note que um grafo não-direcionado com capacidades nos vértices pode ser facilmente transfomado em um grafo direcionado com capacidades nas arestas; logo, o algoritmo para digrafos pode ser usado também para grafos não-orientados com capacidades nas arestas através de uma conversão da entrada do problema.

O capítulo 5 introduz o conceito de árvore de decomposição, relacionado com uma decomposição hierárquica de uma rede. O método de obter um esquema de roteamento desinformado usando uma árvore de decomposição da rede, também descrito neste capítulo, é usado pelos algoritmos apresentados nos capítulos 6 e 7.

O capítulo 6 é dedicado ao estudo do algoritmo de Bienkowski *et al.* [14], que possui razão de competitividade $O(\log^4 n)$. Este algoritmo possui muitas similaridades com o de Räcke [9]. Neste último, o algoritmo é exponencial com razão de competitividade $O(\log^3 n)$, enquanto que no primeiro, apesar da razão de competitividade ser maior, o algoritmo é polinomial.

A idéia principal do algoritmo é construir uma árvore de decomposição e usá-la como base do roteamento que será feito no grafo. A cada passo dado na árvore existe uma redistribuição entre os vértices do grafo para simular a entrada de fluxo em um

nó da árvore. Cada redistribuição produz um congestionamento de no máximo $O(\frac{1}{\log^3 n})$ e, como a árvore de decomposição tem altura logarítmica, faz-se $O(\log n)$ redistribuições, resultando em uma razão de competitividade de $O(\log^4 n)$.

A construção da árvore de decomposição é feita por sucessivas partições do conjunto de entrada até que seja obtida uma configuração em que cada subgrafo de um nó da árvore consiga se comunicar de maneira satisfatória, isto é, quando a partição faz com que o CMCF daquele nó atinja certa fração de vazão. Além disso, esta partição precisa garantir que cada nó da árvore de decomposição possua tamanho menor do que uma fração de seu conjunto pai, conseguindo assim uma altura logarítmica.

No capítulo 7 é estudado o algoritmo desinformado de Harrelson *et al.* [15], que possui razão de competitividade $O(\log^2 n \log\log n)$. Este algoritmo, assim como os publicados por Räcke [9] e Bienkowski *et al.* [14], faz seu esquema de roteamento baseado em uma árvore de decomposição.

A construção da árvore de decomposição neste artigo é feita em dois estágios, produzindo dois tipos de subgrafos na decomposição: os primários e os secundários. Os subgrafos secundários são os nós da árvore de decomposição, enquanto os primários representam apenas estágios intermediários para roteamento do fluxo. Com este refinamento dos conjuntos produzidos, a altura da árvore de decomposição é menor do que nos métodos anteriores, chegando a uma razão de competitividade de $O(\log^2 n \log\log n)$.

Esta dissertação teve como objetivo principal expor com mais detalhes as análises existentes destes algoritmos e as provas de limites inferiores para o problema do roteamento desinformado, além de unificar a notação dos trabalhos abordados. Além disso, este trabalho também introduz problemas fortemente relacionados ao problema do roteamento desinformado, como o problema do corte mais esparso aproximado, CMCF e MCUP, e indica referências na literatura sobre eles.

Os trabalhos abordados nesta dissertação foram escolhidos por fornecer a construção de um roteamento usando informações sobre a estrutura do problema, como por exemplo o uso de cortes esparsos para encontrar pontos de alto congestionamento na rede e a divisão da rede em subconjuntos que contenham seu modo particular de rotear. Este tipo de abordagem fornece uma forma intuitiva de entender como os

caminhos para roteamento são calculados.

Além destes trabalhos, aos interessados no estudo deste tema recomendamos também a leitura dos artigos de Azar *et al.* [21] e Räcke [18]. O primeiro faz uma abordagem do problema através de um programa linear, obtendo o melhor algoritmo desinformado possível para a entrada fornecida. No entanto, o método utilizado torna-se inviável para redes grandes, além de sua solução não fornecer maiores informações sobre a estrutura do problema.

Já o trabalho de Räcke [18] desenvolveu um novo algoritmo para obtenção de roteamentos desinformados usando árvore de decomposição. Na verdade, o problema de roteamento desinformado é apenas um dos vários problemas cuja solução pode ser obtida como uma aplicação do novo algoritmo de decomposição hierárquica apresentado neste artigo.

A decomposição hierárquica em questão é obtida através de uma adaptação do algoritmo de Fakcharoenphol, Rao e Talwar [19] para aproximar caminhos em um grafo por caminhos em árvores. Enquanto o resultado original de Fakcharoenphol aproximava distâncias no grafo por distâncias nas árvores, o algoritmo de Räcke obtém árvores que tentam aproximar os cortes do grafo.

As folhas das árvores obtidas correspondem aos vértices do grafo original, e cada aresta das árvores têm capacidade igual a do corte que corresponde a esta aresta no grafo original. Portanto, o algoritmo obtém uma distribuição de probabilidades sobre árvores, tal que um conjunto de fluxos no grafo com congestionamento c pode ser roteado na árvore com congestionamento esperado menor ou igual a c.

Por outro lado, dado um conjunto de fluxos na árvore, com congestionamento esperado c', estes podem ser mapeados no grafo com congestionamento esperado $O(c' \log n)$.

Isto implica em uma solução para o problema de roteamento desinformado com razão de competitividade $O(\log n)$, já que podemos rotear os fluxos nas árvores com congestionamento menor ou igual ao ótimo (no grafo) de forma trivial, e estes caminhos podem ser mapeados em caminhos no grafo que apresentam um congestionamento esperado no máximo $O(\log n)$ vezes maior. Portanto, o algoritmo usando esta árvore de decomposição é ótimo, devido ao limite inferior provado no hipercubo (capítulo 3).

Por fim, a tabela 8.1 apresenta os limites inferiores e a melhor razão de competitividade conhecida até o momento para cada tipo de grafo.

Tipo de rede		Limite inferior	Razão de competitividade
Digrafos	capacidade nos vértices	$\Omega(\sqrt{n})$	$O(\sqrt{\lvert\mathcal{K}\rvert}\, n^{\frac{1}{4}} \log n)$
	capacidade nas arestas	$\Omega(\sqrt{n})$	
Grafos	capacidade nos vértices	$\Omega(\sqrt{n})$	$O(\min\{\sqrt{\lvert\mathcal{K}\rvert} \log^{\frac{3}{2}} n,$ $\Delta \log n \log\log n\})$
	capacidade nas arestas	$\Omega(\log n)$	$O(\log n)$

Tabela 8.1: Limites inferiores e superiores para o problema do Roteamento desinformado randomizado.

É interessante ressaltar que os algoritmos valem para qualquer rede que se enquadre em certa classe (direcionado ou não, capacidade nas arestas ou vértices) e que nesta análise não importam quais são as demandas. Isso acontece porque os algoritmos constróem o esquema de roteamento de forma proporcional às capacidades das arestas de um certo corte na rede, logo, a topologia exata da rede não importa, a partir do momento que este corte foi encontrado.

8.1 Problemas em aberto

Dado que o trabalho de Räcke (2008) [18] obteve um algoritmo ótimo para redes não-direcionadas com capacidades nas arestas, os principais desafios na área de roteamentos desinformados visando minimizar o congestionamento consistem em obter algoritmos com menor distância entre o limite inferior e superior para digrafos e para grafos com capacidades nos vértices.

Estas versões do problema de fato parecem ser muito mais difíceis porque o fator de aproximação α para obter um corte mais esparso aproximado é significativamente maior em comparação ao fator para grafos com capacidades nas arestas. Além disso, os principais algoritmos para a versão não-direcionada com capacidade nas arestas não fornecem nenhuma intuição para estas versões do problema, visto que até o momento não foi encontrada nenhuma forma de usar uma decomposição hierárquica de forma eficiente (isto é, que reduza o limite superior atualmente conhecido) em

redes com capacidades nas arestas ou redes direcionadas.

Para o caso especial de grafos com capacidades nos vértices, um problema interessante consiste em conseguir um algoritmo cuja razão de competitividade seja $O(\sqrt{n}\log n)$, pois os algoritmos com melhor razão de competitividade atualmente conhecidos, quando $|\mathcal{K}| = O(n^2)$ ou quando $\Delta = O(n^2)$ obtêm razão de competitividade $O(n \log n)$.

Como estes problemas parecem ser difíceis, há atualmente uma tendência na literatura de trabalhos que visam provar que as razões de competitividade obtidas para o congestionamento valem também para funções mais genéricas [20] ou publicações que trabalham em variantes do problema do roteamento desinformado, permitindo por exemplo que se conheça a distribuição da demanda que será dada como entrada [35, 36].

Referências Bibliográficas

[1] AHUJA, R. K., MAGNANTI, T. L., ORLIN, J. B., *Network flows: theory, algorithms, and applications*. 1st ed. Prentice-Hall: Upper Saddle River, 1993.

[2] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al., *Introduction to algorithms*. 2nd ed. MIT Press: Cambridge, 2001.

[3] LEIGHTON, F. T., RAO, S., "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms". In: *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pp. 422–431, IEEE, 1988.

[4] LEONARDI, S., "On-line Network Routing". In: *Online Algorithms*, v. 1442, *Lecture Notes in Computer Science*, pp. 242–267, Springer, 1996.

[5] HARSHA, P., HAYES, T. P., NARAYANAN, H., et al., "Minimizing average latency in oblivious routing". In: *SODA '08: Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 200–207, Society for Industrial and Applied Mathematics: Philadelphia, 2008.

[6] ALY, M., AUGUSTINE, J., "Online packet admission and oblivious routing in sensor networks". In: *ISAAC*, v. 4288, *Lecture Notes in Computer Science*, pp. 680–689, Springer, 2006.

[7] RÄCKE, H., ROSÉN, A., "Distributed online call control on general networks". In: *SODA '05: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 791–800, Society for Industrial and Applied Mathematics: Philadelphia, 2005.

[8] AWERBUCH, B., HAJIAGHAYI, M. T., KLEINBERG, R. D., et al., "Online client-server load balancing without global information". In: *SODA '05: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 197–206, Society for Industrial and Applied Mathematics: Philadelphia, 2005.

[9] RÄCKE, H., "Minimizing congestion in general networks". In: *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*, pp. 43–52, IEEE Computer Society: Washington, 2002.

[10] SZWARCFITER, J. L., *Grafos e algoritmos computacionais*. 2nd ed. Editora Campus: Rio de Janeiro, 1984.

[11] MACULAN, N., FAMPA, M. H. C., *Otimização linear*. 1st ed. Editora UnB: Brasília, 2006.

[12] SHAHROKHI, F., MATULA, D. W., "The maximum concurrent flow problem", *Journal of ACM*, v. 37, n. 2, pp. 318–334, 1990.

[13] RAGHAVAN, P., THOMPSON, C. D., "Randomized rounding: a technique for provably good algorithms and algorithmic proofs", *Combinatorica*, v. 7, n. 4, pp. 365–374, 1987.

[14] BIENKOWSKI, M., KORZENIOWSKI, M., RÄCKE, H., "A practical algorithm for constructing oblivious routing schemes". In: *SPAA '03: Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 24–33, ACM: New York, 2003.

[15] HARRELSON, C., HILDRUM, K., RAO, S., "A polynomial-time tree decomposition to minimize congestion". In: *SPAA '03: Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 34–43, ACM: New York, 2003.

[16] MAGGS, B. M., AUF DER HEIDE, F. M., VÖCKING, B., et al., "Exploiting locality for data management in systems of limited bandwidth". In: *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pp. 284–293, 1997.

[17] MAGGS, B. M., MILLER, G. L., PAREKH, O., et al., "Solving symmetric diagonally-dominant systems by preconditioning", Unpublished manuscript.

[18] RÄCKE, H., "Optimal hierarchical decompositions for congestion minimization in networks". In: *STOC '08: Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pp. 255–264, 2008.

[19] FAKCHAROENPHOL, J., RAO, S., TALWAR, K., "A tight bound on approximating arbitrary metrics by tree metrics". In: *FOCS '03: Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pp. 448–455, 2003.

[20] GUPTA, A., HAJIAGHAYI, M. T., RÄCKE, H., "Oblivious network design". In: *SODA '06: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 970–979, ACM: New York, 2006.

[21] AZAR, Y., COHEN, E., FIAT, A., et al., "Optimal oblivious routing in polynomial time". In: *STOC '03: Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pp. 383–388, ACM: New York, 2003.

[22] HAJIAGHAYI, M. T., KLEINBERG, R. D., LEIGHTON, T., et al., "Oblivious routing on node-capacitated and directed graphs". In: *SODA '05: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete algorithms*, pp. 782–790, Society for Industrial and Applied Mathematics: Philadelphia, 2005.

[23] MATULA, D. W., SHAHROKHI, F., "Sparsest cuts and bottlenecks in graphs", *Discrete Applied Mathematics*, v. 27, n. 1-2, pp. 113–123, 1990.

[24] SCHRIJVER, A., *Combinatorial Optimization Polyedra and Efficiency*. 1st ed., v. C. Springer-Verlag: Berlin Heidelberg, 2003.

[25] AUMANN, Y., RABANI, Y., "An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm", *SIAM Journal on Computing*, v. 27, n. 1, pp. 291–301, 1998.

[26] PLOTKIN, S. A., TARDOS, E., "Improved bounds on the max-flow min-cut ratio for multicommodity flows". In: *ACM Symposium on Theory of Computing*, pp. 691–697, 1993.

[27] EVEN, G., NAOR, J. S., SCHIEBER, B., et al., "Approximating minimum feedback sets and multi-cuts in directed graphs". In: *Proceedings of the 4th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pp. 14–28, Springer-Verlag: London, 1995.

[28] KLEIN, P. N., PLOTKIN, S. A., RAO, S., et al., "Approximation algorithms for Steiner and directed multicuts", *Journal of Algorithms*, v. 22, n. 2, pp. 241–269, 1997.

[29] BORODIN, A., HOPCROFT, J. E., "Routing, merging and sorting on parallel models of computation". In: *STOC '82: Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pp. 338–344, ACM: New York, 1982.

[30] KAKLAMANIS, C., KRIZANC, D., TSANTILAS, T., "Tight bounds for oblivious routing in the hypercube". In: *SPAA '90: Proceedings of the second Annual ACM Symposium on Parallel algorithms and architectures*, pp. 31–36, ACM: New York, 1990.

[31] BARTAL, Y., LEONARDI, S., "On-Line routing in all-optical networks". In: *ICALP '97: Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pp. 516–526, Springer-Verlag: London, 1997.

[32] HAJIAGHAYI, M. T., KLEINBERG, R. D., RÄCKE, H., et al., "Oblivious routing on node-capacitated and directed graphs", *ACM Transactions on Algorithms*, v. 3, n. 4, pp. 51–64, 2007.

[33] HAJIAGHAYI, M. T., RÄCKE, H., "An $O(\sqrt{n})$-approximation algorithm for directed sparsest cut", *Information Processing Letters*, v. 97, n. 4, pp. 156–160, 2006.

[34] HAJIAGHAYI, M. T., LEIGHTON, T., "On the max-flow min-cut ratio for directed multicommodity flows", *Theoretical Computer Science*, v. 352, n. 1, pp. 318–321, 2006.

[35] HAJIAGHAYI, M., KIM, J. H., LEIGHTON, T., et al., "Oblivious routing in directed graphs with random demands". In: *STOC '05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pp. 193–201, ACM: New York, 2005.

[36] HAJIAGHAYI, M. T., KLEINBERG, R. D., LEIGHTON, T., et al., "New lower bounds for oblivious routing in undirected graphs". In: *SODA '06: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm*, pp. 918–927, ACM: New York, 2006.